IpMorph : unification de la mystification de prise d'empreinte

Guillaume Prigent[1], Florian Vichot[1] et Fabrice Harrouet[2]
guillaume.prigent(@)diateam.net
florian.vichot(@)diateam.net
harrouet(@)enib.fr

[1] Diateam : Architectes de l'information,
41, rue Yves Collet, 29200 Brest
[2] LISyC : Laboratoire d'Informatique des Systèmes Complexes,
Technopôle Brest Iroise, 29280 Plouzané

Résumé Il existe aujourd'hui une multitude d'outils d'identification de piles TCP/IP qui permettent de reconnaître relativement aisément le système d'exploitation des cibles pressenties. Le but de cet article est de montrer que la dissimulation et la mystification d'empreinte sont possible uniformément face aux différents outils de prise d'empreinte connus. A titre pédagogique et comme preuve de concept, nous présentons IpMorph, un logiciel de contre-reconnaissance sous la forme d'une pile TCP/IP en mode utilisateur, qui assure le suivi de session et la réécriture des paquets à la volée. Nous détaillons son fonctionnement et son usage face à des outils tels que Nmap, Xprobe2, Ring2, SinFP et p0f et nous évaluons son efficacité grâce à une première implémentation technique qui couvre déjà la plupart de nos objectifs.
Note des auteurs Le logiciel IpMorph est distribué sous licence open-source GPLv3. Ce projet indépendant basé sur nos travaux précédents provient principalement d'un besoin spécifique au sein du projet de simulation d'architecture réseau «Hynesim»(marché public DGA-CELAR-SSI-AMI, http ://www.hynesim.org).

1 Introduction

Du point de vue de l'attaquant, le recueil d'information est la phase préparatoire déterminante à l'élaboration d'une stratégie offensive appropriée. L'enjeu à ce stade est de disposer du maximum d'informations fiables et recoupées afin d'orienter le processus de pénétration et de maximiser la pertinence des actions menées. A contrario, du point de vue du défenseur, la capacité à minimiser la visibilité du périmètre critique de son infrastructure est un facteur limitant ou du moins ralentissant pour les attaques dont il pourrait être victime. Face à la montée des actions de reconnaissance (explosion des scans de ports, utilisations distribuées massives et automatisées d'outils d'analyse de vulnérabilité ou d'agents de prise d'empreinte, codes malveillants adaptatifs à la plate-forme cible) il devient primordial de pouvoir réduire le périmètre informationnel visible de l'infrastructure à protéger. Au niveau réseau, s'il est quasi impossible de dissimuler les services minimum (adresses IP, ports TCP) frontaux supports de

l'interconnexion de l'infrastructure, il est pourtant possible d'imaginer une falsification suffisamment réaliste afin de mystifier les actions de reconnaissances distantes.

Notre constat est le suivant :

> "*Si une machine peut falsifier son identité face aux outils de reconnaissance et usurper celle d'un système moins intéressant, cette dernière machine minimise l'attrait de l'attaquant et perturbe la pertinence des attaques ciblées à sa nature apparente.*"

Le domaine de l'OSFP[1] a déjà été largement discuté dans de nombreux articles [1] au cours des dix dernières années et il existe à ce jour différentes techniques d'identification du comportement des piles TCP/IP des systèmes d'exploitation. L'objet de cet article n'est donc pas de détailler à nouveau ces méthodologies du point de vue de l'attaquant (la personne ou l'entité qui cherche à identifier la personnalité de la cible). Notre propos, de manière pédagogique et du point de vue du défenseur, est de montrer qu'il est possible de réaliser un composant logiciel capable d'empêcher et de perturber énormément cette identification distante, qu'elle soit active ou passive. Cet article se veut donc être à la fois une réflexion pragmatique sur ce qu'il convient de faire pour mystifier la plupart des outils connus à ce jour, et la présentation de l'architecture de l'outil IpMorph que nous développons pour mystifier ces identifications.

Puisque nous souhaitons prendre en compte l'éventail large des différents outils implémentant les différentes techniques d'OSFP (Fig.1), nous commençons dans le paragraphe suivant par en exposer rapidement les deux principales méthodes afin d'introduire la «palette»d'outils que nous souhaitons mystifier.

2 Rappels

Dans le domaine de la prise d'empreinte distante de système d'exploitation, il existe communément deux méthodes, chacune possède ses avantages et ses inconvénients.

2.1 Prise d'empreinte active («Active FingerPrinting»)

Cette méthode est considérée comme active puisqu'il s'agit principalement de contacter la cible par l'envoi de stimuli particuliers (paquets IP et TCP forgés pour l'occasion) et d'analyser les réponses de celle-ci par rapport à une base de connaissances déjà élaborée (base de signatures). Le choix des stimuli et donc des paquets envoyés depuis la source est fait suivant l'axiome que diverses implémentations

[1] OSFP : Operating System FingerPrinting (Prise d'empreinte de système d'exploitation)

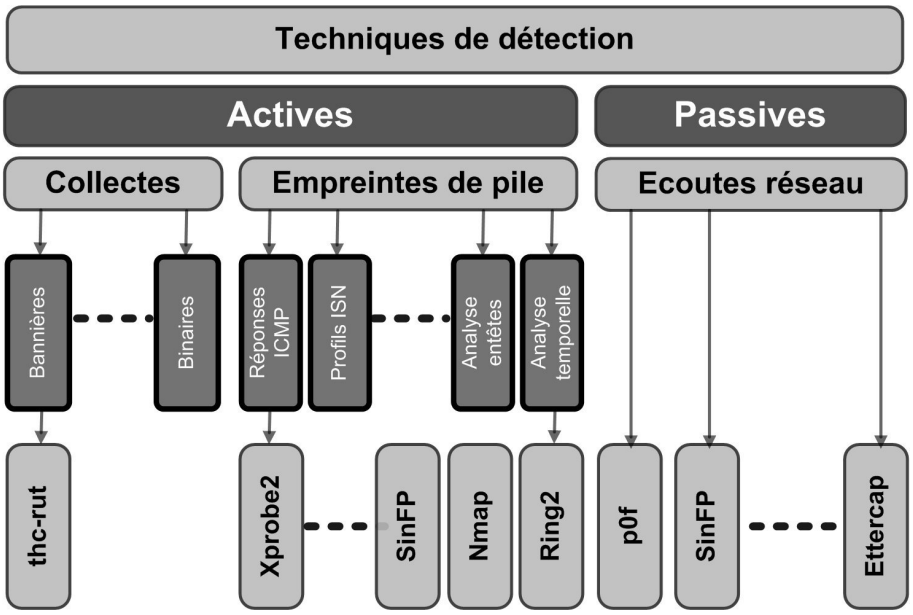

Fig. 1. Classification de quelques techniques illustrées par certains outils associés.

de piles TCP/IP (et donc de systèmes d'exploitation) peuvent répondre différemment en fonction des choix d'implémentation (voire même de conception) [3] [4] [6].

Inconvénients majeurs : Du fait de sa nature même, la prise d'empreinte active laisse de nombreuses traces sur le système sondé ainsi que tout au long du chemin emprunté (équipements filtrants, détection d'intrusion, ...) ce qui rend sa détection et son analyse éventuelle plus aisées. Le modus operandi de ce type d'outils est reconnaissable au point d'en faire des entrées des bases de connaissance de la plupart des sondes NIDS (Network Intrusion Detection System). C'est d'ailleurs ce «bruit» caractéristique et ce trafic particulier que nous utiliserons bien souvent afin d'identifier les stimuli spécifiques de ce type de fonctionnement. De plus, la possibilité d'envoyer ces paquets à la cible et plus généralement la réunion des conditions requises nominales d'expérimentation pour ce type de tests «intrusifs» ne sont pas toujours possible, qu'ils soient effectués à distance ou sur un réseau local (filtrage des paquets par exemple). Cette limitation importante est fonction des outils considérés.

Avantages majeurs : La prise d'empreinte active permet de solliciter à sa convenance la cible et ainsi de tester une vaste panoplie de réponses. Il est ainsi possible d'effectuer des tests adaptatifs (en fonction des réponses précédentes) de différenciation ou de variance pour préciser l'identification d'une cible. Correctement effectués, ces

types de tests permettent d'obtenir rapidement des résultats probants (parfois et en fonction des outils, seuls quelques paquets judicieux suffisent).

Outils : Nmap, Xprobe2, Ring, SinFP, GFI's Languard Network Security Scanner, Queso,...

2.2 Prise d'empreinte passive («Passive FingerPrinting»)

La prise d'empreinte passive effectue seulement une analyse des flux issus de la cible sans jamais la solliciter directement par l'envoi de paquets. La plupart du temps, cette analyse porte sur du trafic «légitime» (non déviant par rapport à un usage normal) entre la cible et un hôte particulier du réseau local ou distant. Du fait de ce fonctionnement, cette méthode présente l'avantage indéniable d'une furtivité de reconnaissance pour l'attaquant qui procède de la sorte. Certains équipements de protection réseau (comme les pare-feux ou les systèmes de détection d'intrusion réseau) utilisent déjà cette méthode passive pour corréler les attaques potentielles par rapport aux systèmes d'exploitation des machines à protéger [2] [4] [6].

Inconvénients majeurs : La reconnaissance passive d'un flux légitime impose de posséder suffisamment de paquets pour effectuer les analyses d'identification des caractéristiques. Cette collecte peut s'avérer longue et fastidieuse et rien n'assure qu'au cours du temps les protagonistes de ces communications capturées ne changent pas. Cet aspect temporel est d'autant plus pénalisant qu'il est bien souvent difficile d'accéder aux flux entre les machines visées (si l'on exclut le cas trivial d'être sur l'équipement filtrant, sur une sonde en coupure ou directement sur un brin réseau). Les techniques à mettre en œuvre afin de capturer les échantillons d'analyse sont dès lors les mêmes que lors d'attaques dites *«Man in the middle»*.

Avantages majeurs : La prise d'empreinte passive n'accède qu'en lecture au flux. La problématique de détection de ce type de comportement est exactement la même que lors d'une capture habituelle du type *«sniffing»* en mode *«promiscuous»* sur un réseau. De plus, la cible d'analyse n'est jamais contactée directement. De ce fait, la prise d'empreinte passive est extrêmement furtive.

Outils : p0f, SinFP, Ettercap, Satori...

2.3 Synthèse des méthodologies de prise d'empreinte

Les avantages et inconvénients des deux méthodologies présentées ci-dessous sont à pondérer en fonction des outils.

Il s'agit d'un tableau de synthèse généraliste et d'une classification issus de notre état de l'art. En particulier, certains outils actifs sont caractéristiquement «bruyants» (cas de Nmap avec l'envoi de nombreux paquets volontairement malformés) alors que

	Prise d'empreinte active	**Prise d'empreinte passive**
Avantages	Rapide Précise dans l'identification	Furtive
Inconvénients	Souvent très «bruyante» Volumineuse en paquets Sensibilité aux conditions de tests	Longue Peu discriminante

Fig. 2. Avantages et inconvénients des deux méthodologies.

d'autres sont beaucoup plus discrets et ne portent que sur l'envoi de quelques paquets standards (cas de SinFP par exemple).

Dans le cadre d'IpMorph, nous souhaitons être capable d'assurer la mystification d'un maximum d'outils, c'est pourquoi nous avons intégré à IpMorph aussi bien des outils de prise d'empreinte active (Nmap, Xprobe2, SinFP, Ring2) que des outils qui fonctionnent en mode passif (p0f, SinFP). Pour chacun de ces outils, nous effectuons dans la section 5 un focus technique sur la spécificité de leur intégration respective dans la conception de notre outil.

3 Etat de l'art de la mystification

Il existe déjà de nombreuses solutions [7] qui tentent, chacune à leur manière, de perturber l'identification exacte de la machine à protéger. La section suivante énumère la plupart des approches existantes et les classifient suivant leur fonctionnement intrinsèque.

3.1 Classification des solutions de perturbation d'identification

Nous considérons qu'il existe trois principales catégories d'approches qui mettent en œuvre des techniques de dissimulation. Celles qui portent sur le filtrage et la réécriture de paquets, celles qui s'attachent à modifier la configuration native de la pile TCP/IP de la machine à protéger et enfin celles qui substituent une nouvelle pile à celle d'origine.

Filtrage

- **Stealth patch** [12] : solution simple d'identification de paquets TCP sous la forme d'un patch de module noyau GNU/Linux 2.2-2.4 qui permet de filtrer et d'ignorer les paquets SYN+FIN (sonde QueSO), la sonde T2 de Nmap (bit *«reserved»*) et les paquets FIN+PUSH+URG (sonde T7 de Nmap).

- **Blackhole** [13] : options noyau de filtrage des paquets TCP et UDP qui permettent de bloquer respectivement les réponses RST et ICMP sur des ports fermés.
- **IPlog** [14] : application *«userland»* qui détecte et répond à la place de la pile native à la plupart des sondes Nmap. Les réponses sont forgées statiquement dans le code (iplog_tcp.c) et permettent d'empêcher Nmap d'identifier correctement la cible.
- **OpenBSD packet filter** [15] : pare-feu officiel d'OpenBSD qui permet de spécifier (fichier pf.conf) la plupart des options des paquets lors de leur émission (comme le TTL par défaut, le MSS maximum, la politique de choix de l'ID IP, ...). Les outils qui se basent sur la correspondance de ces champs et leurs signatures ne trouvent plus la signature réelle de l'équipement protégé par cette solution.

Configuration et modification de pile TCP/IP (*«host based»*)

- **Ip Personality** : Module netfilter pour Linux 2.4 qui permet de configurer certains paramètres de la pile TCP/IP.
- **Fingerprint Fucker** : Module noyau pour Linux 2.2, qui sait lire les signatures de l'ancienne base Nmap et configurer les paramètres de la pile en conséquence. Fonctionne uniquement pour les tests T1, T2 et T7.
- **FreeBSD fingerprint scrubber** [1] : Développé pour FreeBSD, il ne cherche pas à émuler le comportement d'un système en particulier mais simplement de ne ressembler à aucun.
- **OSfuscate** [8] : Logiciel pour Windows qui modifie des clefs du registre pour changer certaines spécificités de la pile TCP/IP.

Substitution de pile TCP/IP (*«proxy behaviour»*)

- **Honeyd** [9] : Probablement le logiciel de honeypot le plus connu, Honeyd a la possibilité de simuler les signatures Xprobe2 et Nmap (ancienne version) pour ses hôtes virtuels.
- **Packet purgatory / Morph** [10] : Morph est un processus capable d'émuler 3 personnalités en modifiant les paquets sortants. Pour se faire il s'intercale entre l'interface réseau et le noyau grâce à PacketPurgatory, une bibliothèque basée sur libPcap.

3.2 Synthèse de l'état de l'art

La majorité des outils présentés sont déjà anciens ou ne sont plus maintenus. Pour la plupart ils n'assurent qu'une perturbation partielle des outils de prise d'empreinte.

A notre sens, ce ne sont pas «véritablement» des outils de personnalisation complète d'identité mais plutôt des techniques unitaires ad-hoc de perturbation.

La mise en œuvre de ces outils est souvent laborieuse et nécessite soit la possession de l'équipement filtrant considéré, soit une administration système spécifique en interaction avec le noyau ou le pare-feu de la cible à protéger. C'est exactement ce qu'IpMorph s'impose de ne pas faire. Nous souhaitons une solution applicative totalement en espace utilisateur qui peut facilement se déployer sur une large gamme d'équipements et qui permet une mystification totale suivant l'identité choisie par l'utilisateur.

En dehors de ces constatations générales, la solution qui semble de prime abord la plus proche de nos enjeux est l'outil Morph basé sur la bibliothèque *«userland»* Packet Purgatory [10]. Malgré des noms très similaires, les projets Morph et IpMorph sont radicalement différents en termes d'objectifs et de mise en œuvre.

Morph est principalement un outil *«proof of concept»*, qui ne gère partiellement que quelques outils de prise d'empreinte et ne possède pas de mécanisme de personnalité ou de paramétrage si ce n'est trois comportements différents codés directement dans les sources (OpenBSD 3.3, Linux 2.4, Windows 2000). Le mécanisme de modification des paquets est peu extensible (implémentation sous forme de if, switch/case en langage C) et les valeurs des champs sont statiques. Il nous semble en l'état quasi impossible de le transformer en un «moteur» de personnalités génériques.

De plus, l'absence de mise à jour de Morph depuis 2005 et les problèmes de compilation lors de la génération de l'exécutable nous laissent supposer que le projet est abandonné. Le seul mode véritablement fonctionnel de Morph est le mode «proxy» mais on constate plusieurs problèmes de performance et de cohérence (un scan Nmap force Morph à consommer 100% d'un CPU et prend jusqu'à plusieurs minutes, les connexions SSH mettent 5 à 10 fois plus longtemps à s'établir), ainsi que quelques limitations dans la gestion d'un volume important de connexions (si on lance un scan Xprobe2 en parallèle d'une détection Nmap, Xprobe2 répond que l'hôte n'est pas en ligne).

Enfin, les versions des outils gérés sont dépassées : Xprobe2 reconnaît un FreeBSD 1.5.1 lorsque la personnalité codée dans les sources est «openbsd», et la dernière version de Nmap ne reconnaît jamais rien. Les outils du type SinFP ou RINGv2 ne sont pas supportés.

4 Objectifs et principes de conception

4.1 Contexte du projet

Nos premiers travaux dans le domaine remontent au projet BridNet[18] où notre objectif était de simuler complètement le comportement réseau d'une machine connectée. Nous avons réalisé alors notre première pile TCP/IP en mode utilisateur. Cette expérience riche d'enseignement nous a montré qu'il était tout à fait possible de réaliser une pile minimale complète en espace utilisateur si l'on savait gérer correctement la fragmentation IP et le suivi de session TCP.

Plus récemment et dans le cadre du projet Hynesim[19], nous souhaitons pouvoir «personnaliser» le comportement réseau de certaines machines virtuelles de la plateforme et principalement de celles basées sur OpenVZ. Afin de scinder les problèmes et les composantes techniques, nous avons décidé de reprendre notre pile TCP/IP et de l'adapter à la mystification de prise d'empreinte. L'idée est simple : «comment faire apparaître une machine virtuelle comme un autre système d'exploitation aux yeux des outils de détection». Le projet IpMorph était né.

La section suivante présente dans un premier temps les objectifs de conception et de réalisation que nous nous sommes fixés et dans un second temps, l'architecture générale de l'outil que nous réalisons.

4.2 Objectifs

La genèse du projet IpMorph résulte directement d'un besoin pratique, aussi bien conceptuel que technique. A ce titre, il est destiné à être utilisé dans plusieurs de nos projets actuels et futurs. Il ne s'agit pas pour nous d'une simple idée ou d'un concept théorique élégant mais bien plus de la conception et la réalisation d'une solution opérationnelle, documentée, robuste et maintenue dans le futur. En plus d'être une excellente occasion d'en apprendre plus sur la prise d'empreinte, ce projet nous donne la possibilité de réutiliser plusieurs composants logiciels validés lors de précédents projets et de mener un travail innovant de synthèse des signatures de différents outils.

Notre principal objectif est de concevoir une application en espace utilisateur qui protège une machine en empêchant l'identification de son empreinte TCP/IP, et qui permet de surcroît de personnaliser à loisir son apparence de telle sorte que chaque outil de test la perçoive comme celle configurée dans IpMorph, aussi bien face à des techniques de prise d'empreinte actives que passives. Dès lors, nous parlons d'unification car nous réussissons à mystifier plusieurs outils différents avec une unique solution, IpMorph.

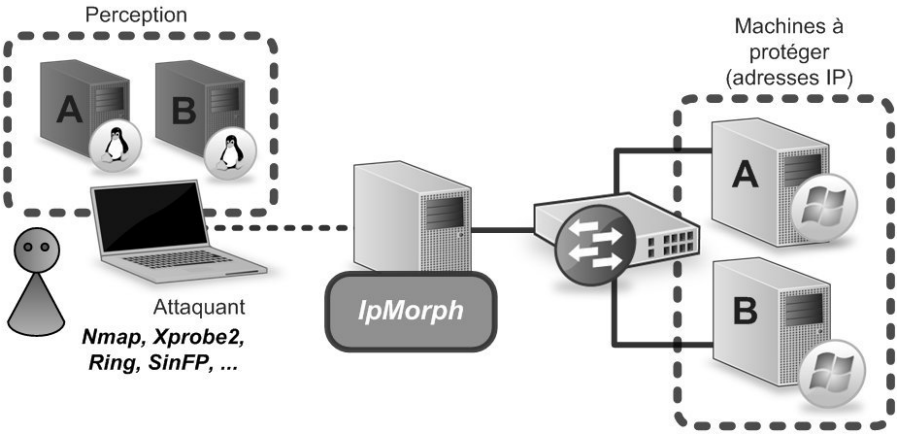

Fig. 3. Utilisation d'IpMorph en coupure afin de masquer des hôtes «réels».

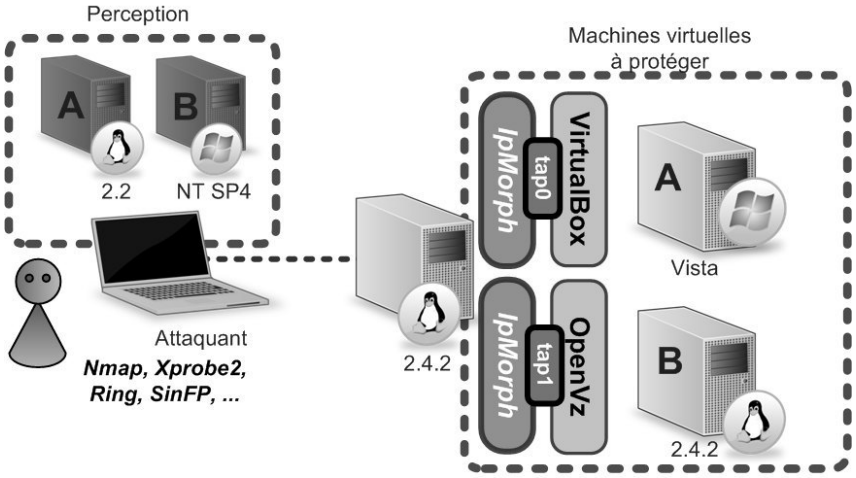

Fig. 4. Utilisation d'IpMorph sur un hôte afin de masquer des machines virtuelles hébergées par un hôte.

L'impératif qu'il convient de garder tout au long de cette conception est que la machine protégée doit continuer à être complètement opérationnelle d'un point de vue réseau. En particulier, il est primordial que les services hébergés par la machine en protection continuent à fonctionner nominalement. IpMorph ne doit en aucun cas introduire une rupture réseau applicative. Ce challenge impose un soin scrupuleux aux techniques mises en œuvre lors de la manipulation des paquets transitant et notamment lors de leur réécriture. Nous souhaitons concevoir et réaliser une application qui permette une grande variété de mises en œuvre et ainsi protéger des machines réelles d'un réseau (Fig.3)ou bien des machines virtuelles hébergées par un hôte qui intègre IpMorph (Fig.4). Au niveau de l'implémentation de cet outil, nous souhaitons réaliser le cœur de l'application en C++ , afin d'allier performance et programmation orientée objet, cela de manière portable et en minimisant les dépendances de bibliothèques, pour assurer sa compatibilité et sa portabilité «source» sur un large éventail de plateformes (principalement à ce jour BSD, Linux, Solaris et MacOS).

4.3 Architecture générale

Le principe de base de notre solution est la mise en place d'une pile TCP/IP en espace utilisateur qui possède deux faces, l'une exposée aux outils de prise d'empreinte («face exposée») et l'autre exposée à la machine que l'on souhaite protéger («face protégée»). Chacune de ces faces est en liaison directe avec une interface réseau du système hôte, sous les mécanismes de filtrage type pare-feu, qu'elle soit réelle (eth0, ...) ou bien virtuelle (tap0,...) et se charge de lire et écrire les trames dont elle a la responsabilité (les trames en provenance ou vers les outils de prise d'empreinte pour la face exposée et les trames en provenance ou vers la machine protégée pour la face protégée). Cette lecture sur chacune des interfaces correspond exactement à un *«sniff»* en mode *«promiscuous»* comme on le trouve dans bon nombre d'outils d'analyse réseau. Le travail principal du cœur de l'application est d'assurer un aiguillage intelligent entre ces deux faces en fonction de l'état des flux et des actions que l'on souhaite mettre en œuvre.

Nous souhaitons nous placer en protection dès la couche liaison, c'est pourquoi nous effectuons de prime abord une modification de l'adresse Ethernet lors des requêtes ARP et RARP (Fig.5). Ainsi, la face exposée révèle une adresse MAC qui n'est pas réellement celle de la machine protégée. Pour le moment ce n'est pas géré, mais il serait judicieux d'assurer une cohérence entre le constructeur désigné par le préfixe de l'adresse MAC exposée, et la personnalité choisie pour IpMorph (par exemple, il paraît assez improbable d'être réellement en face d'une machine sous Windows Vista si le préfixe de l'adresse MAC est celle d'un constructeur de téléphone VoIP).

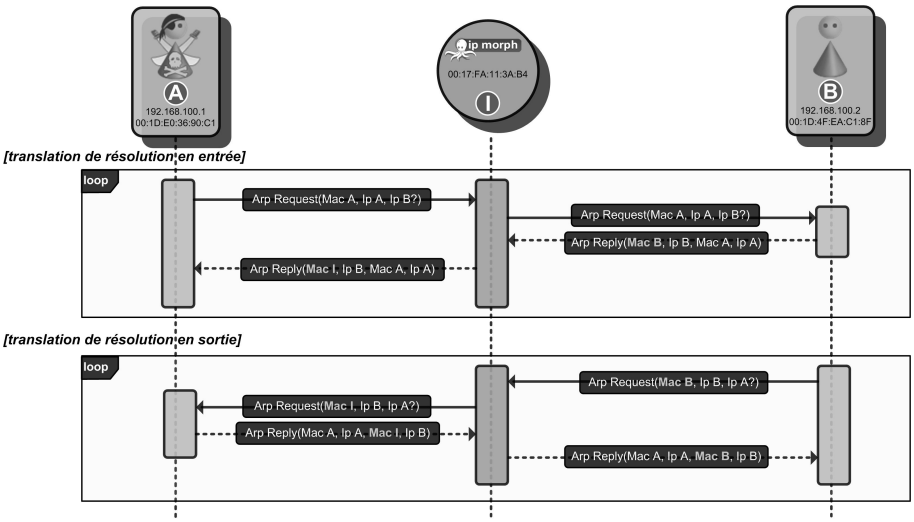

Fig. 5. Traduction bidirectionnelle des résolutions ARP

IpMorph ne s'intéresse qu'au trafic légitime à destination ou en provenance de la machine à protéger et filtre aussi bien au niveau Ethernet qu'IP les flux qui transitent. En d'autres termes, IpMorph ne s'introduit que dans les échanges en relation avec la machine protégée.

Le travail ainsi déporté aux niveaux IP et supérieurs, IpMorph prend en charge, pour chaque face, le réassemblage et la fragmentation IP ainsi que le suivi de session TCP à l'image de ce que font la plupart des pare-feux modernes *«stateful»*. Un paquet de données IP, qu'il soit lu ou écrit sur l'une ou l'autre des faces, passe obligatoirement par le module de réassemblage (pour la lecture) et le module de fragmentation (lors de l'écriture). Ce fonctionnement est stricto sensu celui d'un routeur situé par exemple entre deux brins aux MTU différents. Dès lors, au niveau de la couche réseau, chaque datagramme IP est directement analysable par IpMorph (on a l'assurance qu'il s'agit d'un paquet complet d'une couche supérieure, par exemple ICMP, TCP ou encore UDP). Une fois le paquet (et donc les entêtes) complet, nous remontons les couches jusqu'à s'intéresser à TCP pour assurer un suivi de session et relayer les flux jusqu'à la face opposée (Fig.6). Pour chaque flux, un contexte IpMorph est créé et stocké afin de garder la mémoire des sessions, des états, des décisions et éventuellement des données échangées précédemment.

Dotés de ces contextes, nous effectuons à chaque étape d'introspection des traitements sur les données et des choix de filtrage. Le fonctionnement d'IpMorph est la plupart du temps celui d'un relais et d'un outil de normalisation de paquets. Dans

certains cas et notamment lors de la détection de sondes d'outils de prise d'empreinte actifs, IpMorph ne relaie pas ces trames et forge autoritairement (au sens où c'est IpMorph qui décide de répondre à la place de la machine protégée) chaque réponse attendue par ces outils. De manière simplificatrice on peut considérer que notre solution fonctionne en mode «interception» pour les outils actifs et en mode «relais» pour les outils passifs (ce relais n'est évidemment pas une recopie directe puisque nous modifions les paquets afin qu'ils présentent les caractéristiques attendues).

Le principal défi en suivant ce fonctionnement est d'être capable de garder une cohérence de bout en bout (pour TCP par exemple) entre les flux côté exposé et les flux associés du côté protégé. Le trafic réseau de production doit continuer à s'effectuer sans encombre et chaque état des connexions doit continuer à correspondre des deux côtés à la fois. De manière analogique, cela peut être vu comme une sorte de SNAT/DNAT étendu à l'ensemble des champs IP et TCP en plus des ports source et destination. La plupart du temps nous effectuons des traductions sur les champs des paquets en les gardant en mémoire dans les contextes pour les appliquer à l'inverse au retour.

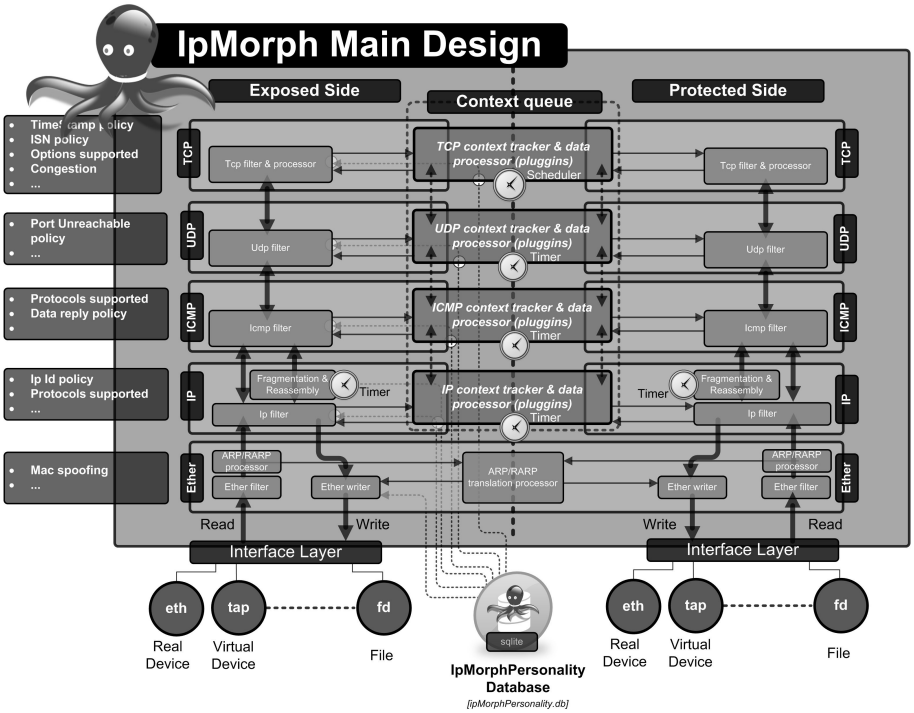

Fig. 6. Architecture fonctionnelle détaillée d'IpMorph

5 Mise en œuvre et focus technique

IpMorph est composé d'un cœur (la pile TCP/IP virtuelle à deux faces) et d'un ensemble d'outils externes d'utilisation (ipmController, ipmPersonalityDBManager, ipmGui).

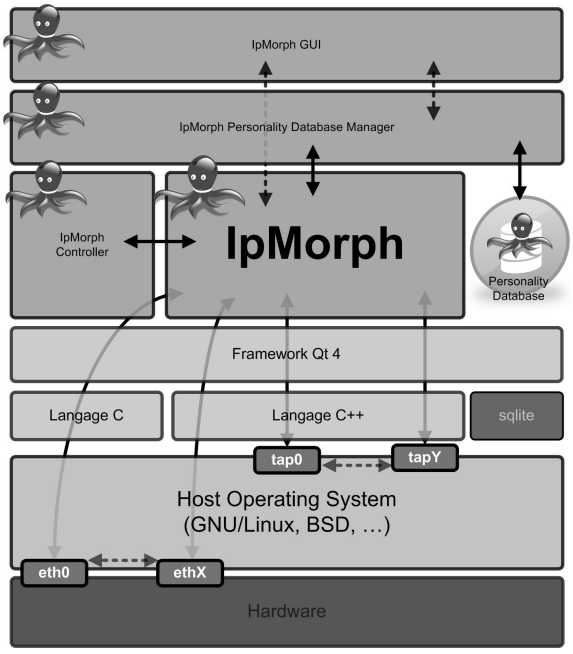

Fig. 7. Dépendances logicielles des composants d'IpMorph

Contrairement à beaucoup d'outils open source (par exemple Morph) qui lisent et envoient des paquets réseaux, IpMorph n'utilise pas le tandem traditionnel libpcap/libdnet. Même s'il n'est pas exclu qu'un jour, afin d'assurer la portabilité, nous utilisions ces bibliothèques dédiées, nous les avons pour le moment volontairement laissées de côté afin de maîtriser au plus près le code et d'unifier notre conception dans les objets manipulés (aussi bien en lecture, en traitement et en envoi). Nous nous reposons exclusivement sur du code système C standard pour les couches basses de notre solution. Aucune dépendance (hormis la bibliothèque C++ Qt4 que nous utilisons comme facilitatrice de développement et de structure d'accueil de nos classes C++) n'est requise dans le projet(Fig.7).

5.1 Intégration de Nmap

Nmap est probablement l'outil de prise d'empreinte actif le plus utilisé et le plus complet en termes de signatures. De ce fait et même si nous considérons que ce n'est pas forcément l'outil le plus pertinent, il est primordial de pouvoir le mystifier grâce à IpMorph.

Lors d'une tentative de prise d'empreinte, Nmap commence par effectuer un balayage des ports ouverts et fermés sur sa cible. Durant ce balayage préalable, IpMorph n'a aucune raison de s'immiscer dans ces échanges avec la machine protégée, il fonctionne donc en mode relais. Une fois ce scan effectué, Nmap envoie une série de 16 paquets qui constituent ses tests discriminants pour tenter d'identifier le système distant. Les 16 sondes sont constituées par l'envoi séquencé (intervalle moyen de 110 ms) d'une série de 6 paquets TCP sur un port préalablement détecté comme ouvert, de deux paquets ICMP *echo request*, d'un paquet TCP de test du support de congestion (ECN), d'une série de 6 paquets TCP vers des ports ouverts et/ou fermées et d'un paquet UDP vers un port fermé. A ce niveau IpMorph fonctionne en mode interception et répond (ou ne répond pas) directement selon sa propre initiative sans jamais relayer les paquets vers le côté protégé. En fonction de la personnalité désirée, IpMorph tente de répondre exactement ce qu'attend Nmap. Le travail d'IpMorph est donc d'interpréter la signature Nmap désirée et de réagir en adéquation face aux sondes.

Ce fonctionnement en interception est possible car chacun des 16 paquets est facilement identifiable et largement documenté (si un doute persiste, l'analyse du code source de Nmap révèle clairement ce qu'il fait). La plupart des réponses (ou la nécessité de ne pas répondre) sont facilement gérables même si ce travail a été laborieux du fait de la variété des éléments de signatures et, par conséquent, des réponses possibles. Pour autant, parmi les tests effectués par Nmap, certains sont plus difficiles à satisfaire que d'autres et pour rester concis, nous avons décidé de présenter ci-dessous uniquement les points particuliers que nous considérons comme non triviaux et qui ont demandé un travail plus important.

Tests *«Sequence generation (SEQ, OPS, WIN and T1)»* : Ce test porte sur l'analyse des numéros de séquences initiaux TCP (ISN) choisis par la pile cible. Cet élément de la signature est très discriminant au sens de Nmap pour l'identification d'un système distant. Le test consiste en l'émission d'une séquence de six paquets TCP sur un port ouvert. Ensuite, Nmap analyse puis calcule trois valeurs depuis les six numéros de séquences initiaux glanés (dans les réponses SYN+ACK à ses paquets SYN). Ces trois valeurs sont les trois attributs de signature GCD, ISR et SP. Dès lors qu'une nouvelle session est détectée sur la face exposée (réception d'un paquet

SYN), un contexte TCP est créé et une nouvelle session est initiée sur la face protégée (envoi d'un paquet SYN). Au retour du paquet SYN+ACK de la machine protégée, un numéro de séquence initial est choisi par IpMorph pour remplacer dans la session côté exposé celui utilisé par la face protégée. Cette différence est mémorisée dans le contexte précédemment crée pour l'appliquer au retour et ainsi assurer le maintien de la connexion côté protégé.

La problématique que nous adressons ici est de correctement générer nos ISN afin que les 6 ISN consécutifs attendus par Nmap lui permettent de calculer les valeurs que nous souhaitons pour les attributs de signature GCD, ISR et SP. Autrement formulé, le problème consiste à trouver comment générer successivement des ISN à partir uniquement de trois informations stockées dans la signature Nmap ?

Ces trois paramètres sont les suivants :

- **TCP ISN greatest common divisor (GCD)** : le plus grand commun diviseur des cinq écarts d'ISN (différences des 6)
- **TCP ISN counter rate (ISR)** : le taux moyen d'incrément par seconde d'ISN c'est-à-dire à quelle vitesse s'incrémente les ISN
- **TCP ISN sequence predictability index (SP)** : la dispersion dans la génération des ISN par rapport à la moyenne, plus précisément l'écart-type des 5 taux d'incréments entre eux par rapport à la moyenne (ISR ci-dessus).

S'il est aisé de générer des valeurs qui respectent une moyenne donnée et un écart-type connu, respecter de surcroît que leurs différences respectives possèdent le même plus grand commun diviseur complique sérieusement la tâche. Cette tâche de génération contrainte par le GCD n'est pas triviale (sans parler du fait que les ISN étant stockés sur 32 bits, ils peuvent boucler) et a représenté un verrou technique sur IpMorph pendant plusieurs semaines. Poussés par le sentiment qu'il était sûrement possible (au moins en utilisant des méthodes issues de la recherche opérationnelle) de trouver une solution acceptables nous avons tenté plusieurs approches.

A ce jour voici la technique que nous utilisons pour résoudre cette problématique :

1. Nous mémorisons systématiquement les 6 derniers ISN générés et leurs dates de génération
2. À chaque génération d'un nouvel ISN nous déterminons, à partir de ce qu'attend Nmap (ISR) et du retard que nous avons éventuellement pris, le meilleur taux d'ISN que nous allons essayer de respecter (autrement formulé : quel est l'ISN idéal que nous devrions renvoyer pour respecter l'ISR que Nmap va calculer sur notre série de réponses ?)

3. En utilisant la formule de Box-Muller, nous générons une série de valeurs qui respecte pour chacune l'écart-type que Nmap attend. Chacune de ces valeurs est arrondie au multiple de GCD près
4. Sur la série pré-calculée à l'étape précédente et pour chacune des valeurs potentielles, nous calculons, comme le ferait Nmap, l'ISR et le SP résultant si nous retenions cette valeur, puis nous appliquons une fonction de coût proportionnelle au carré de la distance par rapport à l'idéal de la signature (de manière à pénaliser les valeurs d'ISN qui s'écartent beaucoup de l'idéal)
5. Au final nous gardons la valeur d'ISN qui minimise la distance, nous la stockons en mémoire ainsi que la date courante de génération en microsecondes. C'est cette valeur d'ISN qui est utilisée dans la réponse SYN+ACK à Nmap.

Test *«TCP RST data checksum (RD)» :* Le test RD nous a posé un problème intéressant que nous présentons ici.

Certaines piles ajoutent un message ASCII dans les données d'un paquet TCP RST (en général le message d'erreur). Ce cas est relativement rare et en effet, peu de signatures Nmap possèdent ce champ RD, mais afin d'assurer la complétude de notre approche, nous devions prendre en charge ce test pour mystifier totalement Nmap.

Afin d'éviter d'avoir à stocker le message présent dans le paquet TCP RST, Nmap stocke un CRC-32 de ces données dans ses signatures. Hélas cela signifie qu'IpMorph n'a plus accès au message d'origine, à renvoyer lors de l'émission d'un paquet TCP RST. Nous devons donc générer des données correspondantes au même CRC-32 qui sera calculé par Nmap en retour. Fort heureusement, l'algorithme CRC-32 est «cryptographiquement» faible, et il possède la propriété que chaque bloc de 4 octets ajoutés à un message peut changer le CRC-32 en n'importe quel CRC-32 voulu. Ce qui induit directement que tous les CRC-32 ont un message de 4 octets correspondant d'origine.

A partir de ce constat, la première approche fut évidemment de tenter le *brute force*, la plage des valeurs à tester étant très limitée (2^{32} combinaisons) au vu des capacités des processeurs actuels. Au delà du fait qu'il faut tout de même une trentaine de secondes en moyenne pour trouver une collision, (ce qui est un trop long, même si on mémorise le résultat dans la personnalité IpMorph) cette approche n'est pas très élégante scientifiquement parlant.

Plus subtilement, nous avons tenté de générer des collisions CRC-32 en O(1), c'est-à-dire d'inverser l'algorithme du CRC-32.

Le calcul d'un CRC-32 s'apparente à la division du message par un polynôme, en utilisant «l'arithmétique polynomiale modulo 2», ce qui correspond en fait à de

l'arithmétique binaire sans retenue (c'est à dire 1+1=0, sans reporter le 1 sur le bit suivant). Dans cette arithmétique, l'addition et la soustraction sont identiques, et équivalent à un XOR. Une version plus efficace de l'algorithme (*Table driven CRC*) utilise une table pré-calculée pour effectuer cette division par bloc de 8 bits (voir [17]) plutôt que bit à bit. Le CRC est ainsi calculé en décalant le message octet par octet à travers un «registre» de la taille du CRC (dans notre cas 32 bits). La valeur sortante résultante du décalage est mémorisée (c'est le *topbyte*). Le registre est combiné par un XOR à la valeur pré-calculée extraite de la table (à la position égale au *topbyte*). On effectue ces manipulations une fois par octet du message, et au final le registre contient notre CRC-32.

Inverser cet algorithme équivaut donc à inverser ce calcul jusqu'à trouver le message de 4 octets produisant le CRC-32 extrait de la signature Nmap. Nous nous sommes directement inspirés de travaux existants [16].

Le principe de l'algorithme est le suivant :

1. Prendre un tableau de 8 octets, remplir les positions 0 à 3 avec le CRC-32 du message jusqu'à maintenant. Remplir les positions 4 à 7 avec la valeur désirée du CRC-32.
2. Prendre la valeur depuis la position 7 et l'utiliser pour retrouver dans la table la valeur complète.
3. Faire un XOR de cette valeur (4 octets) avec les octets 4 à 7.
4. Faire un XOR de la position dans la table avec l'octet 3.
5. Répéter les étapes 2 à 4 trois fois, en décrémentant les indices de 1 à chaque fois (donc valeur 6,5,4 pour l'étape 2 ; 3 à 6, 2 à 5, 1 à 4 pour l'étape 3 et 3,2,1,0 pour l'étape 4)

Au final, les données générant le CRC-32 désiré se trouvent dans le tableau en position 0 à 3. Ainsi, nous pouvons systématiquement retrouver 4 octets dont le CRC-32 est celui des signatures Nmap, en O(1).

Ce *reverse* CRC-32, c'est-à-dire la génération de 4 octets qui corresondent au RD de la signature Nmap est fait lors de la génération de la personnalité. Fort heureusement pour nous, Nmap n'utilise pas une fonction de hachage telle que MD5 ou SHA1, auquel cas nous serions contraints de disposer des véritables systèmes d'exploitation afin d'analyser leurs paquets TCP RST spécifiques, d'en capturer le véritable champ texte et de le stocker en base dans la personnalité d'IpMorph.

Test *«TCP IP ID sequence generation algorithm (TI) and ICMP IP ID sequence generation algorithm (II)»* : Lors de ses tests, Nmap tente de

déterminer l'algorithme de génération de l'ID de paquet IP, à la fois lorsqu'il s'agit d'un paquet TCP et lorsqu'il s'agit d'un paquet ICMP. Comme pour les autres paramètres de signature, IpMorph doit respecter cette politique de génération d'ID de paquet IP. En particulier, nous devons respecter le fait que cette génération d'ID peut être partagée entre TCP et ICMP ou bien distincte (paramètre *Shared IP ID sequence Boolean* (SS)). Pour ce faire IpMorph tient à jour deux attributs `_lastIpIdTcp` et `_lastIpIdIcmp`, qu'il incrémente indépendamment ou ensemble à chaque nouvelle génération d'ID en fonction de la politique d'incrémentation désirée par Nmap. Cette politique d'incrémentation fait aussi lieu d'un test (tests TI et II), par exemple systématiquement égal à zéro (TI=Z), en permanence égal à une valeur spécifique stockée en hexadécimal dans la signature (TI=A400), incrémenté à chaque fois (TI=I), ...

La prise en compte de ce test ne fut pas spécifiquement difficile car, au niveau d'IpMorph, l'appel de notre méthode `_generateIpIdent()` doit simplement respecter scrupuleusement l'éventail des cas possibles et le partage éventuel de cette génération entre TCP et ICMP.

L'implémentation de cette mystification de génération d'ID IP nous a permis de découvrir un dysfonctionnement de Nmap dû à un «bug» dans son code et cela depuis plusieurs versions. Lors de nos tests (qui consistent principalement à lancer IpMorph avec une signature Nmap aléatoire et vérifier pour chaque exécution de Nmap si la signature détectée correspond, ceci en boucle sur l'intégralité des signatures Nmap), nous avons constaté que pour certaines signatures, Nmap ne réussissait pas à identifier l'empreinte désirée. Suivant notre classification de ces signatures problématiques et l'analyse du code source de Nmap (fichier `osscan2.cc`) nous avons découvert que la fonction d'analyse de Nmap de la série d'ID IP renvoyée était incorrecte par rapport à ses objectifs. Plus précisément, fort probablement du fait d'une erreur de copier/coller dans son code, le tableau d'ID mesurés par Nmap était modifié avant sa fonction d'analyse. Ce «bug» constaté a pour nous une importance non négligeable car cela signifie que, depuis plusieurs versions de Nmap, plusieurs signatures de la base Nmap ne seront jamais détectées par l'outil même si l'OS distant est bien celui de la signature présente en base (autrement dit, la signature en base est correcte, mais du fait de son analyse erronée Nmap ne la détectera jamais). Nous avons corrigé ce dysfonctionnement, et avons naturellement remonté un patch auprès des développeurs.

Cette analyse du code de Nmap sur la dernière version (à ce jour en version 4.85 BETA 4) nous a permis de comprendre que certains tests non documentés sont déjà en préparation. A titre d'exemple et d'ici peu, Nmap effectuera son test de génération d'ID IP à la fois sur les ports TCP ouverts, sur les ports TCP fermés et sur ICMP. Un nouveau champ CI (correspondant à la politique de génération détectée

lorsque le port TCP est fermé) devrait faire son apparition dans les futures bases de signatures, ce qui portera à trois le nombre de paramètres pour ce test d'IP ID. Pour nous il s'agit simplement de rajouter un attribut spécifique `_lastIpIdTcpClosed` afin de différencier si besoin cette génération dans la méthode `_generateIpIdent()` d'IpMorph.

5.2 Intégration de SinFP

SinFP[6], développé en Bretagne par Patrice Auffret, est le premier outil de prise d'empreinte à unifier la prise d'empreinte active à celle passive et à effectuer de la reconnaissance de pile sur IPv6. Par sa conception et son fonctionnement (envoi au maximum de trois paquets TCP en mode actif) c'est un outil intéressant et radicalement différent de Nmap en termes du volume de stimuli et de caractérisation des sondes (envoi de paquets conformes). Du fait de ses caractéristiques et principalement de son mode d'analyse passif, il était primordial pour nous de pouvoir l'intégrer et ainsi de le mystifier. Contrairement aux sondes Nmap qui sont aisément détectables (où IpMorph fonctionne alors en interception), dans ce cas présent, notre outil fonctionne uniquement en relais entre la face exposée et la face protégée. Notre approche a été la suivante : tout d'abord mystifier SinFP lors d'une détection d'empreinte active, puis tenter de le mystifier lors d'une prise d'empreinte passive.

Mystification de SinFP en mode actif : A partir de la base de signature de SinFP (au format SQLite), nous extrayons la signature désirée. Nous n'interprétons d'ailleurs que sa première heuristique et nous ne tenons pas compte de ses masques de déformation car notre intention est de renvoyer exactement ce à quoi s'attend SinFP pour s'assurer d'une mystification parfaite dès la première analyse des résultats par SinFP. Les trois sondes de SinFP sont des paquets standards TCP sur un port spécifié comme ouvert (SYN, SYN avec options et SYN+ACK) qui provoque les réponses respectives de la part de la machine protégée (SYN+ACK, SYN+ACK et RST). Comme il n'y a pas lieu de différencier une telle sonde SinFP d'une connexion légitime, IpMorph se place systématiquement en relais sur les retours des paquets de la face protégée tout en modifiant ceux-ci lors de l'envoi sur la face exposée (vers SinFP). Ce relais/réécriture au retour concerne uniquement les paquets SYN+ACK et RST concernés et consiste à modifier les champs des entêtes IP et TCP en fonction de la signature SinFP attendue. Afin d'illustrer concrètement ce relais, nous donnons ci-dessous un exemple de réécriture sur le test P2 de SinFP :

– Lorsqu'un paquet SYN arrive sur la face exposée, un contexte est créé et le paquet est transmis à la face protégée.

– Lors de la réception du paquet SYN+ACK de réponse de la machine protégée, nous consultons la personnalité IpMorph. Pour cet exemple, nous considérons que cette dernière a été construite depuis la signature 140 de SinFP, c'est à dire un OpenBSD 3.7.
– En fonction des paramètres de cette personnalité, nous effectuons un certain nombre de transformations :
– Nous appliquons directement les valeurs de TTL initial, de bit Don't Fragment (DF), de maximum Segment Size (MSS), d'option TCP et de Windows Scale au paquet, tel que nous les avons lues de la signature 140.
– Nous appliquons ensuite les bons algorithmes pour générer les valeurs de SEQ, ACK, et IP ID (encore une fois, en se basant sur la signature SinFP).
– Le paquet ainsi modifié est transmis à la face exposée.

Mystification de SinFP en mode passif : Contrairement à ce que nous pouvions penser au départ, la prise en charge de la mystification de SinFP en mode passif est quasi immédiate si l'on sait aussi modifier les paquets SYN sortants, c'est à dire issus de la machine protégée. Ce constat vient principalement du fait que SinFP utilise une approche unifiée aussi bien en actif qu'en passif (à quelques détails près) sur l'analyse des paquets acquis.

En passif SinFP peut détecter l'identité de la machine en communication à la fois sur ses réponses aux SYN (c'est-à-dire les paquets SYN+ACK) d'un client et à la fois sur les connexions initiales de la machine vers un serveur (c'est-à-dire les paquets SYN). Pour nous, le premier cas est automatiquement géré car nous nous s'assurons déjà de bien répondre en SYN+ACK comme s'y attend SinFP. Ce n'est rien de moins que le même cas lors de la mystification en mode actif.

A titre illustratif, en mode passif et pour un paramétrage d'IpMorph avec la signature 140 (OpenBSD 3.7) de la base SinFP, voilà ci-dessous la détection qui est faite :

```
$ sudo /usr/local/sinfp/bin/sinfp.pl -P -d eth0
192.168.100.110:80 > 192.168.100.73:47979 [SYN|ACK]
P2: B11111 F0x12 W16384  O0204ffff01010402010303000101080affffffffffffffff
    M1460
IPv4: HEURISTIC0/P2: BSD: OpenBSD: 3.5
IPv4: HEURISTIC0/P2: BSD: OpenBSD: 3.6
IPv4: HEURISTIC0/P2: BSD: OpenBSD: 3.7
IPv4: HEURISTIC0/P2: BSD: OpenBSD: 3.8
IPv4: HEURISTIC0/P2: BSD: OpenBSD: 3.9
IPv4: HEURISTIC0/P2: BSD: OpenBSD: 4.0
```

La machine 192.168.100.110 (qui est une Ubuntu 8.04) protégée par IpMorph apparaît bien sur les SYN/ACK comme une machine de la famille OpenBSD 3.5 à

4.0 (cette plage d'incertitude est due au fait que ces signatures SinFP sont identiques pour le test P2 et que c'est la seule identification à laquelle a accès l'outil dans ce mode passif).

Dans le deuxième cas et lorsque SinFP n'a accès qu'aux paquets SYN issus de la machine protégée, nous devons naturellement rajouter à notre fonction de relais/réécriture la prise en charge de ces paquets de demande de connexion vers le monde extérieur. Dans ce mode de détection SinFP analyse le paquet SYN en le normalisant à sa manière pour le faire correspondre à une analyse de test P2 (plusieurs champs du paquet reçu et de la signature en base sont désactivés puisque le paquet initial de comparaison P1 ou P2 n'a pas été envoyé par SinFP). Notre tâche consiste dès lors à modifier uniquement le paquet SYN sur les champs qui ont un sens, c'est-à-dire l'ordre et les valeurs des options TCP donnés par la signature SinFP, le MSS, et la taille de la fenêtre. Le numéro de séquence initial est choisi et modifié comme à l'habitude dans IpMorph (cf Nmap), le numéro d'acquittement est fixé à zéro et le TTL celui de la personnalité d'IpMorph (issu de la signature d'autres outils).

Dans ce cas précis, SinFP ne peut pas légitimement se baser sur les champs de l'entête IP (ID, TTL, bit DF) ni sur les numéros de séquence et d'acquittement du paquet puisqu'il n'a pas de signature correspondant en base (les motifs acquis sont souvent des différences par rapport à une requête et uniquement sur des réponses SYN+ACK). Conscient de la chose et lors de l'analyse de ce paquet en mode passif, l'auteur ne tient pas compte de la plupart des champs pour sa correspondance dans la base de signature.

Malheureusement pour SinFP, certains champs d'analyse et de *pattern matching* sont conservés alors que nous pensons qu'ils ne devraient pas l'être car cela rend l'identification beaucoup moins précise du premier coup (sans activer les masques de déformation avancés). Pour s'en convaincre voici ci-dessous la détection qui est faite en mode passif et pour le même paramétrage d'IpMorph avec la signature 140 (OpenBSD 3.7) quand SinFP n'accède qu'aux paquets SYN :

```
$ sudo /usr/local/sinfp/bin/sinfp.pl -P -d eth0
192.168.100.110:35366 > 192.168.100.73:80 [SYN]
P2: B11110 F0x12 W16384 O0204ffff01010402010303000101080affffffff00000000 M1460
IPv4: unknown
```

Si nous activons maintenant les masques de déformation avancés (ce qui revient à autoriser n'importe quelle valeur pour l'analyse B11110 et donc omettre ces critères d'analyses), la nouvelle détection est :

```
$ sudo /usr/local/sinfp/bin/sinfp.pl -P -d eth0 -H
192.168.100.110:59893 > 192.168.100.73:80 [SYN]
```

```
P2: B11110 F0x12 W16384 O0204ffff01010402010303000101080affffffff00000000 M1460
IPv4: BH2FH0WH00H0MH1/P2: BSD: OpenBSD: 3.5
IPv4: BH2FH0WH00H0MH1/P2: BSD: OpenBSD: 3.6
IPv4: BH2FH0WH00H0MH1/P2: BSD: OpenBSD: 3.7
IPv4: BH2FH0WH00H0MH1/P2: BSD: OpenBSD: 3.8
IPv4: BH2FH0WH00H0MH1/P2: BSD: OpenBSD: 3.9
IPv4: BH2FH0WH00H0MH1/P2: BSD: OpenBSD: 4.0
```

Soucieux de contribuer à la qualité d'outils majeurs comme Nmap ou SinFP, nous avons fait part à l'auteur de notre analyse. Pour nous, il n'y a pas lieu d'effectuer des tests de comparaison lorsque l'échantillon de référence est trop faible et qu'aucune comparaison avec des valeurs de référence n'est possible.

5.3 Intégration de p0f

p0f a été conçu et réalisé par Michal Zalewski et c'est historiquement le premier outil opérationnel et efficace de prise d'empreinte en mode passif (d'autres outils préexistants tel que Siphon n'ont pas dépassé le stade de preuve de concept). Outil incontournable de la détection et de la prise d'empreinte en mode passif, il se devait d'être intégrer au sein de notre mystificateur.

Par défaut, p0f s'intéresse à une analyse des paquets SYN, c'est-à-dire pour nous les paquets SYN «sortants» (issus de la machine en protection). Non seulement pour nous il s'agit exactement du même branchement (relais et réécriture) que pour le mode passif de SinFP sur les paquets SYN, mais de plus, la réécriture de ces paquets est la même puisque les signatures de p0f et donc des champs analysés sont quasiment les mêmes que ceux de SinFP (ou réciproquement).

Aucune spécificité de p0f n'a due être prise en compte puisque nous avions déjà traité l'intégration de SinFP. Autrement formulé, le mode opératoire et les critères d'analyses de SinFP «recouvrent » intégralement ceux de p0f.

Pour nous en convaincre et sans avoir modifié IpMorph depuis l'intégration complète de SinFP, nous avons effectué quelques tests en configurant IpMorph avec des signatures issues de SinFP qui correspondent à des piles identifiées dans la base de p0f. Voici ci-dessous un exemple pour la même personnalité que précédemment (signature SinFP N ° 140 : `OpenBSD 3.7`) :

```
\scriptsize \$ sudo ./p0f -i eth0
p0f - passive os fingerprinting utility, version 2.0.8
(C) M. Zalewski <lcamtuf@dione.cc>, W. Stearns <wstearns@pobox.com>
p0f: listening (SYN) on 'eth0', 262 sigs (14 generic, cksum 0F1F5CA2), rule: '
    all'.
192.168.100.110:35365 - OpenBSD 3.0-3.9 (up: 0 hrs)
-> 192.168.100.73:80 (distance 0, link: ethernet/modem)
```

La machine 192.168.100.110 protégée par IpMorph apparaît bien comme un système d'exploitation OpenBSD 3.0 – 3.9.

5.4 Intégration de Ring2

Ring2[11], développé en Bretagne par Franck VEYSSET, Olivier COURTAY et Olivier HEEN, tente d'effectuer une prise d'empreinte distante en s'intéressant aux spécificités temporelles des piles TCP/IP lors d'une simulation provoquée de la congestion TCP. Les solutions de type «relais SYN» ne traitent que très rarement les transmissions à l'initiative des machines protégées [11] (lors de l'établissement de la session et donc de l'envoi par la machine protégée d'un paquet SYN et de l'attente du SYN+ACK). Cette caractéristique est encore plus flagrante s'il s'agit de la déconnexion (fermeture de session par un FIN puis attente du ACK en réponse) à l'initiative de la machine protégée. Puisque ce sont deux des principaux tests de RINGv2 (mesures temporelles sur une simulation de congestion) et que nous considérons cette approche comme novatrice et élégante, nous avons décidé d'intégrer ces types de relais au sein d'IpMorph.

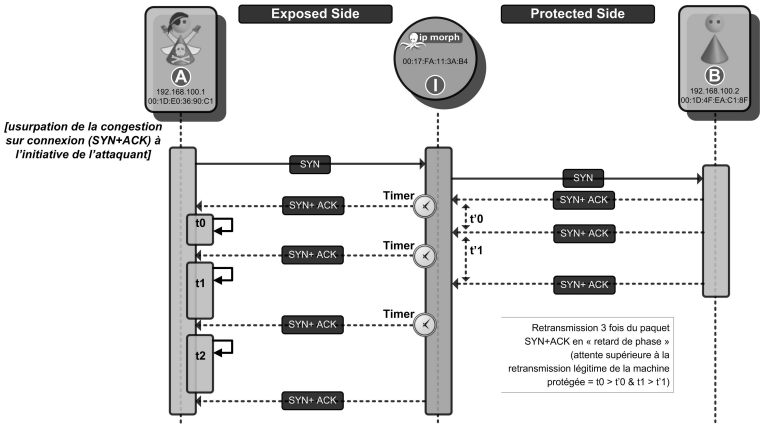

Fig. 8. Mise en œuvre de la mystification de congestion sur une connexion à l'initiative de la machine «côté exposé» (au sens «client»).

La prise en compte de ces tests nous impose d'étendre notre suivi de sessions aux phases où la retransmission TCP doit avoir lieu. En particulier, il peut être nécessaire de répondre «en avance de phase» en fonction de la personnalité définie, et de réémettre à l'initiative d'IpMorph le paquet SYN ou le paquet FIN avant la machine protégée elle-même. Le même phénomène est à prendre en compte pour le cas inverse (introduction d'un retard de retransmission). En outre, et dans ces cas, il faut bloquer les retransmissions légitimes de la machine protégée et gérer correctement

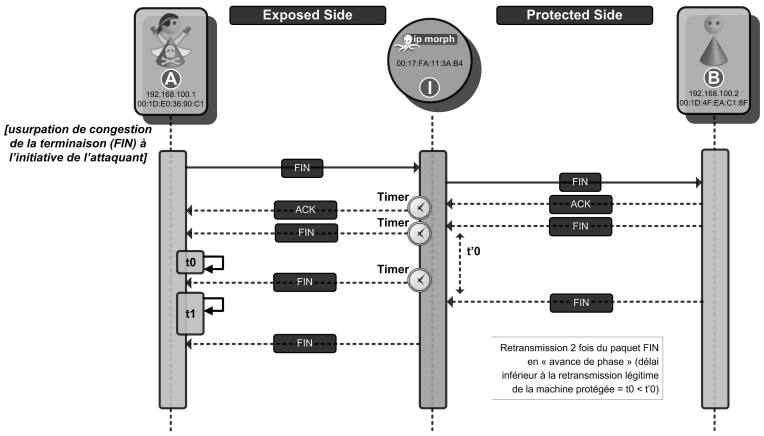

Fig. 9. Mise en œuvre de la mystification de congestion sur une terminaison de session à l'initiative de la machine «côté exposé» (au sens «client»).

cette congestion côté protégé en fonction des résultats du côté exposé. Les schémas Fig. 8 et Fig. 9 illustrent graphiquement ces cas d'utilisations.

Au niveau de l'implémentation nous avons ajouté une horloge (des *timers*) au niveau du contexte IpMorph de suivi de session TCP. Dans chaque cas concerné et donc aussi bien au niveau de l'émission d'un paquet SYN qu'au niveau des FIN nous déclenchons systématiquement un compteur temporel qui déclenche à son tour la réémission du dernier paquet stocké dans le contexte IpMorph (mémoire du trafic) et ceci à intervalles définis et autant de fois que nécessaire en fonction des signatures de Ring2. Comme déjà indiqué plus haut, un soin particulier doit être apporté au filtrage des flux légitimes de réémission de la machine protégée tant que la simulation de retransmission n'est pas terminée sur la face exposée vers Ring2.

5.5 Unification de signature

Le comportement d'IpMorph est défini en interne par un ensemble de paramètres qui concerne aussi bien les options de certaines réponses que le choix algorithmique (branchement) à effectuer sur certains stimuli. L'ensemble de ces paramètres est personnalisable à souhait et c'est ce que nous définissons comme la personnalité IpMorph.

Ce concept de personnalité n'est finalement que le paramétrage multi-niveau et granulaire de l'application en fonction des signatures des outils. Pour nous, une personnalité IpMorph est l'union (ou l'agrégation) de multiples signatures issues des différents outils et de paramètres propres à l'application elle-même.

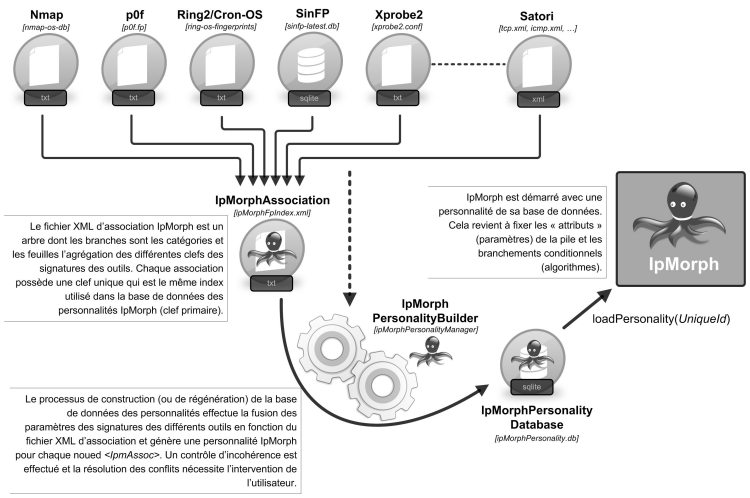

Fig. 10. Principe d'indexation/association, de génération de la base des personnalités et de leurs utilisations.

Au sein d'un format XML d'indexation (Fig.10), nous décrivons textuellement (à l'aide d'un simple éditeur de texte ou grâce à l'emploi de l'IHM IpMorphGui dotée de son module PersonalityDBManager) les associations structurées en catégories et sous-catégories des différentes signatures des outils considérés.

A partir de ce schéma d'indexation et de l'analyse et de l'extraction (class IpMorphFpParser), nous construisons (ou reconstruisons si telle est la demande de l'utilisateur) une base de données SQLite des personnalités IpMorph (au format proche des structures/classes C++ que nous utilisons par la suite dans la pile virtuelle).

Ce mécanisme d'extraction et de construction des personnalités est capable de détecter certaines incohérences entre les signatures, qu'elles soient du fait d'une incohérence de signatures entre outils ou d'une mauvaise indexation dans le fichier XML d'association. Nous pensons par exemple à des attributs «uniques» de la pile IpMorph (comme par exemple icmp.ip.defaultTtl ou icmp.ip.idPolicy) qui peuvent être renseignés ou décrits avec des valeurs différentes dans les différentes signatures (Fig.11). En fonction de la portée de ces incohérences, l'outil de génération de personnalité peut nécessiter l'interaction (résolution de conflit) de l'utilisateur.

6 Conclusion et perspectives

Nous avons montré dans cet article qu'il était tout à fait possible de concevoir et de réaliser un *fingerprint scrubber* qui rende inopérants la plupart des outils de

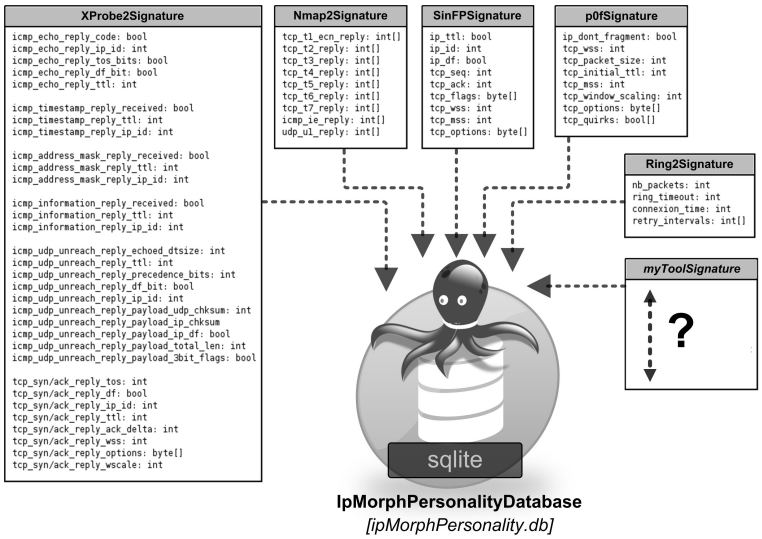

Fig. 11. Exemple détaillé des structures de lecture et de stockage des paramètres des outils d'extraction de signature (classe abstraite IpMorphFpParser dérivée en fonction des outils).

prise d'empreinte, qu'ils soient en mode actif ou en mode passif. Le travail le plus significatif a été d'analyser en détail les modus operandi de chaque outil et nous a permis d'en corriger certains dysfonctionnements (cas de Nmap) ou de suggérer des modifications afin de rendre ces outils plus pertinents (cas de SinFP). La nécessaire prise en compte des multiples signatures des différents outils nous a permis d'effectuer un travail de synthèse concernant les différents paramètres des signatures et nous a menés naturellement au concept de personnalité unifiée. Enfin, la mise en œuvre d'un succédané de pile TCP/IP au sein de notre outil et l'étendue des paramètres et branchements qui ne sont pas à ce jour utilisés dans les signatures des outils de prise d'empreinte nous poussent à croire que de nombreuses nouvelles techniques d'OSFP peuvent voir le jour. IpMorph s'arrête pour le moment à TCP et donc là où, à notre sens, l'avenir de la détection de personnalité commence. Il nous apparaît primordial un jour d'être capable de remonter au moins la plupart des couches applicatives réseau de base afin de mystifier aussi le contenu «réel» du trafic (DNS, DHCP, SMB, ...). Cette perspective semble d'autant plus pertinente que certains outils comme Satori mènent déjà ce type d'analyse sur les flux pour identifier l'empreinte des machines sondées.

Néanmoins, et même si nos principes de conception doivent rendre IpMorph plus facile à évoluer (afin de gérer de nouvelles versions des outils existants ou en devenir ainsi que de nouvelles méthodes de prise d'empreinte), ce type d'approche est fortement dépendante des évolutions des outils que nous souhaitons mystifier et reste bien souvent un travail ad-hoc difficilement généralisable. En particulier il nous apparaît évident qu'il est relativement aisé de détecter la présence d'un outil tel qu'IpMorph. Cette dernière constatation est à replacer dans son contexte et à modérer car au final la personne saura uniquement qu'elle ne doit accorder aucune confiance à sa prise d'empreinte effectuée et qu'un mécanisme filtrant l'empêche d'arriver à ses fins.

Enfin, et au-delà de ce qu'est uniquement IpMorph pour le moment (pour nous un simple prototype prometteur issu d'une étude de faisabilité), il est primordial de mener une phase d'expérimentation et de mesure de cet outil en production ainsi que comme pour la plupart des prototypes, une nouvelle phase d'implémentation afin de rendre IpMorph robuste, documenté, distribuable et pérenne.

Pour autant, le cadre de cette application (usurper la véritable identité des ressources protégées), les enjeux potentiels, les perspectives d'évolution et les premiers retours des personnes à qui nous avons présenté nos concepts nous poussent à croire que ce projet peut rapidement mener à un apport significatif dans le monde de la sécurité des systèmes d'information.

Références

1. Smart, M., Malan, G.R., Jahanian, F. : Defeating TCP/IP Stack Fingerprinting, 9th USENIX Security Symp.
`http://www.usenix.org/events/sec00/full_papers/smart/smart_html/index.html`
2. Smith, C., Grundl, P. : Know Your Enemy : Passive Fingerprinting (2002)
`http://old.honeynet.org/papers/finger/`
3. Fyodor : Remote OS detection via TCP/IP Stack FingerPrinting
`http://www.insecure.org/nmap/nmap-fingerprinting-article.txt`
4. Spangler, R. : Analysis of Remote Active Operating System Fingerprinting Tools, Ettercap, Nmap and other OS detection tools (2008)
`http://www.packetwatch.net/documents/papers/osdetection.pdf`
5. Veysset, F., Courtay, O., Heen, O. : New Tool And Technique For Remote Operating System Fingerprinting (2002)
`http://www.hackerz.ir/e-books/remote_os_detection.pdf`
6. Auffret, P. : SinFP, Unification de la prise d'empreinte passive et active des systèmes d'exploitation, SSTIC 2008
`http://www.gomor.org/bin/view/GomorOrg/ConfSstic2008`
7. Berrueta, D.B. : A practical approach for defeating Nmap OS-Fingerprinting (2003)
`http://nmap.org/misc/defeat-nmap-osdetect.htm`

8. Crenshaw, A. : OSfuscate : Change your Windows OS TCP/IP Fingerprint to confuse P0f, NetworkMiner, Ettercap, Nmap and other OS detection tools (2008)
http://www.irongeek.com/i.php?page=security/osfuscate-change-your-windows-os-tcp-ip-fingerprint-to-confuse-p0f-networkminer-ettercap-nmap-and-other-os-detection-tools
9. Provos, N. : Honeyd : A Virtual Honeypot Daemon (2003)
http://www.citi.umich.edu/u/provos/papers/honeyd-eabstract.pdf
10. Wang, K. : Frustrating OS Fingerprinting with Morph (2004)
http://www.synacklabs.net/projects/morph/Wang-Morph-TheFifthHOPE.pdf
11. Veysset, F., Courtay, O., Heen, O. : Détection des systèmes d'exploitation avec RINGv2 Actes SSTIC 2003
12. Trifero, S., Callaway, D. : Linux Stealth patch (2002)
http://www.innu.org/~sean/
13. Rehmet, G. : FreeBSD Blackhole
http://www.gsp.com/cgi-bin/man.cgi?section=4&topic=blackhole
14. McCabe, R. : IPlog (2001)
http://ojnk.sourceforge.net/stuff/iplog.readme
15. Hartmeier, D. : OpenBSD Packet Filter
http://www.openbsd.org/faq/pf/index.html
16. CRC and how to Reverse it
http://www.codebreakers-journal.com/downloads/cbj/2004/CBJ_1_1_2004_Anarchriz_CRC_and_how_to_Reverse_it.pdf
17. A painless guide to CRC error detection
http://www.repairfaq.org/filipg/LINK/F_crc_v3.html
18. BridNet SSTIC 2005
http://www.bridnet.fr/files/23/sstic2005_bridnet.pdf
19. Hynesim
http://www.hynesim.org