SinFP, unification de la prise d'empreinte active et passive des systèmes d'exploitation

Patrice Auffret

Thomson Corporate Research
Security Labs
1, av. Belle-Fontaine - CS 17616
35576 Cesson-Sévigné - France
patrice.auffret@thomson.net

Résumé Depuis l'omniprésence des pare-feux utilisant de la translation d'adresses et de ports (*NAT/PAT*), l'inspection d'états, ou encore la normalisation de trames, les approches actuelles à la prise d'empreinte des systèmes d'exploitation montrent leurs faiblesses. C'est à partir de ce constat que l'outil *SinFP* fut développé, tentant ainsi de contourner les limitations des outils actuels. *SinFP* implémente de toutes nouvelles méthodes, comme l'utilisation de signatures acquises activement lors de prises d'empreinte passives. De plus, *SinFP* est le premier outil à faire de la prise d'empreinte sur *IPv6* (en mode actif et passif). Grâce à son algorithme de recherche de correspondance, il devient presque inutile d'ajouter de nouvelles signatures dans la base. En effet, son algorithme heuristique le rend très robuste face à des signatures qui ont été modifiées par des dispositifs de filtrage et/ou routage en coupure, ou bien par une personnalisation de la pile *TCP/IP*.

Ce document présente en profondeur les concepts et techniques implémentés dans l'outil *SinFP*.

1 Introduction

La première version de *SinFP* [2] a été diffusée en Juin 2005 [3]. Le programme a beaucoup évolué depuis, et à atteint une bonne maturité. Pourquoi alors publier un article trois ans après ? Tout simplement parce que d'autres publications ont depuis été faites sur la prise d'empreinte des systèmes d'exploitation. Certaines de ces publications [4][5][6] citent *SinFP*, en introduisant des inexactitudes sur son fonctionnement, ou pire, en réinventant *SinFP* [7]. De même[1], *SinFP* n'est pas un outil de prise d'empreinte active ou un outil de prise d'empreinte passive, mais bien les deux.

Dans cet article, nous ne reviendrons pas sur les concepts de la prise d'empreinte active et passive, considérant ceux-ci acquis par le lecteur. L'article suivant [1] introduit ces concepts. Le principe général et les fonctions supportées par l'outil seront vus dans la section 2, puis le fonctionnement interne de la

[1] On peut trouver sur divers blogs et/ou forums des inexactitudes concernant les possibilités de *SinFP*.

prise d'empreinte active dans la section 3 et la version passive dans la section 4. L'algorithme de recherche de correspondance sera traîté dans la section 5 et les masques de déformation (le plus important concept introduit dans *SinFP*) dans la section 5. Enfin, avant de conclure (section 7), nous aborderons les aspects de contournement des *IDS* (*Intrusion Detection System*) dans la section 6.

2 Principe de conception

À l'origine de *SinFP*, une question : est-il possible, dans les pires conditions réseau, d'identifier de manière fiable un système d'exploitation ? Les pires conditions réseau sont les suivantes :

1. un seul port *TCP* ouvert ;
2. tous les autres ports (*TCP* et *UDP*) en mode rejeté ;
3. un dispositif de filtrage à état pour protéger ce port ;
4. un dispositif de normalisation de trames.

Dans une telle configuration, seules les trames conformes au sens des standards de l'*IETF* (*Internet Engineering Task Force*) atteignent leur cible, et permettent d'obtenir une réponse. Il n'est alors possible que d'utiliser des requêtes standard, et uniquement en *TCP*, afin de construire une signature fiable.

Pour choisir la première requête, avec comme contrainte qu'elle atteigne la cible quelque soit le dispositif de filtrage, nous avons capturé celle émise par l'appel système `connect()` (en l'occurrence le `connect()` d'un système *Linux 2.4.x*). Elle possède de nombreuses options *TCP*. La deuxième requête est une copie de la première, mais en supprimant toutes les options. Ces deux requêtes vont provoquer l'émission de deux réponses *TCP SYN+ACK*. Afin de créer une signature possédant le plus de caractéristiques possibles, nous avons ajouté une troisième requête, de manière à forcer la cible à émettre une réponse *TCP RST+ACK*. Cette requête n'a aucune option *TCP*, et possède les drapeaux *TCP SYN+ACK*. Elles sont toutes destinées au même port *TCP* ouvert.

Une fois les trames pertinentes identifiées, tous les champs de la réponse pouvant varier d'une implémentation de pile *TCP/IP* à une autre sont récupérés. Ceci inclut tous les champs *IP* et *TCP*, voire le contenu de la couche applicative[2]. Certains champs ayant des valeurs aléatoires, telles les *timestamp* dans les options *TCP*, il est indispensable de les formater, nous verrons de quelle manière dans la section 3.1.2.

Une fois la signature obtenue, un algorithme de recherche de correspondance dans une base de référence doit être écrit. *SinFP* utilise un algorithme de type moteur de recherche Web. Il s'agit de rechercher l'intersection entre plusieurs domaines.

Une fois la partie prise d'empreinte active implémentée, un portage vers une prise d'empreinte passive a été réalisé. Puis vint le portage vers *IPv6* [10], qui

[2] HP-UX 11.x ajoute la chaîne `No TCP` en couche applicative lors de l'émission d'un *TCP RST+ACK*.

se traduisit par une simple recherche d'équivalence entre les champs *IPv4* [9] et les champs *IPv6*.

En résumé, voici la liste des fonctionnalités supportées par *SinFP* :

1. prise d'empreinte active et passive sur *IPv4* ;
2. prise d'empreinte active et passive sur *IPv6* ;
3. si une correspondance *IPv6* n'est pas trouvée dans la base, possibilitié d'utiliser les signatures *IPv4* ;
4. analyse passive en mode en ligne et hors ligne ;
5. analyse active rejouable hors ligne sur un fichier *pcap* généré en ligne ;
6. possibilité de ne jouer qu'une partie des requêtes pour contourner les *IDS* ;
7. algorithme de recherche heuristique, permettant d'identifier la cible même si la pile *TCP/IP* a été légèrement modifiée ;
8. une base de données de signatures en *SQL* au format *SQLite* [21] ;
9. intégration facilitée dans tout autre programme parce qu'écrit sous forme de module *Perl*.

Chacun de ces concepts est approfondi dans la suite de ce document.

3 Prise d'empreinte active

Le principe de la prise d'empreinte active est d'émettre des requêtes dans un format connu et spécialement choisi vers une cible, et d'analyser les réponses afin de construire la signature la plus discriminante possible. *SinFP* émet au maximum trois requêtes, toutes standard, à destination du même port *TCP* [11]. La première est une requête *TCP SYN* sans option (test *P1*), la seconde une requête *TCP SYN* avec de nombreuses options (test *P2*), et la troisième une requête *TCP SYN+ACK* (test *P3*).

Les deux premières requêtes provoquent l'émission par la cible[3] de deux réponses *TCP* avec les drapeaux *SYN+ACK*, deuxième étape de l'établissement d'une connexion *TCP*. La troisième provoque[4] l'émission par la cible d'une réponse *TCP RST+ACK*.

Aujourd'hui, l'approche la plus utilisée [8] est d'émettre un grand nombre de requêtes, sur des ports et des protocoles de couche transport différents. Il en résulte une signature qui peut potentiellement être l'assemblage de différents systèmes présents entre la source et sa cible. Une telle signature n'est pas fiable. Elle est en effet composée de l'analyse de réponses appartenant à différents systèmes et il est impossible de savoir lesquelles viennent du système ciblé. *SinFP* adresse ces problématiques.

Si une prise d'empreinte est lancée avec les trois requêtes (mode par défaut), nous sommes dans la même situation problématique vue précédemment,

[3] Certaines piles *TCP/IP*, ou dispositifs de filtrages, ne répondent pas à une requête *TCP SYN* ne possédant pas au minimum l'option *MSS*.

[4] Sauf dans le cas où un dispositif de filtrage à états est présent.

essentiellement à cause du test *P3*. En effet, certains pare-feux gérant l'état des connexions répondent à la place de la cible, et le *TCP RST+ACK* de réponse n'appartiendra pas à celle-ci. Il est donc possible de ne jouer que certains des trois tests[5][6]. Par exemple, si la réponse à la troisième requête (un segment *TCP RST+ACK*) est émise par un dispositif de filtrage en coupure, nous pouvons avoir une signature finale qui aura comme correspondance un mauvais système. Dans ce genre de cas, il convient de lancer *SinFP* en ne jouant que les tests *P1* et *P2*.

Dans tous les cas, l'absence de réponse à une requête n'est jamais considérée comme un élément de signature. Seules les réponses obtenues sont utiles à la création d'une signature la plus discriminante possible. Voir la figure 1 pour un exemple d'usage de l'outil.

```
% sinfp.pl -ai 10.100.0.22 -p 22
P1: B11013 F0x12 W65535 O0204ffff M1460
P2: B11013 F0x12 W65535 O0204ffff0103030001010 80affffffff44454144 M1460
P3: B11020 F0x04 W0 O0 M0
IPv4: BH1FH0WH0OH0MH0/P1P2P3: BSD: Darwin: 8.6.0
```

Fig. 1. Exemple de prise d'empreinte active

Voyons maintenant le format des requêtes, ainsi que la manière dont les réponses sont analysées. Une bonne connaissance des champs des en-têtes *IPv4*, *IPv6* et *TCP* est vivement conseillée.

3.1 Sur *IPv4*

Les trois requêtes sont complètement conformes aux standards. Elles vont provoquer trois réponses de la part de la cible. Une fois ces réponses reçues, une signature est générée. Dans les sections suivantes, nous verrons de quelle manière nous passons des réponses obtenues par l'envoi des requêtes vers une signature complète (nous décrypterons le format des signatures en section 5.1).

3.1.1 Analyse des en-têtes *IPv4*

Le champ *TTL* : certains systèmes[7] ne mettent pas la même valeur de *TTL* lors de l'émission d'un segment *TCP SYN+ACK* et d'un segment *TCP RST+ACK*. Aussi, nous analysons la différence entre le *TTL* de réponse au test *P3*, avec celui de réponse au test *P2*. Si le *TTL* de la réponse à *P3* est différent

[5] Via le paramètre `-2` de la ligne de commande pour ne jouer que les tests *P1* et *P2*.
[6] Via le paramètre `-1` de la ligne de commande pour ne jouer que le test *P2*.
[7] *SunOS* en fait partie.

du *TTL* de réponse à *P2*, on positionne une constante à 0. Sinon, à 1. Cette constante est toujours 1 pour la réponse à *P1* et *P2*.

Le champ *ID* : c'est une comparaison entre l'*ID* de la requête, et l'*ID* de la réponse. Si l'*ID* de la réponse est 0, nous positionnons une constante à 0. Si il est égal à celui de la requête, nous le positionnons à 2. Si il est incrémenté de 1, nous le positionnons à 3. Dans les autres cas, il est positionné à 1. L'*ID* pouvant être modifié par un dispositif de filtrage, nous verrons comment l'algorithme de recherche de signature traite le problème dans la section 5.3.

Le bit *Don't Fragment* : si la réponse possède le bit *Don't Fragment*, une constante est positionnée à 1, sinon à 0.

3.1.2 Analyse des en-têtes *TCP*

Le champ numéro de séquence : c'est également une comparaison, mais entre le numéro de séquence de la requête, et celui de la réponse. Si le numéro de la réponse est 0, nous positionnons une constante à 0. Si il est égal à celui de la requête, nous le positionnons à 2. Si il est incrémenté de 1, nous le positionnons à 3. Dans les autres cas, il est positionné à 1.

Le champ numéro d'acquittement : nous appliquons la même analyse que pour le numéro de séquence, mais en l'appliquant au numéro d'acquittement.

Les drapeaux *TCP* sont copiés tel quel dans la signature, de même que la taille de la fenêtre.

Enfin, le champ des options est également copié dans la signature, mais avec les modifications suivantes :

1. la valeur du *MSS* (*Maximum Segment Size*) (si présent) est extraite pour former un élément de signature, et remplacée par `ffff` dans l'élément option ;
2. si les valeurs des *timestamp* sont supérieurs à zéro, nous les remplaçons par `ffffffff`.

La valeur du *MSS* est extraite afin de simplifier l'écriture des expressions rationnelles (nous y reviendrons en section 5.3). De même pour les *timestamp*, nous voulons juste savoir si elles sont différentes de la valeur 0.

Une fois toutes ces analyses effectuées, nous obtenons la signature de la cible. Nous verrons cela plus en détails dans la section 5.1.

3.2 Sur *IPv6*

La différence avec la prise d'empreinte sur *IPv4* ne porte que sur les entêtes *IPv6*, puisque la partie *TCP* des requêtes/réponses ne change pas. Pour porter *SinFP* sur *IPv6*, une équivalence entre les champs disponibles en *IPv4* est effectuée vers les champs disponibles en *IPv6*.

L'équivalence des champs est la suivante :

1. *IPv4 ID* => *IPv6 Flow Label* ;
2. *IPv4 TTL* => *IPv6 Hop Limit* ;
3. drapeau (*Don't Fragment*) => *IPv6 Traffic Class*.

C'est la seule différence entre la prise d'empreinte sur *IPv6* et celle sur *IPv4*.

3.3 Limitations du mode actif

Lorsque les options *TCP* contenues dans la réponse à *P2* sont peu nombreuses (quand l'élément option de la réponse à *P2* est égal à `0204ffff`), l'entropie de la signature devient trop faible. En effet, les options *TCP* sont le caractére le plus discriminant composant une signature, chaque système n'implémentant pas toutes les options et ne les rangeant pas dans le même ordre. Si les options sont juste limitées à `0204ffff`, nous avons une unique option à analyser (la taille du *MSS*). Un risque d'erreur important intervient et un message d'attention est affiché pour prévenir l'utilisateur.

De temps à autres, aucune réponse n'est obtenue pour le test *P1* alors qu'en général le type de système cible répond. Dans ce cas, soit une signature inconnue est retournée, soit c'est un résultat erroné qui est retourné. Aussi, il est nécessaire de relancer l'outil en ne jouant que le test *P2*. Dans la plupart des cas, un résultat correct est retourné.

Dans d'autres cas, aucun résultat n'est retourné, alors que toutes les réponses aux requêtes ont été reçues. Il est alors nécessaire de relancer l'identification en activant les masques de déformation avancés[8]. Nous fournirons plus de détails sur les masques de déformation dans la section 5.3.

3.4 Autres fonctionalités

À chaque prise d'empreinte, un fichier de trace au format *pcap* est généré. Il permet de rejouer l'algorithme de recherche de correspondance. Il est alors possible d'utiliser une base de signature différente, ou encore d'utiliser un masque de déformation personnalisé (voir la section 5.3).

Le fichier généré par défaut est anonymisé. Si une signature inconnue est rencontrée, merci de la poster sur la liste *SinFP discuss* [12] avec la version exacte du système ciblé, ainsi que ce fichier.

Ce fichier sert également à rejouer la recherche de correspondance dans la base. En effet, de temps à autres, aucune correspondance n'est trouvée. Il suffit de relancer *SinFP* en activant les masques de déformation avancés et de préciser le fichier *pcap* a analyser[9].

4 Prise d'empreinte passive

Ce mode fonctionne aussi bien sur un fichier *pcap* qu'en écoutant le réseau. Il est idéal pour l'intégration dans un *IDS* ou dans un pare-feux.

Malheureusement, le portage du mode actif vers le mode passif n'est pas aussi immédiat que le portage de la prise d'empreinte active en *IPv4* vers celui en *IPv6*. En effet, tout ce qui est comparaison entre la requête et la réponse ne peut être effectué, étant donné qu'il n'y a pas de requêtes.

[8] Via le paramètre `-H` de la ligne de commande.
[9] Via le paramètre `-f` de la ligne de commande.

De plus, la base de signatures ne connaît que les réponses aux requêtes *SYN*, donc des segments *TCP SYN+ACK* ou *TCP RST+ACK*. Pour pouvoir aussi analyser les segments *TCP SYN*, une modification doit être apportée à la trame capturée, nous y reviendrons dans la section 4.3. Voir la figure 2 pour un exemple d'usage en mode passif.

```
% sinfp.pl -Pf ~/sinfp4-passive.pcap
10.100.0.1:80 > 10.100.0.68:39503 [SYN|ACK]
P2: B10111 F0x12 W5672 O0204ffff0402080affffffffffffffff01030306 M1430
IPv4: BH0FH0WH1OH0MH1/P2: GNU/Linux: Linux: 2.6.x
```

Fig. 2. Exemple de prise d'empreinte en mode passif

Dans les sections suivantes, nous décrivons de quelle manière nous transformons une signature passive pour qu'elle puisse être comparée avec une signature active.

4.1 Analyse des en-têtes *IPv4*

La constante concernant le *TTL* est toujours positionnée à 1. Cela ne nuit pas à la recherche de correspondance, étant donné que les trames que nous analysons en mode passif sont équivalentes aux réponses du test *P2* du mode actif. Ces réponses possèdent également la constante du champ *TTL* positionnée à 1. La constante concernant l'*ID* est positionnée à 1 si l'*ID* de la trame interceptée est supérieur à 0, sinon à 0. En mode actif, la constante peut être supérieure à 1. Nous verrons de quelle manière nous contournons le problème en section 4.4. La méthode d'analyse concernant le bit *Don't Fragment* ne nécessite aucune modification.

4.2 Analyse des en-têtes *IPv6*

Étant donné l'équivalence entre les en-têtes *IPv4* et *IPv6*, la même transformation est appliquée en *IPv4* et en *IPv6*.

4.3 Analyse des en-têtes *TCP*

La constante concernant le numéro de séquence est positionnée à 1 si il a une valeur supérieure à 0. La constante concernant le numéro d'acquittement suit la même logique. De même que précédemmant, nous n'avons accès qu'aux réponses, et nous ne ponvons pas faire de comparaison. Les constantes obtenues dans le mode actif pouvant avoir des valeurs supérieurs à 1, nous verrons en section 4.4 la méthode de contournement. Concernant les drapeaux de l'en-tête *TCP*, si il s'agit d'un *SYN+ACK*, aucune modification n'est faite par rapport à la version active. Par contre, si il s'agit d'un *SYN*, nous le remplaçons par

un *SYN+ACK*, afin qu'il corresponde aux signatures actives de la base. Nous devons modifier le drapeau *TCP* puisque nous n'avons que des *TCP SYN+ACK* (et des *TCP RST+ACK* dans la base de données. Grâce à cette modification, nous pouvons aussi comparer les *TCP SYN* capturés sur le réseau. La méthode d'analyse des autres champs ne change pas.

4.4 La recherche de correspondance

Il subsiste encore une difficulté. En effet, les signatures stockées en base sont des signatures de prise d'empreinte active. À ce titre, elles ont un format non compatible avec la prise d'empreinte passive. En effet, certaines constantes de la signature, issues d'une comparaison entre une requête et sa réponse, ont des valeurs supérieures à 1. Comme nous n'avons accès qu'aux réponses, nous ne pouvons comparer. La solution retenue est de modifier à la volée les signatures extraites de la base lors de la recherche. La modification est simple, les constantes ayant une valeur supérieure à 1 sont remplacées par la valeur 1.

Puisque la signature passive ressemble maintenant à une signature active, l'algorithme de recherche ne nécessite aucune modification. En outre, seules les signatures prises en mode actif existent dans la base de données. Le portage vers la prise d'empreinte passive est terminé.

4.5 Limitations du mode passif

De manière générale, la prise d'empreinte passive souffre d'une limitation, et *SinFP* ne déroge pas à cette règle. En effet, il n'y a aucune requête émise vers une cible, il n'y a donc aucun contrôle sur le format de la réponse. Nous savons que pour obtenir une réponse en mode actif, nous choisissons nos requête d'une certaine manière. En mode passif, nous n'avons pas ce choix, c'est le système d'exploitation, source de la connexion *TCP*, qui choisit le format de la requête. En mode actif, les trames sont construites par l'outil, et le système n'a pas le choix de leurs formats.

Si le système d'exploitation source a une pile *TCP/IP* qui implémente peu d'options *TCP*, la réponse que l'outil de prise d'empreinte passive va obtenir ne sera pas celle que l'on aurait obtenu en mode actif. Il y aura une certaine déformation. C'est ici qu'intervient l'algorithme de recherche de correspondance qui, dans *SinFP*, est capable d'accepter de fortes déformations pour contourner ces limitations.

5 Algorithme de recherche de correspondance

Cet algorithme est très sembable à celui utilisé dans les moteurs de recherche Web [13], puisqu'il s'agit d'un algorithme de recherche d'intersection entre plusieurs domaines. Pour bien comprendre son fonctionnement, il est nécessaire d'introduire le format des signatures, le format de la base, ce que sont les masques de déformation, et finalement la recherche de correspondance.

5.1 Format des signatures

Une signature (voir figure 3) est composée de trois réponses (*P1(R)*, *P2(R)* et *P3(R)*). Chaque réponse possède 5 d'éléments. Une signature est ainsi composée de 3 x 5 éléments, soit un total de 15.

```
B11113 F0x12 W65535 O0204ffff M1460
B11113 F0x12 W65535 O0204ffff010303000101080affffffff44454144 M1460
B11020 F0x04 W0 O0 M0
```

Fig. 3. Signature du système *Darwin 8.6.0*

Ceci peut sembler faible. En effet, d'autres outils comme *nmap* [14] possédent bien plus d'éléments [15]. Pourtant, dans la pratique, cela se montre bien suffisant pour identifier de manière sûre un système d'exploitation, et souvent, 5 éléments sont suffisants[10].

Chaque réponse à une requête possède les éléments suivants :

1. B : une liste de constantes (exemple : `B11013`) ;
2. F : les drapeaux *TCP* (exemple : `F0x12`) ;
3. W : la taille de la fenêtre *TCP* (exemple : `W65535`) ;
4. O : les options *TCP*, reécrites pour ignorer les valeurs changeantes telle les *timestamp* (exemple : `O0204ffff010303000101080affffffff44454144`) ;
5. M : la taille du *MSS* (exemple : `M1460`).

Dans la figure 1, les trois premières lignes sont les trois réponses (*P1(R)*, *P2(R)* et *P3(R)*) aux trois requêtes (*P1*, *P2* et *P3*). Elles forment la signature d'un système. La ligne finale est la correspondance trouvée en base pour cette signature.

Cette dernière ligne est composée de plusieurs éléments. Le premier est le type de correspondance trouvé (`BH1FHOWHOOHOMHO`) ; il s'agit du masque de déformation (nous les verrons bientôt, dans la section 5.3) qui a permis de trouver la correspondance. Le deuxième élément (`P1P2P3`) indique quelles sont les réponses qui ont trouvé une correspondance dans la base, pour le masque de déformation sélectionné. Ici, les trois requêtes ont trouvé une correspondance. Enfin, nous avons la classe du système (`BSD`), son nom (`Darwin`) et sa version (`8.6.0`).

5.2 La base de signatures

Chacun des quinze éléments possède un *ID* dans une base de données relationnelle. Ces éléments sont communs à toutes les signatures de la base, seule la nature unique de la chaîne de caractère détermine son *ID*. En effet, chaque

[10] En ne lancant que le test *P2*, qui apporte le plus d'information sur la cible.

élément pris séparément est indépendant d'une signature, donc d'un système d'exploitation.

Par exemple, l'élément `W65535` peut être commun à de nombreux systèmes. Ainsi, tous les systèmes ayant une valeur de 65535 pour la taille de la fenêtre *TCP* possèdent l'*ID* de cet élément dans leur signature. Une signature, d'un point de vue base de données relationnelle, est simplement une suite d'*ID*.

Chaque système d'exploitation (avec sa version) possède une unique signature dans la base. Dans *SinFP*, nous n'ajoutons pas une signature pour tel ou tel système, même si la cible a, par exemple, désactivé une option *TCP*. L'algorithme de recherche, grâce aux masques de déformation (que nous verrons dans la prochaine section), est là pour gérer les cas de personnalisation des piles *TCP/IP*.

Une base de signature se doit d'être propre. Toutes les signatures ne sont pas éligibles à une inclusion dans celle-ci. En effet, ajouter une signature erronée entraînerait de gros risques d'erreurs dans l'identification. Aussi, si un doute subsiste quant à la possibilité qu'un dispositif de filtrage en coupure ait modifié la signature, celle-ci n'est pas incluse.

Pour être éligible à l'inclusion dans la base, il est obligatoire que la signature ait été prise dans des conditions parfaites :

1. aucun dispositif de filtrage en coupure ;
2. aucun routeur en coupure ;
3. au moins un port *TCP* ouvert.

Pour résumer, les conditions parfaites sont soit un accés réseau sur le même brin ethernet, soit la cible tourne sur une machine locale (dans une machine virtuelle par exemple). Toute autre condition réseau laisserait persister un doute, et finirait à plus ou moins court terme par entraîner une base incohérente. *nmap* en a fait les frais et profite du développement de sa deuxième génération de prise d'empreinte active pour refaire une base propre.

5.3 Les masques de déformation

La réponse à une requête peut être modifiée par un dispositif de filtrage, qu'il soit sur le chemin, ou directement sur la cible. Ainsi, *SinFP* introduit les masques de déformation. Ceux-ci sont implémentés par des expressions rationnelles. Chaque élément de la signature possède deux expressions rationnelles, en plus de la valeur prise dans les conditions parfaites (appelée heuristique 0). L'une de niveau 1 (appelée heuristique 1), l'autre de niveau 2 (appelée heuristique 2). Un masque de déformation est appliqué sur une signature de référence prise dans les conditions idéales (voir figure 4) à l'extraction de la base.

Chaque type d'élément autorise certaines déformations. Par exemple, les déformations autorisées pour l'élément *O* ne sont pas les mêmes que celles pour l'élément *F*. Dans le cas de *F*, aucune déformation n'est autorisée.

Par exemple, une valeur fréquemment modifiée sur le chemin est le *TCP MSS*. Dans les conditions parfaites, il a souvent une valeur de 1460. Mais régulièrement, nous obtenons une valeur de 1430, probablement dû à un routeur ayant un *MTU*

Signature de référence d'origine (masque parfait `HEURISTIC0`) :

```
B10113 F0x12 W5840 O0204ffff M1460
B10113 F0x12 W5792 O0204ffff0402080affffffff4445414401030306 M1460
B10120 F0x04 W0 O0 M0
```

Signature de référence après application du masque `BH1FH0WH1OH0MH1` :

```
B...13 F0x12 W5[789].. O0204ffff M1[34]..
B...13 F0x12 W5[678].. O0204ffff0402080affffffff4445414401030306 M1[34]..
B...20 F0x04 W0 O0 M0
```

Signature référence après application du masque `BH1FH0WH2OH1MH2` :

```
B...13 F0x12 W\d+ O0204ffff M\d+
B...13 F0x12 W\d+ O0204ffff(?:0402)?(?:080affffffff44454144)?(?:01)?(?:030306)? M\d+
B...20 F0x04 W0 O0 M0
```

Fig. 4. Déformation de la signature de référence du système *Linux 2.6.x* après l'application de trois masques

(*Maximum Transmission Unit*) inférieur à cette valeur. L'élément de base pour cette valeur étant `M1460`, nous écrivons la valeur heuristique 1 (*H1*) `M1[34]..`, et la valeur heuristique 2 (*H2*) `M\d+`. La valeur *H1* autorise une déformation pour le *TCP MSS* allant d'une valeur de 1300 à 1499. En *H2*, nous ignorons simplement sa valeur. Chaque type d'élément posséde des types d'heuristique différents.

Nous avons vu que chaque élément d'une signature est unique en base, et possède un également un *ID* unique. Les masques de déformation étant reliés à un élément, ils sont écrits pour un élément donné, et sont décorrélés de la signature d'un système. Toutefois, il reste possible d'écrire des valeurs de masques spécifiquement pour un système.

Un masque de déformation est l'association de tous les masques pour chaque élément d'une signature. Ainsi, en reprenant notre exemple de prise d'empreinte active pour le système *Darwin* (figure 1), la correspondance trouvée est `BH1FH0WH0OH0MH0`. Une correspondance parfaite étant `BH0FH0WH0OH0MH0`, écrite plus simplement `HEURISTIC0`. C'est le masque le plus fiable. Dans notre exemple, nous n'avons pas trouvé de correspondance parfaite, mais juste une légère déformation sur l'élément *B*, une déformation en heuristique 1 (`BH1`, au lieu de `BH0`). Le résultat est considéré comme très fiable, étant donné que les autres éléments n'ont subi aucune déformation.

Dans *SinFP*, il existe deux grands niveaux de masques de déformation. Le premier niveau contient des masques qui entraînent très peu d'erreurs d'identification. Ce sont les masques standards, au nombre de huit (dans la version *2.06* [16]). Le second niveau[11] intervient si aucune correspondance n'est trouvée en utilisant les masques standards. Mais il entraîne assez souvent des erreurs,

[11] Activable par le paramètre `-H` de la ligne de commande.

autorisant une plus grande déformation sur les réponses. Ces masques sont les masques avancés, au nombre de quatorze. Ils nécessitent une grande maîtrise de l'outil pour conclure si la réponse obtenue est fiable ou non.

Tous ces masques sont obtenus de manière empirique. À chaque utilisation de l'outil, certaines déformations sur les réponses sont relevées. Si une déformation intervient très souvent, un masque spécifique pour cette déformation est ajouté dans le code. Les déformations qui interviennent souvent sont principalement dues à des routeurs qui modifient le *MTU* ou des pare-feux qui modifient les en-têtes *IP*.

Les masques de déformation sont classés du moins déformant, vers le plus déformant. `HEURISTIC0` (`BH0FH0WH0OH0MH0`) est le masque qui accepte le moins de déformations, et celui acceptant le plus de déformations (par conséquent le moins fiable) est le masque `HEURISTIC2` (`BH2FH2WH2OH2MH2`). Il existe aussi le masque intermédiaire `HEURISTIC1` (`BH1FH1WH1OH1MH1`), qui lui aussi est classé dans les masques avancés. Entre ces trois grands niveaux, nous avons les masques obtenus de manière empirique.

5.4 La recherche de correspondance dans la base

Chaque élément d'une signature requiert une correspondance dans la base. Pour chaque élément de *P1* (*E1*, *E2*, ..., *E5*), nous cherchons la liste des *ID* de signatures possédant cette caractéristique. L'algorithme recherche ensuite les *ID* de signatures qui se trouvent dans toutes les listes (l'intersection des domaines *E1(P1)*, *E2(P1)*, ..., *E5(P1)*) obtenues pour chaque élément. Cette intersection nous donne le domaine *I(P1)*. Nous recommencons pour *P2* et *P3*, afin d'obtenir *I(P2)* et *I(P3)*.

La correspondance finale est l'intersection des domaines *I(P1)*, *I(P2)* et *I(P3)*, c'est-à-dire les *ID* de signatures qui se trouvent dans ces trois domaines. Si il n'y a pas de correspondance, on tente une correspondance entre les domaines *I(P1)* et *I(P2)* uniquement. Si il n'y a toujours pas de résultat, nous appliquons la même recherche, mais avec le prochain masque de déformation dans la liste des masques (du moins déformant, vers le plus déformant). La recherche est stoppée dès qu'une (ou plusieurs) correspondance est trouvée pour un masque donné, en faisant le tour de toutes les signatures.

En termes mathématiques, cet algorithme est décrit par le système :

$I(P1) = E1(P1) \bigcap E2(P1) \bigcap ... \bigcap E5(P1)$
$I(P2) = E1(P2) \bigcap E2(P2) \bigcap ... \bigcap E5(P2)$
$I(P3) = E1(P3) \bigcap E2(P3) \bigcap ... \bigcap E5(P3)$
$I = I(P1) \bigcap I(P2) \bigcap I(P3)$
Si I est nul :
$I = I(P1) \bigcap I(P2)$

En mode passif, l'algorithme est décrit par le système :

$$I = E1(P2) \bigcap E2(P2) \bigcap ... \bigcap E5(P2)$$

Il n'y a aucune différence entre la recherche en *IPv4* et en *IPv6*. Nous l'avons vu précédemment, il existe une équivalence entre les en-têtes *IPv4* et *IPv6*. Ainsi, il est possible de directement utiliser les signatures *IPv4* lors d'une prise d'empreinte en *IPv6*. Si aucune signature *IPv6* n'existe dans la base pour la cible, il est possible d'utiliser les signatures *IPv4*[12] lors de la recherche de correspondance. Lors de nos expérimentations, nous constatons que ce mode de "compatibilité" se montre très fiable.

5.5 Utilisation avancée des masques de déformation

Nous l'avons vu précédemment, les masques sont ajoutés de manière empirique[13]. Par exemple, le masque `BH0FH0WH2OH0MH0` était[14] nécessaire pour identifier le système d'exploitation du serveur `www.openbsd.org` (figure 5).

SinFP exécuté en mode actif avec un masque de déformation personnal isé afin d'identifier le serveur `www.openbsd.org` :

```
% sinfp.pl -ai www.openbsd.org -p 80 -A BH0FH0WH2OH0MH0
P1: B11113 F0x12 W536 O0204ffff M536
P2: B11113 F0x12 W1460 O0101080affffffff4445414401030300020 4ffff M1460
P3: B01120 F0x04 W0 O0 M0
IPv4: BH0FH0WH2OH0MH0/P1P2P3: Unix: SunOS: 5.6
```

La signature du système *SunOS 5.6* contenue dans la base de données :

```
B11113 F0x12 W9112 O0204ffff M536
B11113 F0x12 W10136 O0101080affffffff4445414401030300020 4ffff M1460
B01120 F0x04 W0 O0 M0
```

Fig. 5. Prise d'empreinte du serveur `www.openbsd.org`

Le serveur utilisé par `www.openbsd.org` avait une réponse quasiment identique au système *SunOS 5.6*, mais avec une taille de fenêtre *TCP* de 536 pour *P1* et 1460 *P2*. En utilisant le masque précité, nous ignorons les valeurs de la taille de la fenêtre (`WH2`) contenue dans les réponses. Ainsi, nous trouvons une correspondance avec une heurisitique faible, et le système est bien identifié en *SunOS 5.6* (voir figure 5). Le serveur `www.openbsd.org` n'est pas le seul dans

[12] Via le paramètre `-4` de la ligne de commande.

[13] Le paramètre `-A` de la ligne de commande permet de tester des masques de déformation, avant de les ajouter dans le code.

[14] Était, puisque qu'aujourd'hui il n'est plus nécessaire, le serveur en question semble avoir modifié son architecture réseau, et ce masque n'est plus utile pour cette cible.

ce cas, ce qui a justifié l'ajout de ce masque dans le code de *SinFP*. Nous ne pouvons que supposer qu'il s'agit d'un dispositif de filtrage et/ou routage qui modifie la taille de la fenêtre *TCP*, ou encore une personnalisation de la pile *TCP/IP*.

Tout comme pour les signatures, il faut bien choisir les formats de masques à ajouter dans l'outil. Si des masques acceptant de trop nombreuses déformations sont ajoutés, les signatures risques de toutes se ressembler, entraînant un résultat composé de nombreux systèmes différents. Le choix d'ajouter un nouveau masque est un processus manuel qui requiert une grande expertise de l'outil.

6 Mode invisibilité aux *IDS*

SinFP utilise des requêtes entièrement standard, il est donc difficile d'écrire des règles pour un *IDS* afin de détecter une utilisation de *SinFP* sur un réseau. C'est possible, lorsque l'outil est utilisé en mode par défaut, il lance deux requêtes *TCP SYN* et une requête *TCP SYN+ACK* vers un même port en un court laps de temps. Ces évènements peuvent être mis dans une règle de détection d'un *IDS*.

C'est ici qu'interviennent les autres modes de l'outil :

1. `-3` : lance toutes les requêtes (défaut) ;
2. `-2` : ne lance que les deux premières requêtes ;
3. `-1` : ne lance que la deuxième requête.

Lancer les deux premières requêtes reste identifiable par un *IDS*, même si il y a des risques de faux positif. Par contre, ne lancer que la deuxième requête devient bien plus difficile à détecter, étant donné le caractère standard de la requête *TCP SYN*. En outre, le système servant à lancer l'identification émettra un segment *TCP RST* lorsque la cible répondra par un *TCP SYN+ACK*, puisque la requête émise par *SinFP* ne vient pas du noyau système lui-même. En effet, en mode actif, *SinFP* forge ses requêtes et le système n'a pas connaissance d'avoir demandé l'établissement d'une connexion. En mode passif, c'est le système lui-même qui établit la connexion et ce problème disparait.

SinFP exécuté en mode passif dans un terminal pendant qu'une connexion sur `www.sstic.org` est effectuée par un navigateur Web :

```
% sinfp.pl -PF 'host www.sstic.org and src port 80'
88.191.41.247:80 > 192.168.0.101:60623 [SYN|ACK]
P2: B11111 F0x12 W5792 O0204ffff0402080affffffffffffffff01030305 M1460
IPv4: BH1FHOWHOOHOMHO/P2: GNU/Linux: Linux: 2.6.x
```

Fig. 6. Exemple de prise d'empreinte en mode actif/passif

Ainsi, il existe un mode mixte, à la fois actif et passif. Il suffit d'utiliser *SinFP* en écoute sur le réseau, en mode passif, et d'établir une connexion *TCP*, par

exemple par l'intermédiaire de son navigateur (voir un exemple d'usage dans la figure 6). Sur cet exemple, le résultat est très fiable, seule une légère déformation existe sur l'élément *B*, une déformation en heuristique 1 (`BH1`).

7 Conclusion

Dans cet article, nous avons décrit en profondeur les choix et l'implémentation de *SinFP*. Nous avons montré qu'il était possible d'unifier la prise d'empreinte active et passive dans un même outil, avec un unique type de signature prises de manière active. De plus, *SinFP* est le premier outil public à implémenter la prise d'empreinte sur *IPv6*.

L'algorithme de recherche de correspondance de type moteur de recherche Web donne d'excellents résultats, surtout associé aux masques de déformation. Cependant, certains cas peuvent entraîner des erreurs d'identification, et nous avons des solutions pour les limiter. Ces solutions ne sont pas encore implémentées dans l'outil, et donneront lieu, peut-être, à une nouvelle publication.

En attendant, si vous souhaitez comparer la prise d'empreinte active de *nmap* à celle de *SinFP*, consultez les sites suivants [17][18][19][20].

Références

1. Prise d'empreinte active des systèmes d'exploitation
   `http://www.gomor.org/article/misc7`
2. SinFP OS fingerprinting tool
   `http://www.gomor.org/cgi-bin/sinfp.pl`
3. `Net::SinFP 0.92`
   `http://search.cpan.org/~gomor/Net-SinFP-0.92/`
4. Stateful Passive Fingerprinting for Malicious Packet Identification
   `http://www.andrew.cmu.edu/user/xsk/XenoKovahThesis.pdf`
5. IPv6 Neighbor Discovery Protocol based OS fingerprinting
   `http://hal.inria.fr/docs/00/16/99/90/PDF/technical_report_fingerprinting.pdf`
6. A Hybrid Approach to Operating System Discovery using Answer Set Programming
   `http://ieeexplore.ieee.org/iel5/4258513/4258514/04258556.pdf?tp=&isnumber=&arnumber=4258556`
7. Toward Undetected Operating System Fingerprinting
   `http://www.usenix.org/events/woot07/tech/full_papers/greenwald/greenwald.pdf`
8. Remote OS Detection using TCP/IP Fingerprinting (2nd Generation)
   `http://insecure.org/nmap/osdetect/`
9. Internet Protocol (version 4)
   `ftp://ftp.rfc-editor.org/in-notes/rfc791.txt`
10. Internet Protocol (version 6)
    `ftp://ftp.rfc-editor.org/in-notes/rfc2460.txt`

11. Transmission Control Protocol
ftp://ftp.rfc-editor.org/in-notes/rfc793.txt
12. sinfp – News about SinFP
http://lists.gomor.org/mailman/listinfo/sinfp
13. Analyse fine : bornes inférieures et algorithmes de calculs d'intersection pour moteurs de recherche
http://www.cs.uwaterloo.ca/~jbarbay/Recherche/Publishing/Publications/these.pdf
14. Nmap - Free Security Scanner For Network Exploration and Security Audits
http://insecure.org/nmap/
15. TCP/IP Fingerprinting Methods Supported by Nmap
http://insecure.org/nmap/osdetect/osdetect-methods.html
16. Net::SinFP 2.06
http://search.cpan.org/~gomor/Net-SinFP-2.06/
17. SinFP vs Nmap
http://www.computerdefense.org/2006/12/04/sinfp-vs-nmap/
18. Nmap vs SinFP
http://www.computerdefense.org/2006/12/08/nmap-vs-sinfp/
19. Introduction and Comparison with Nmap 4.10, Part I
http://www.phocean.net/?p=13
20. Comparison with Nmap 4.20, Part II
http://www.phocean.net/?p=14
21. SQLite Home Page
http://www.sqlite.org/