

Robustesse et Identification des Applications Communicantes

Jérôme François

▶ **To cite this version:**

Jérôme François. Robustesse et Identification des Applications Communicantes. Réseaux et télécommunications [cs.NI]. Université Henri Poincaré - Nancy 1, 2009. Français. NNT : . tel-01748663v3

HAL Id: tel-01748663

https://theses.hal.science/tel-01748663v3

Submitted on 5 Jan 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Nancy-Université
Université Henri Poincaré

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Robustesse et Identification des Applications Communicantes

THÈSE

présentée et soutenue publiquement le 7 décembre 2009

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1
(spécialité informatique)

par

Jérôme François

Composition du jury

Rapporteurs :	Hervé Debar	Professeur, Télécom & Management SudParis, Évry
	Philippe Owezarski	Chargé de Recherche, CNRS, Toulouse
Examinateurs :	Olivier Festor	Directeur de Recherche, INRIA, Nancy
	Dominique Méry	Professeur, Université Henri Poincaré, Nancy
	Cristina Nita-Rotaru	Professeur Associé, Purdue University, États-Unis
	Radu State	Chercheur, Université du Luxembourg, Luxembourg

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Loria

Mis en page avec la classe thloria.

Remerciements

Je tiens tout d'abord à remercier les rapporteurs et les examinateurs de cette thèse pour l'intérêt qu'ils portent à mes travaux de recherche et pour avoir accepté de faire partie de mon jury de thèse.

Je remercie très sincèrement Olivier Festor, mon directeur de thèse, pour m'avoir accueilli au sein de l'équipe MADYNES et accompagné tout au long de ces trois années au cours desquelles j'ai apprécié son soutien et ses nombreux conseils. Le suivi qu'il a effectué ainsi que ses capacités d'écoute et d'échange m'ont aidé à réaliser efficacement cette thèse.

Je souhaite également remercier Radu State, mon co-encadrant de thèse, qui, malgré sa mutation professionnelle, n'a jamais cessé de dialoguer avec moi. Ses nombreux conseils ainsi que sa disponibilité furent des éléments clés au bon déroulement de cette thèse.

Mes remerciements s'adressent également à l'ensemble de l'équipe MADYNES au sein de laquelle l'ambiance de travail est propice à l'échange entre tous aussi bien sur le plan personnel que professionnel. Sur ce dernier plan, je tiens spécialement à remercier Humberto Abdelnur avec lequel j'ai plus particulièrement travaillé et dont la collaboration fut agréable et efficace.

Je remercie aussi l'ensemble des personnes que j'ai pu côtoyer au LORIA et dont les discussions variées m'ont permis d'approfondir mes connaissances professionelles ou non dans de nombreux domaines. Plus précisément, mes remerciements s'adresse à Yann Guermeur, chargé de recherche du CNRS, pour la patience qu'il m'a accordé et son aide dans le domaine de la classification.

Plus personnellement, je tiens à remercier ma famille et en tout premier lieu mes parents qui m'ont accompagné tout au long de mes études. Je n'oublie pas mon épouse, Stéphanie, pour l'aide et la patience qu'elle m'a accordée.

A toutes et à tous, merci.

À ma femme.
Mes parents.
Ma famille.

Table des matières

Introduction générale **13**

1 Contexte et problématiques 13

2 Contributions 15

3 Organisation 16

Partie I État de l'art **19**

Chapitre 1 Malwares et botnets **21**

1.1 Introduction 21

1.2 Théorie de la virologie informatique 22

1.2.1 Machine de Turing 22

1.2.2 Formalisme de Fred Cohen 23

1.2.3 Formalisme de Leonard Adleman 24

1.3 Taxonomie des malwares 25

1.3.1 Bombe logique 26

1.3.2 Cheval de Troie 26

1.3.3 Virus 27

1.3.4 Vers 27

1.4 Botnets 28

1.4.1 Architecture générale 28

1.4.2 Menaces et impacts 29

1.4.3 Botnets IRC 31

1.4.4 Vers de nouveaux botnets 32
1.5 Détection et mesure des botnets 33
1.5.1 Observer le botnet 34
1.5.2 Détection et mesures passives 35
1.5.3 Détection et mesures actives 37
1.6 Modèles 38
1.6.1 Propagation d'un vers ou d'un botnet 38
1.6.2 Structure d'un botnet 40
1.7 Bilan 42

Chapitre 2 Rétro-ingénierie de protocoles **43**
2.1 Introduction 43
2.2 Généralités 44
2.2.1 Les protocoles réseaux 44
2.2.2 Définition 47
2.2.3 Applications 49
2.3 Découverte des messages : types et syntaxe 50
2.3.1 À partir de traces réseaux 50
2.3.2 En observant la machine hôte 52
2.4 Apprentissage de la machine à états 55
2.5 Le rejeu 56
2.5.1 Cas de figure simple 56
2.5.2 Problème 57
2.5.3 Solution 58
2.5.4 Autres applications 59
2.5.5 Définition formelle 59
2.6 Bilan 60

Chapitre 3 Fingerprinting **63**
3.1 Introduction 63
3.2 Méthodes de fingerprinting 64
3.2.1 Fingerprinting passif 64
3.2.2 Fingerprinting actif 65
3.2.3 Fingerprinting semi-passif 66
3.3 Applications 66
3.3.1 Systèmes d'exploitation 67
3.3.2 Protocole 69

3.3.3 Équipement, pile protocolaire . . . 75
3.3.4 Comptage . . . 77
3.4 Bilan . . . 78

Partie II Contribution **81**

Chapitre 4 Modélisation des botnets **83**
4.1 Introduction . . . 83
4.2 Cas d'étude . . . 84
4.2.1 Supervision des réseaux et services . . . 84
4.2.2 Supervision d'un pot de miel distribué . . . 85
4.2.3 Techniques existantes . . . 86
4.3 Terminologie . . . 86
4.3.1 Notations . . . 86
4.3.2 Métriques . . . 87
4.4 Modèles Internet Relay Chat . . . 88
4.4.1 Topologie Internet Relay Chat . . . 88
4.4.2 Premier modèle . . . 89
4.4.3 Second modèle . . . 92
4.4.4 Délais . . . 97
4.5 Modèles P2P . . . 97
4.5.1 Totalement maillé, Slapper . . . 97
4.5.2 Table de hachage distribuée, Chord . . . 99
4.5.3 Délais . . . 104
4.6 Bilan . . . 104

Chapitre 5 Nouvelles représentations et méthodes pour la rétro-ingénierie **107**
5.1 Introduction . . . 107
5.2 Mini protocole . . . 109
5.3 Distances entre deux messages . . . 110
5.3.1 Longueur d'un message . . . 110
5.3.2 Distribution des caractères . . . 111

5.3.3 Positions des caractères 113
5.3.4 Positions pondérées des caractères 114
5.3.5 Distribution ordonnée des caractères 115
5.3.6 Alignement de séquences 117
5.4 Partitionnement des données 118
5.4.1 Définition du problème 118
5.4.2 Méthodes utilisées 119
5.4.3 Méthode globale 120
5.5 Architecture générale 120
5.5.1 Fonctionnement 121
5.6 Bilan 123

Chapitre 6 Nouvelles techniques de fingerprinting **125**
6.1 Introduction 125
6.2 Définitions des problèmes 126
6.2.1 Terminologie 126
6.2.2 Fingerprinting supervisé 126
6.2.3 Fingerprinting non supervisé 127
6.3 Machines à vecteur de support multi-classes 127
6.4 Fingerprinting comportemental et temporel 128
6.4.1 Modélisation formelle 129
6.4.2 Générations des signatures 130
6.4.3 Mise en œuvre 132
6.5 Fingerprinting syntaxique 134
6.5.1 Représentation 134
6.5.2 Distances 136
6.5.3 Technique supervisée 140
6.5.4 Technique non supervisée 141
6.6 Conclusion 146

Partie III Evaluation **147**

Chapitre 7 Performances des botnets **149**
7.1 Introduction . . . 149
7.2 Réseaux IRC . . . 150
7.2.1 Premier modèle . . . 151
7.2.2 Paramètres . . . 151
7.2.3 Atteignabilité moyenne . . . 152
7.2.4 Nombre de sauts . . . 153
7.2.5 Nombre de nœuds . . . 155
7.3 Paramètres des modèles pair-à-pair . . . 156
7.4 Slapper . . . 156
7.4.1 Nombre de sauts . . . 156
7.4.2 Nombre de nœuds . . . 157
7.5 Chord . . . 158
7.5.1 Nombre de sauts . . . 159
7.5.2 Distance entre deux identifiants d . . . 160
7.5.3 Nombre de nœuds . . . 161
7.5.4 Atteignabilité moyenne . . . 161
7.6 Comparaison . . . 162
7.6.1 Facteur de branchement . . . 162
7.6.2 Impact des attaques . . . 162
7.6.3 Délais . . . 163
7.7 Conclusion . . . 164

Chapitre 8 Découverte automatique des types de messages **167**
8.1 Introduction . . . 167
8.2 Protocoles testés . . . 168
8.2.1 SIP . . . 168
8.2.2 Courriels . . . 170
8.3 Évaluation de la classification . . . 171
8.3.1 Précision . . . 171
8.3.2 Nombre de types . . . 171
8.3.3 Sensibilité . . . 171
8.4 Expérimentations sur SIP . . . 172
8.4.1 Partitionnement hiérarchique . . . 172
8.4.2 Support Vector Clustering . . . 174
8.4.3 Méthode globale . . . 175
8.5 Autres protocoles . . . 176

8.5.1 Partitionnement hiérarchique 176
8.5.2 Support Vector Clustering 177
8.6 Paramétrage automatique de SVC 177
8.7 Conclusion 179

Chapitre 9 Identification des équipements **181**
9.1 Introduction 181
9.2 Métriques d'évaluation 182
9.3 Fingerprinting comportemental 183
9.3.1 Jeu de données 183
9.3.2 Paramétrage 184
9.3.3 Réseau opérateur 188
9.4 Fingerprinting syntaxique 188
9.4.1 Jeux de données 190
9.4.2 Fingerprinting supervisé 191
9.4.3 Fingerprinting non supervisé 192
9.4.4 Opérateur 194
9.5 Conclusion 195

Conclusion générale **197**
1 Résumé des contributions 197
2 Perspectives 200

Glossaire **203**

Publications relatives **205**

Bibliographie **207**
1 Malwares, attaques, défenses 207
2 Modélisation des botnets 212
3 Rétro-ingénierie 213
4 Fingerprinting 215
5 Classification 217
6 Supervision des réseaux et services 219
7 RFC 220
8 Divers 221

Annexe A Partitionnement non supervisé de données **223**
A.1 Rappel des notations 223
A.2 Clustering hiérarchique ascendant 223
A.3 Support vector Clustering 225
A.3.1 Apprentissage 225
A.3.2 Découverte des clusters 228

Annexe B Machines à vecteurs de support multi-classes **229**
B.1 Apprentissage 229
B.2 Utilisation 231

Table des figures

1.1 Machine de Turing . . . 23
1.2 Taxonomie générale des malwares, les vers et virus contiennent en général un programme annexe (payload) qui est lui aussi un malware . . . 26
1.3 Nombre de machines infectées par CodeRed le 19/07/2001 . . . 27
1.4 Architecture générale d'un botnet . . . 29
1.5 Les deux principaux types d'architecture des botnets . . . 32
1.6 Nombres de machines infectées par Storm découvertes selon la méthode classique de crawling ou PPM [15] . . . 38
1.7 Modèle infectieux d'un vers . . . 39
1.8 Comparaison entre le modèle de [73] et le nombre réel de machines infectées par CodeRed 39
1.9 Effet des défenses (aléatoire ou ciblée) dans un réseau P2P non structuré . . . 41
1.10 Distribution des bots du vers Nugache selon leur degré de connectivité . . . 41
1.11 Effets similaires des défenses (aléatoires ou ciblées) dans Overnet . . . 42

2.1 Les différents modèles en couches . . . 45
2.2 Un exemple de session du protocole SIP . . . 45
2.3 Une sous-partie de la machine à état de SIP . . . 45
2.4 Extrait de la syntaxe de SIP . . . 46
2.5 Classification des différents types de champs d'un message . . . 47
2.6 La rétro-ingénierie de protocoles illustrée par un exemple sur un *ping* ICMP [12] . . . 48
2.7 Analyse par tainting . . . 52
2.8 Déduction du format général d'un message . . . 54
2.9 Construction de l'arbre APTA . . . 56
2.10 Attaque par rejeu simple . . . 57
2.11 Protection de l'attaque par rejeu par l'utilisation d'un cookie . . . 58

3.1 Les deux approches traditionelles de fingerprinting . . . 65
3.2 Différents types de fingerprinting à différents niveaux de la pile TCP/IP (Beaucoup de méthodes se basent également sur les délais entre les messages qui dépendent en partie de la couche *Accès Réseau*) . . . 66
3.3 Fenêtres TCP de l'émetteur . . . 68
3.4 Exemples de caractéristiques utilisées dans BLINC . . . 72
3.5 Graphe de sous-chaînes possibles pour le protocole SSH . . . 74
3.6 Différence d'ordonnancement des champs *Date* et *Server* dans les entêtes HTTP . . . 75
3.7 Tests des piles protocolaires sans fil 802.11 grâce aux bits de contrôle . . . 76

4.1 Supervision des réseaux à large échelle . . . 85
4.2 Cas de figure du honeypot distribué . . . 85
4.3 Notations générales utilisées pour la modélisation des botnets . . . 87
4.4 Contraintes sur les nœuds du réseau . . . 88
4.5 Calcul des probabilités de distance exacte dans un arbre aléatoire récursif . . . 89
4.6 Calcul des probabilités de distances minimales dans un arbre aléatoire . . . 91
4.7 Second modèle pour les botnets IRC avec des nœuds non numérotés, propagation d'une requête . . . 93
4.8 Calcul du nombre de voisins au second saut pour le second modèle de botnets IRC . . . 94
4.9 Limitation du nombre maximal de serveurs atteignables . . . 96
4.10 Mécanisme de broadcast dans les réseaux P2P à partir du nœud 0 . . . 100
4.11 Fonctionnement général de Chord . . . 101
4.12 Délai de diffusion d'une requête broadcast dans les réseaux P2P ($m = 3$) . . . 104

5.1 Attaque ciblées à différents moments de la communication . . . 108
5.2 Illustration du mini protocole utilisé . . . 109
5.3 Distribution des caractères entre tous les messages — SP est le caractère d'espacement . . . 111
5.4 Comparaison des distributions des caractères de deux messages particuliers — SP est le caractère d'espacement . . . 112
5.5 Distribution des caractères entre les messages chiffrés $m_0 \oplus' a'$ et $m_1 \oplus' f'$ — la valeur hexadécimale des caractères est utilisée . . . 116
5.6 Distribution ordonnée des caractères entre les messages chiffrés $m_0 \oplus' a'$, $m_1 \oplus' f'$ et $m_4 \oplus' k'$ 117
5.7 Les étapes de SVC (Les étoiles symbolisent les vecteurs supports) . . . 119
5.8 Méthode globale . . . 120
5.9 Architecture générale . . . 121

6.1 Classification multi-classes avec les SVM . . . 128
6.2 Illustration d'un arbre TR-FSM . . . 129
6.3 Exemple de génération d'une signature pour un équipement de type Asterisk (SIP) . . . 132
6.4 Signature de deux équipements différents. Les attributs des arcs sont les délais moyens. Les chemins partagés sont colorés . . . 133
6.5 Fingerprinting comportemental supervisé . . . 134
6.6 Élement de définition d'une grammaire . . . 135
6.7 Construction des arbres syntaxiques à partir d'une grammaire . . . 136
6.8 Bijections de sous-arbres syntaxiques . . . 138
6.9 Biais introduit par les structures trop imposantes . . . 139
6.10 Calcul de la similarité en déterminant l'intersection des chemins des différents arbres . . . 140
6.11 Fingerprinting syntaxique supervisé . . . 142
6.12 Classification avec ROCK . . . 143
6.13 Graphe des points partageant des voisins avec ROCK . . . 143
6.14 Fingerprinting syntaxique non supervisé . . . 146

7.1 Premier modèle IRC — Atteignabilité selon différentes distances et 10 serveurs dans le réseau . . . 151
7.2 Modèle IRC — Atteignabilité selon différents facteurs de branchement . . . 153
7.3 Modèle IRC — Nœuds atteints avec $\alpha(m) = \alpha_2(m)$ . . . 154
7.4 Modèle IRC — Dépendance entre atteignabilité et facteur de branchement . . . 155
7.5 Modèle IRC — Atteignabilité selon N et le nombre de sauts j . . . 156

7.6 Slapper — Atteignabilité . . . 157
7.7 Slapper — Influence du nombre de sauts k sur l'atteignabilité . . . 158
7.8 Slapper — Influence du nombre de sauts i sur l'atteignabilité absolue . . . 158
7.9 Chord — Atteignabilité selon le nombre total de nœuds (S = Slapper) . . . 159
7.10 Chord — Atteignabilité (S = Slapper) . . . 160
7.11 Chord – Atteignabilité moyenne . . . 161
7.12 Effet des attaques - $r(n)$ . . . 163
7.13 Comparaison des modèles avec $N = 5000$ . . . 163

8.1 Généralités sur SIP . . . 169
8.2 Distribution empirique des message SIP (le préfixe 0 indique une requête tandis que 1 préfixe les réponses) . . . 170
8.3 Distributions empiriques des autres protocoles . . . 170
8.4 Clustering hiérarchique – Distribution ordonnée des caractères . . . 172
8.5 Clustering hiérarchique – Distribution des caractères . . . 173
8.6 Clustering hiérarchique – Distribution des positions pondérées des caractères . . . 174
8.7 SVC – Distribution ordonnée des caractères . . . 175
8.8 SVC – Distribution des positions pondérées des caractères . . . 176
8.9 Partitionnement hiérarchique – autres protocoles (τ en abscisse) . . . 177
8.10 SVC - SMTP dataset . . . 178
8.11 SVC – Paramétrage automatique de q . . . 179
8.12 tuningSIP . . . 179

9.1 Fingerprinting comportemental – Caractéristiques des jeux de données (échelle logarithmique, un point par type, valeur médiane représentée par une barre horizontale) . . . 185
9.2 Paramétrage du fingerprinting comportemental (taille de test = 10, taille d'apprentissage = 5) . . . 187
9.3 Fingerprinting syntaxique supervisé, distance d_1 . . . 190
9.4 Fingerprinting syntaxique supervisé, distance d_3 . . . 191
9.5 Fingerprinting syntaxique non supervisé amélioré . . . 194

A.1 Partitionnement hiérarchique – les clusters en rouge sont les clusters finaux si une distance maximal représentée par la ligne rouge est spécifiée. . . . 224
A.2 Clustering hiérarchique selon deux types de distances inter-clusters . . . 226
A.3 Les étapes de SVC (Les étoiles symbolisent les vecteurs supports) . . . 226
A.4 Forme aléatoire d'un cluster, les lignes en pointillés montrent que pour tout point du cluster, il existe un segment linéaire le reliant à un autre sans sortir de la forme du cluster 228

Introduction générale

1 Contexte et problématiques

Essor des applications communicantes

L'essor d'Internet se traduit par l'augmentation d'équipements connectés aussi bien dans les entreprises que chez les particuliers, tous à la recherche d'une connexion quasi permanente (équipements mobiles) et d'une connexion toujours plus performante, en termes de bande passante et de délais, obligeant les fournisseurs d'accès à s'orienter vers la mise en place de réseaux hauts voire très hauts débits (ADSL, FTTH...). Alors qu'auparavant, Internet se limitait dans la majorité des cas à la consultation d'une page Web ou à l'envoi de courriel, de nombreuses autres applications sont apparues depuis telles que les différentes messageries instantanées, le partage et la diffusion de fichiers en pair-à-pair ou via des serveurs centraux, les jeux en ligne, l'écoute ou le visionnage en direct de fichiers multimédias, la téléphonie, la vidéo-conférence, des protocoles tels que RTMP (protocole mis en œuvre dans les applications Flash[11]) permettant de développer des sites Webs plus interactifs et plus riches... En outre, il existe un certain nombre d'applications plutôt destinées aux entreprises et dont le nombre n'a cessé d'augmenter à l'instar des imprimantes en réseaux, des serveurs d'annuaire, des protocoles d'accès distants tels que SSH (Secure Shell), des serveurs de travail collaboratif ou de bases de données... À cela se rajoutent encore tous les protocoles de plus bas niveaux qui encapsulent ces protocoles car ces derniers sont uniquement de haut niveau (couche application). Certains d'entre eux sont inévitables pour véhiculer les informations d'une machine à une autre grâce aux protocoles de routage couplés avec des serveurs de noms (DNS) pour localiser les machines.

Il est clair que dresser une liste exhaustive des protocoles réseaux est impossible car chacun peut concevoir son propre protocole. Ainsi, les flux réseaux circulant sur Internet ou les réseaux locaux sont variés de la même manière que les applications communicantes associées. Rapidement, les administrateurs furent confrontés à cette diversité nécessitant des solutions de supervision adaptées pour déterminer rapidement l'état d'un service ou du réseau en un point afin d'en modifier la configuration si nécessaire. Par conséquent, la conception d'autres protocoles spécifiques à ces tâches a été nécessaire.

Par ailleurs, pour un protocole donné, il peut exister un grand nombre d'applications comme, par exemple, les serveurs ou clients Webs ou de courriels. Cette diversité s'est accentuée ces dernières années par les efforts des entreprises rendant la spécification de leurs protocoles partiellement ou entièrement publiques ou en fournissant des interfaces de programmation pour les utiliser.

Brièvement, la multiplicité des protocoles et des applications associées n'est plus à démontrer aujourd'hui.

Défis majeurs

Conception efficace d'une application communicante

Face à la multitude des protocoles existants, deux solutions sont envisageables lors de la réalisation d'un nouveau logiciel :

– la sélection d'un protocole existant ;
– la conception d'un protocole dédié.

La seconde approche a l'avantage de créer un protocole sur mesure à une application donnée et de le protéger de la copie évitant ainsi l'apparition d'applications clones concurrentes. Toutefois, elle engendre des coûts de conception élevés (étude, développement, tests...).

Cependant, quelque soit le choix effectué, une question subsiste dans tous les cas et concerne la robustesse de l'application communicante. Généralement, les utilisateurs souhaitent une application disponible et rapide malgré la présence d'un grand nombre d'utilisateurs en parallèle. Ainsi, la robustesse se définit dans cette thèse comme la capacité pour une application communicante à passer à l'échelle, c'est-à-dire de gérer un grand nombre d'utilisateurs à la fois tout en étant confrontée à différents problèmes comme les pannes réseaux et les attaques informatiques.

Enfin, il arrive parfois que l'adoption d'un protocole soit imposée, notamment dans le cadre de la conception de logiciels interopérables. Toutefois, les spécifications ne sont pas toujours disponibles ou que partiellement. Dans un premier temps, le fonctionnement du protocole doit être étudié en observant les messages échangés et/ou la manière dont se comporte l'application.

Sécurité

En parallèle à l'essor d'Internet, le nombre d'attaques n'a cessé d'augmenter et les actions malveillantes qui, à l'origine, étaient plutôt des démonstrations d'informaticiens en quête de reconnaissance, sont passées aujourd'hui à un stade beaucoup moins « attrayant ». En effet, beaucoup d'entre elles visent directement à engendrer des revenus financiers (spam, attaque d'une entreprise concurrente, fraude à la carte bancaire...) voire à influer sur des thèmes politiques (propagande, censure, espionnage...). Bien évidemment, la sécurité n'a donc cessé d'être un problème préoccupant pour les particuliers mais encore plus fortement pour les entreprises. Ainsi, la mise en place de pare-feux ou de systèmes de détection d'intrusion est obligatoire pour repérer et stopper toute action malveillante le plus rapidement possible.

Économie

Alors qu'aujourd'hui l'accès à Internet devient de plus en plus aisé et moins coûteux aussi bien en terme d'abonnement que d'équipement comme le montre l'apparition des ordinateurs à bas prix, les entreprises doivent s'efforcer de fournir des applications toujours plus innovatrices. Néanmoins, le développement rapide d'applications concurrentes ou de logiciels libres exerce une pression accrue sur cette activité. Aujourd'hui, un produit innovant est un produit qui s'adaptera au mieux au client. Ainsi, il est nécessaire de fournir des services personnalisés pour séduire les clients.

Supervision

Comme mentionné précédemment, la supervision des réseaux et des services fut logiquement engendrée par leur croissance. Cependant, celle-ci est aujourd'hui confrontée à plusieurs problèmes. Tout d'abord, le nombre d'équipements à superviser ne cesse de croître : équipements réseaux, serveurs, imprimantes, ordinateurs (fixes ou portables), téléphones mobiles... Deuxièmement, le nombre d'applications

utilisées est de plus en plus important. En clair, un challenge majeur est de gérer un grand nombre de services variés sur toujours plus d'équipements de différentes natures.

Problématiques

De nos jours, la robustesse d'une application communicante est souvent étudiée empiriquement. Ainsi, le passage à l'échelle ou non est généralement observé lors de l'utilisation du logiciel. Une avancée serait dont de pouvoir prédire les performances atteignables à l'avance. Il deviendrait alors possible de choisir la meilleure solution technique à un problème posé selon les besoins (nombre d'utilisateurs) et les coûts (architecture à utiliser, développement, maintenance...).

La compréhension d'un protocole inconnu, c'est-à-dire dont les spécifications ne sont pas disponibles, s'apparente à la rétro-ingénierie des protocoles dont la difficulté et la complexité demeurent aujourd'hui un problème important. En effet, beaucoup des méthodes existantes reposent sur des techniques assez complexes nécessitant parfois l'instrumentation d'une machine exécutant une application utilisant le protocole, ce qui n'est pas toujours possible (risque de sécurité, coûts...). Ainsi, retrouver automatiquement le fonctionnement d'un protocole à partir des traces réseaux reste un challenge à l'heure actuelle. De plus, la compréhension d'un protocole s'avère essentielle dans le domaine de la sécurité car une intrusion ou une attaque ne prendra pas la même forme selon le protocole sur lequel elle se base.

Les problèmes de sécurité sont associés à des failles de sécurité qui ne sont pas les mêmes selon le protocole et l'application employés. Ainsi, lors de la découverte d'une faille, identifier le plus rapidement possible les machines concernées est un atout essentiel pour réduire l'impact des attaques envisageables. Cependant, maintenir un inventaire de tous les équipements et logiciels installés reste difficile du fait de leur variété et de leur dynamique (installation, désinstallation) d'où l'intérêt des techniques de fingerprinting. Face à la variété des applications, un problème majeur actuel est d'avoir un système de fingerprinting rapide et générique, c'est-à-dire pouvant être rapidement utilisé sur n'importe quel protocole avec une intervention humaine très limitée.

Par ailleurs, le fingerprinting peut aider à identifier des applications ou des équipements dont la présence n'est pas normale et pouvant alors être synonyme d'intrusion.

De plus, le fingerprinting est un support essentiel à la vente de services personnalisés aux utilisateurs dans le sens où il aide à identifier quels sont les logiciels ou les équipements employés par un utilisateur facilitant ainsi le ciblage des offres qui peuvent lui être faites.

Enfin, dans le domaine de la supervision, le passage à l'échelle demeure problématique bien que ne constituant pas le point essentiel de cette thèse.

2 Contributions

Cette thèse s'articule autour de trois contributions principales dans deux thèmes différents. Tout d'abord, l'étude de la robustesse des applications communicantes est illustrée par l'analyse des logiciels de type botnet. Différents modèles mathématiques ou algorithmiques sont proposés pour étudier les performances correspondantes. Ils tiennent compte également des risques de déconnexion ou d'attaques qui peuvent surgir. Malgré le rôle incontesté de ce type de réseau dans des actions malveillantes, cette thèse montrera que de tels réseaux peuvent également servir à élucider le problème du passage à l'échelle pour la supervision. Par ailleurs, l'élaboration des modèles dans différents cas (pseudo-centralisé ou P2P) souligne plusieurs méthodes de modélisation possibles adaptables à d'autre cas. Enfin, cette étude aide aussi à mieux comprendre les botnets et donc à mieux s'en prémunir.

Le deuxième thème de cette thèse est l'identification des applications. Ce terme relève en premier lieu de la rétro-ingénierie du protocole utilisé par cette dernière. Une nouvelle méthode se basant uniquement sur les traces réseaux et avec une complexité très réduite est introduite afin de découvrir le type général de chaque message. Alors qu'un tel objectif peut paraître très limité, le type des messages est la brique de base essentielle à la reconstruction de la machine à états du protocole permettant sa bonne compréhension générale. De plus, elle est directement utilisable avec la dernière contribution de cette thèse visant à identifier les équipements ou les applications utilisées. En effet, la première méthode repose uniquement sur les types de messages échangés entre plusieurs entités pour les identifier. Ce type de fingerprinting sera qualifié de comportemental par la suite. Pour ce faire, une nouvelle représentation formelle des équipements ou des applications est proposée ainsi qu'une fonction noyau associée utilisable avec les techniques récentes de classification que sont les machines à vecteurs supports. Sa validité est également formellement prouvée. Une autre technique complémentaire s'appuie quant à elle sur la reconstruction des arbres syntaxiques de chaque message et nécessite donc la connaissance de la grammaire du protocole. La nouveauté de cette approche réside principalement dans la prise en compte de la structure même d'un message et plus seulement son contenu. L'identification des applications sera illustrée via plusieurs protocoles : SIP [6], IMAP, [4] et SMTP [1].

3 Organisation

L'organisation générale de la thèse repose sur trois grandes parties. La première dresse un bilan des recherches actuelles dans le cadre des problématiques associées à la thèse. La seconde se focalise sur la description des nouvelles méthodes proposées alors que la dernière est une évaluation de ces dernières. Ces trois grandes parties se retrouvent dans le tableau suivant et chacun d'elle est dérivée pour chacun des thèmes abordés précédemment.

Partie	Robustesse des botnets	Rétro-ingénierie	Fingerprinting d'équipement
État de l'art – Partie I	Chapitre 1	Chapitre 2	Chapitre 3
Contribution – Partie II	Chapitre 4	Chapitre 5	Chapitre 6
Évaluation – Partie III	Chapitre 7	Chapitre 8	Chapitre 9

Robustesse des botnets

Le chapitre 1 répertorie les différents travaux d'analyse des botnets. Cependant, un botnet étant une catégorie de malware, cela impose d'en dresser un aperçu également. Il termine par les différentes approches de modélisation de botnets existantes en soulignant leurs limites contournées par les modèles proposés dans le chapitre 4. Ce chapitre introduit également l'application de ce type d'architecture à la supervision des réseaux et des services. Ainsi, l'objectif du chapitre 7 est de montrer à la fois les performances des différentes architectures de botnets sous différentes conditions tout en justifiant la mise en œuvre de celles-ci dans le cadre de la supervision.

Rétro-ingénierie

La rétro-ingénierie de protocoles est un domaine dans lequel de nombreuses méthodes existent et que le chapitre 2 s'efforce de décrire. Chaque méthode se différencie d'une autre selon les hypothèses sous-jacentes, les données analysées ou les méthodes d'analyse employées. Ainsi, un bilan est donné en soulignant les avantages et les contraintes de chacune d'entre elles. Le chapitre 5 décrit une nouvelle méthode de rétro-ingénierie reposant sur une technique relativement récente, les machines à vecteurs supports, peu employées dans ce domaine. De plus, de nouvelles représentations des messages sont proposées dont les avantages sont exposés ainsi que la manière de s'en servir. Enfin, le chapitre 8 représente l'évaluation de cette nouvelle technique.

Fingerprinting d'équipement

Le fingerprinting est une discipline non restreinte à l'identification des équipements. Le chapitre 3 détaille les différents cas de fingerprinting existant tout en mettant en avant leur intérêt. Il souligne notamment le peu de méthodes génériques existant pour le fingerprinting d'équipement. C'est pour cette raison que la chapitre 8 propose deux nouvelles méthodes applicables à tout protocole : une première reposant sur les interactions entre les machines et une seconde étudiant la structure syntaxique des messages. Une fois de plus, les machines à vecteurs supports seront mises en œuvre et complétées par un nouvel algorithme de classification. Ce dernier est d'ailleurs applicable à n'importe quel domaine et n'est pas spécifique du fingerprinting ou des réseaux. Enfin, le chapitre 9 met en exergue de très bons résultats justifiant l'utilisation de ces nouvelles techniques.

Enfin, l'ensemble des contributions de ces trois années de recherche sont récapitulées dans la conclusion et sont complétées par un descriptif des perspectives de recherche.

Première partie

État de l’art

1

Malwares et botnets

Sommaire

1.1 Introduction . . . 21
1.2 Théorie de la virologie informatique . . . 22
1.2.1 Machine de Turing . . . 22
1.2.2 Formalisme de Fred Cohen . . . 23
1.2.3 Formalisme de Leonard Adleman . . . 24
1.3 Taxonomie des malwares . . . 25
1.3.1 Bombe logique . . . 26
1.3.2 Cheval de Troie . . . 26
1.3.3 Virus . . . 27
1.3.4 Vers . . . 27
1.4 Botnets . . . 28
1.4.1 Architecture générale . . . 28
1.4.2 Menaces et impacts . . . 29
1.4.3 Botnets IRC . . . 31
1.4.4 Vers de nouveaux botnets . . . 32
1.5 Détection et mesure des botnets . . . 33
1.5.1 Observer le botnet . . . 34
1.5.2 Détection et mesures passives . . . 35
1.5.3 Détection et mesures actives . . . 37
1.6 Modèles . . . 38
1.6.1 Propagation d'un vers ou d'un botnet . . . 38
1.6.2 Structure d'un botnet . . . 40
1.7 Bilan . . . 42

1.1 Introduction

Au cours de son expansion, l'informatique a rapidement montré ses faiblesses avec les premières pannes et bogues. Alors que ces problèmes furent d'abord dûs à des erreurs de conception, des personnes malintentionnées ont vite compris qu'elles pouvaient les exploiter de manière volontaire. Ainsi, les

problèmes de sécurité informatique et les programmes malveillants ou malware sont rapidement arrivés. La dénomination malware regroupe l'ensemble des logiciels ayant des fins malveillantes. En quelques mots, on peut dire qu'un malware cherche à détourner un ordinateur de son comportement normal. Cette définition est assez vague et va être affinée au cours de ce chapitre car, pour se défendre efficacement contre eux, les comprendre est obligatoire.

Les virus, malwares connus de tous, ont été à la base de ce développement et ce chapitre commencera donc par donner les fondements théoriques de la virologie informatique. Dans un deuxième temps, une classification des malwares sera proposée car la diversité de ces derniers est très grande. Plus particulièrement, quelques détails sur les vers informatiques seront exposés et montreront leurs énormes capacités infectieuses, nécessaires au déploiement d'un réseau de machines compromises contrôlées par un pirate (attaquant). Ces réseaux, nommés botnets, seront approfondis dans ce chapitre. Ils représentent aujourd'hui une des plus grandes menaces sur Internet grâce aux attaques diverses et surtout coordonnées qu'ils peuvent réaliser.

Une fois de plus, pour mieux s'en défendre, il est nécessaire de les étudier car il est légitime de se demander quel est exactement leur potentiel d'attaque, les motivations qui se cachent derrière ou encore leur réel déploiement actuel et son évolution dans le futur. Ce chapitre va donc mettre l'accent sur les techniques et les modèles qui permettent de mieux les appréhender.

1.2 Théorie de la virologie informatique

1.2.1 Machine de Turing

Cette partie présente une formalisation des virus informatiques. Le lecteur intéressé pourra consulter [61] pour de plus amples détails sur la théorie des virus ainsi que leurs aspects pratiques.

Un virus étant un programme informatique, la formalisation de ces derniers, proposée par Alan Turing [16], est rapidement rappelée ici. Une machine de Turing est souvent illustrée comme sur la figure 1.1 :

- un ruban *tape* infini (ou bande) contenant des symboles, en l'occurrence des 0 ou des 1 ;
- une tête de lecture et écriture pointant sur une cellule de ce ruban ;
- une fonction de contrôle F qui correspond à un automate et dont les actions possibles sont le déplacement de la tête de lecture (à gauche ou à droite) et l'écriture sur la bande à l'endroit où est positionnée la tête d'écriture.

Son fonctionnement est assez simple, la tête de lecture lit un caractère sur le ruban qui sert d'entrée à la fonction F. L'automate change alors d'état selon cette valeur lue et l'état actuel et peut exécuter une opération (déplacement, écriture).

Bien que cette description et le langage utilisé (1 ou 0) soient relativement simples, ils permettent de modéliser tout programme informatique. Sa définition plus formelle est la suivante :

Définition 1 :*Une machine de Turing est une fonction M définie par :*

$$M : \{0,1,\ldots,n\} \times \{0,1\} \rightarrow \{0,1\} \times \{D,G\} \times \{0,1,\ldots,n\}$$

où $\{0,1,\ldots,n\}$ *sont les états possibles de l'automate (état courant dans la partie gauche, état après transition dans la partie droite),* $\{0,1\}$ *les symboles pouvant être dans les cellules de la bande (lecture dans la partie gauche, écriture dans la partie droite) et* $\{D,G\}$ *représente les déplacements possibles de la tête (droite ou gauche).*

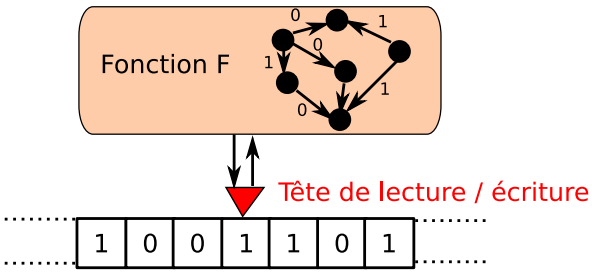


FIGURE 1.1 – Machine de Turing

Il est important de noter qu'une machine de Turing correspond à un programme donné seulement. Pour palier cette limite, une machine de Turing universelle est capable de simuler le fonctionnement de n'importe quelle machine de Turing dont la description serait stockée sur le ruban. Ainsi, une machine de Turing universelle se comporte comme un ordinateur capable d'exécuter divers programmes.

1.2.2 Formalisme de Fred Cohen

Dans les années 70, les premiers travaux sur la sécurité informatique mettent en évidence l'apparition de programmes malveillants [80]. Dans les années 80, plusieurs virus font leur apparition comme *Elk Cloner* [7] écrit par Rich Skrenta alors âgé 15 ans. Ce virus infectait les machines Apple et la seule action qu'il faisait était d'afficher un poème après 50 démarrages. Ce virus se transmettait via des disquettes sur lesquelles il se clonait.

C'est à cette époque que Fred Cohen [79] proposa une définiton d'un virus :

Définition 2 :*Un virus est décrit par une série de symboles qui, lorsqu'ils sont interprétés dans un environnement convenable, peuvent modifier d'autres symboles de cet environnement en créant une copie exacte ou modifiée d'eux-mêmes.*

Cohen propose alors une définition d'une machine de Turing équivalente à la précédente :

Définition 3 :*Une machine de Turing est définie par :*
- *un ensemble d'états* $S = \{s_0, s_1, \ldots, s_n\}$
- *un ensemble de symboles* $I = \{i_0, i_1, \ldots, i_m\}$
- *des déplacements possibles* $d = \{+1, 0, -1\}$
- *une fonction de sortie qui peut générer un symbole selon celui lu et l'état courant* $O : S \times I \rightarrow I$
- *une fonction de transition qui, selon le symbole lu, change l'automate d'état* $N : S \times I \rightarrow S$
- *une fonction de déplacement qui détermine le déplacement à faire selon l'état courant et le symbole lu :* $D : S \times I \rightarrow d$

Cohen introduit une notion temporelle aux machines de Turing. Il considère l'état courant *state* de la machine et la position *pos* de la tête de lecture/écriture sur le ruban. Au temps t, une machine de Turing M est donc caractérisée par le tuple $\Theta_t =< state_t, pos_t, tape_t >$. $tape[j]$ dénote le j^e symbole du ruban $tape$ et $write(t, s, p)$ représente le ruban t où le symbole s a été écrit à la position p. On peut alors définir le résultat de la machine de Turing à l'instant t :

Définition 4 : *Étant donnée la configuration Θ_t à l'instant t tel que $\Theta_t =< state_t, pos_t, tape_t >$, la configuration à l'instant $t+1$ est définie par :*

$$\Theta_{t+1} =< N(state_t, tape_t[pos_t]), pos_t + D(state_t, tape_t[pos_t]), write(tape_t, O(state_t, tape_t[pos_t]), pos_t >$$

De plus, l'état initial est défini par $\Theta_0 =< s_0, 0, tape_0 >$ où $tape_0$ représente le programme donné en entrée.

Ces précisions permettent alors à Cohen de spécifier l'ensemble des virus pour une machine de Turing M et les programmes V qu'elle peut interpréter. Supposons que la tête de lecture pointe sur la cellule c à l'instant t, que M soit dans son état initial et que le texte d'un virus $v \in V$ commence à la cellule j alors il existe un virus $v' \in V$ à l'instant t' et à la position c' si et seulement si :

- le texte du virus v' commence à la cellule c' ;
- $c' > c$ tel que les textes considérés ne se superposent pas ;
- $\exists t''$ tel que $t < t'' < t'$ et qu'à l'instant t'', v' est écrit par M

Cette définition formelle signifie donc que toute machine de Turing dans son état initial et ayant le début d'un virus sous sa tête de lecture entraîne la création d'un autre virus, soit une copie soit une forme modifiée.

1.2.3 Formalisme de Leonard Adleman

Il est important de remarquer que le formalisation de Cohen ne fait en aucun cas intervenir la notion de contenu et d'action du virus. Ici, un virus n'est donc pas forcément un programme nuisible mais seulement un programme auto-reproducteur. À l'inverse certains malwares ne sont pas auto-reproducteurs et ne peuvent donc pas être définis grâce à cette formalisation. C'est pour cette raison qu'Adleman [78] en proposa une autre qui spécifie trois types d'actions des malwares :

- infection : le malware est capable d'infecter d'autres programmes ;
- nuisance : tout programme infecté réalise une autre tâche que celle normalement prévue ;
- furtivité : le malware effectue la tâche normalement prévue.

La formalisation d'Adleman, présentée dans cette section, repose sur les fonctions récursives limitées aux entiers naturels qui sont équivalentes aux machines de Turing. Par ailleurs, un système informatique est modélisé par le couple (s,t) où $s =< s_1, s_2, \ldots, s_p >$ est une séquence d'entiers naturels représentant les programmes du système et t, une séquence du même genre, représentant les données (paramètres). L'exécution d'un programme correspond à une fonction $f(s,t)$ et plus particulièrement à la fonction partielle correspondante $\varphi_f(s,t)$ (c'est-à-dire en la limitant au domaine où elle est calculable). La formalisation est résumée ci-dessous par l'intermédiaire de quatre définitions majeures :

Définition 5 : *Pour tout programme p, il existe une forme infectée $I_v(p)$ par le malware v.*

Définition 6 : *Pour tout système (s,t), un malware v est nuisible si deux programmes infectés, a et b, par celui-ci exécutent une même tâche identique indépendante de celle normalement prévue :*

$$\varphi_{I_v(a)}(s,t) = \varphi_{I_v(b)}(s,t)$$

Définition 7 : *Pour tout système (s,t), un malware v est infectieux si pour un programme a, la tâche réalisée par la forme infectée est identique à celle réalisée par la forme normale, tout en ayant modifié certains programmes du système :*

Classe	Pathogène	Contagieux
Bénin		
Epien (Simple)	×	
Disséminateur		×
Malveillants	×	×

TABLE 1.1 – Les différentes classes de malwares selon Adleman

- $\varphi_{I_v(a)}(s,t) = g(s',t)$
- $\varphi_a(s,t) = g(s',t)$ *où :*
  - $\exists i, s'_i \neq s_i$.
  - $\forall i, s'_i = s_i \vee s'_i = I_v(s'_i)$

Définition 8 : *Pour tout système (s,t), un malware v est furtif si pour un programme a, la tâche réalisée par la forme infectée est identique à celle réalisée par la forme normale et si les programmes du système restent inchangés :*

- $\varphi_{I_v(a)}(s,t) = a(s,t)$

Adleman considère alors qu'un malware est contagieux s'il possède des capacités infectieuses et il est pathogène s'il est nuisible. Quatre classes de malwares sont alors identifiables et représentées dans le tableau 1.1. Il est important de noter que les programmes auto-reproducteurs de Cohen sont équivalents à la classe des malwares disséminateurs.

1.3 Taxonomie des malwares

Alors que la section précédente donnait les bases théoriques de la virologie informatique, des définitions plus pratiques sont introduites ci-dessous. Il existe plusieurs critères de classification possible des malwares. Cependant, malgré un critère précis, un malware peut se retrouver dans plusieurs classes distinctes du fait de la variété des actions qu'il peut mener. Une taxonomie des programmes infectieux, dont est dérivée la figure 1.2, est proposée dans [61].

Deux types distincts sont considérés. Les malwares qui ont infecté d'autres fichiers tels que précédemment introduits :

- ceux, dits simples, qui ne se propagent pas (voir formalisme d'Adleman),
- ceux auto-reproducteurs (voir formalisme de Cohen)

À l'inverse, il existe des malwares dits non infectieux qui sont en fait les programmes contenus dans les précédents (payload) et les deux ainsi associés forment alors un malware dangereux. Ce type de malware correspond alors à la classe *malveillante* d'Adleman. C'est véritablement le programme annexe qui a la charge de l'action malveillante. À titre d'exemple, on peut citer des publiciels ou adwares (logiciels affichant de la publicité), des logiciels espions (enregistreur de frappe, récupération des mots de passe), des rootkits permettant de dissimuler les autres malwares et actions malveillantes ou encore des portes dérobées (backdoor en anglais)... Ce dernier type est aussi très dangereux car l'attaquant peut réaliser de nombreuses actions malveillantes : récupération ou destruction de données de la victime, récupération des mots de passe, attaques vers d'autres machines, passerelles pour anonymiser sa connexion... Les programmes infectieux sont donc associés la plupart du temps à un autre programme de nature diverse mais le but ici n'est pas de détailler les diverses possibilités. C'est pourquoi seuls les types infectieux sont détaillés ci-après.

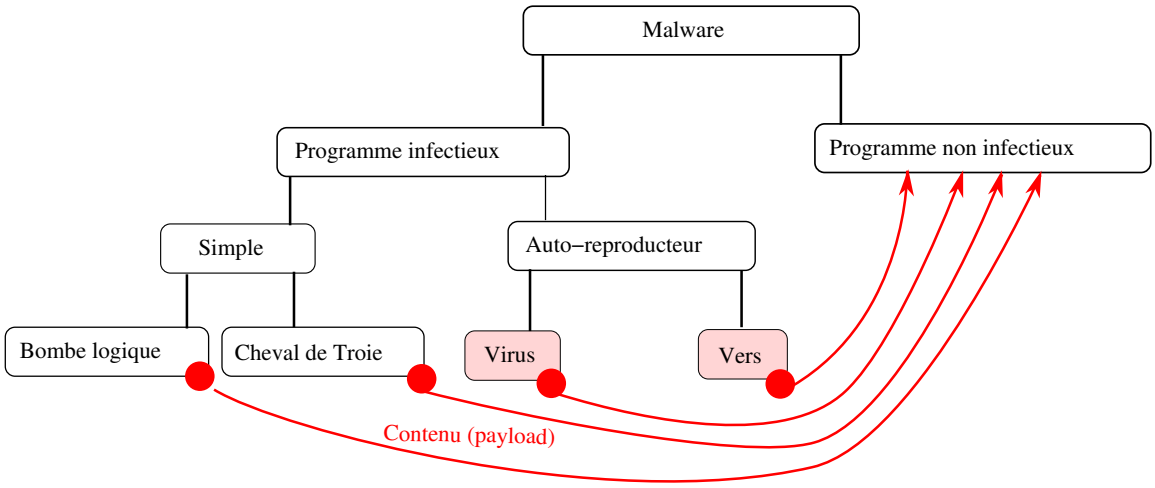


FIGURE 1.2 – Taxonomie générale des malwares, les vers et virus contiennent en général un programme annexe (payload) qui est lui aussi un malware

1.3.1 Bombe logique

Une bombe logique est un malware dont l'action malveillante est conditionnée par la satisfaction de certaines conditions : date, actions de l'utilisateur de la machine... L'exemple le plus connu d'un tel malware est le virus *Tchernobyl* ou CIH. Celui-ci a sévi à la fin des années 90 et au début des années 2000 et était très virulent. Il est considéré comme une bombe logique car il s'activait uniquement le jour de l'anniversaire de la catastrophe nucléaire de *Tchernobyl*.

Une bombe logique ne se propage pas d'elle même et est la plupart du temps associée à un vers ou un virus pour pouvoir se répandre à grande échelle.

1.3.2 Cheval de Troie

Les chevaux de Troie sont des logiciels en apparence bénin mais dont certaines fonctionnalités sont cachées à l'utilisateur. Ainsi, en installant un logiciel tout à fait banal, celui-ci va effectuer des actions non conformes à ses spécifications, comme par exemple supprimer des fichiers ou redémarrer la machine. Les chevaux de Troie sont en général associés à des portes dérobées qui permettent à un attaquant de pouvoir prendre le contrôle de la machine à distance. La nature du logiciel peut être très différente et se limiter simplement à ne rien faire. Par exemple, un programme qui serait téléchargé et installé d'après sa description et qui ne ferait rien hormis des actions malveillantes serait aussi considéré comme un cheval de Troie.

Ils ont été popularisés par *Back Orifice* [13] qui fut intégré et caché dans de nombreux malwares et qui permet de contrôler une machine Windows à distance. Ce programme n'est pas un malware lorsqu'il est installé légitimement et les auteurs ont défendu l'intérêt légal de ce programme notamment pour les administrateurs réseaux. Plus récemment, Silentbanker [2] visait à faire de la fraude bancaire.

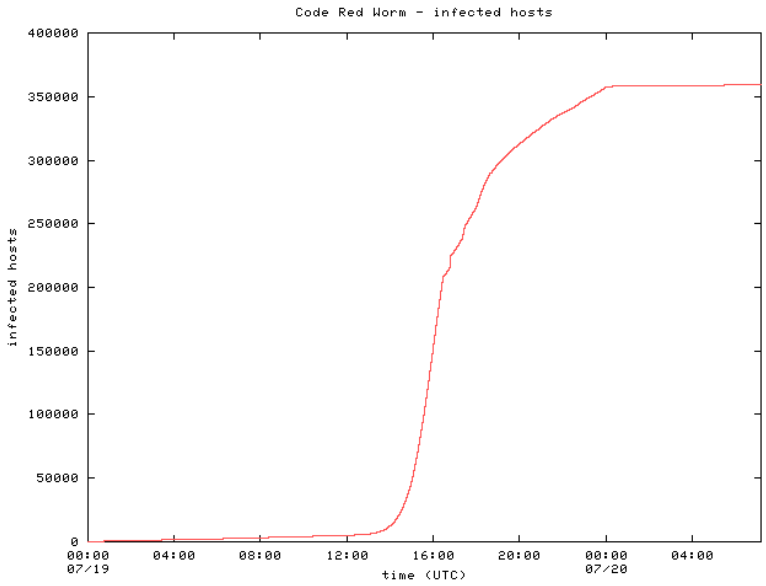


FIGURE 1.3 – Nombre de machines infectées par CodeRed le 19/07/2001

1.3.3 Virus

Un virus est un programme auto-reproducteur qui cherche à se cloner dans différents endroits d'un système informatique (fichiers, mémoire...) (voir section 1.2). La méthode la plus simple consiste à écraser un fichier en partie ou en entier. De ce fait, l'exécution même du programme ne pourra pas fonctionner convenablement et le virus sera facilement repéré. D'autres méthodes consistent à insérer le virus dans le fichier sans en altérer son comportement. La taille de ce dernier se retrouve donc augmentée mais son comportement reste normal aux yeux de l'utilisateur.

1.3.4 Vers

Un vers est un malware infectieux qui se transmet via les réseaux de communication. Aujourd'hui, nombreux sont les vers qui ne se limitent pas à ce moyen de propagation et qui en incluent d'autres habituellement utilisés par les virus tel qu'Conficker [20] qui se transmet aussi via les supports USB. Un bref aperçu des vers est donné ici et illustré par quelques exemples. Historiquement, le vers Morris (1989) du nom de son auteur est considéré comme le premier [77].

Les vers se propagent généralement grâce à des failles présentes sur les systèmes informatiques à l'instar de CodeRed [73]. Ce ver, apparu en 2001, exploite une faille du serveur Web IIS de Microsoft [10]. Les études réalisées dans [10] montrent son efficacité et surtout sa vélocité d'infection comme illustré sur la figure 1.3 où en une journée, le ver a contaminé 350.000 machines soit le maximum observable avec la méthode utilisée. Ce ver se propageait de manière aléatoire car chaque instance balayait 100 adresses IPs à la recherche d'un serveur IIS. La même méthode fut appliquée par Slammer mais de manière encore plus impressionnante car 90% des machines vulnérables furent infectées en 10 minutes [70]. Les auteurs de [64, 74] proposent des modèles plus évolués en attribuant à chaque hôte infecté une liste d'adresses à tester précise et différente, au moins en partie, de celle d'une autre machine infectée.

Les adresses à balayer sont ainsi distribuées tout au long de la propagation du vers. En outre, certains vers peuvent aussi établir des listes selon ce qu'ils recherchent (serveurs de mail, Web...) [74].

Cependant la recherche exhaustive de machines à infecter n'est pas très discrète et est donc facilement détectable. De nouvelles techniques de propagation plus ciblées ont alors vu le jour. Par exemple, ILoveYou [75] se propageait en envoyant des courriels vers les adresses des carnets d'adresses des machines infectées. Aujourd'hui, grâce au nombre de messageries différentes qui existent, les virus peuvent passer ou collecter des informations via de nombreux réseaux [57] qui ont l'avantage de fournir une liste de victimes potentielles et leurs états (connectées ou non).

Bien qu'ici seuls les aspects de propagation soient traités, les vers comme les virus sont généralement associés à un autre programme ou module dont les actions possibles sont diverses. A titre d'exemple, les ordinateurs infectés par CodeRed lançaient des attaques de déni de service [74]. De plus, un vers peut combiner plusieurs moyens de propagation [74].

1.4 Botnets

Une fois qu'une machine est infectée par un malware, il peut installer un programme "bot". L'ensemble des machines réalisant cette tâche se retrouvent alors regroupées dans un grand réseau : un botnet. De manière plus formelle, il est possible de définir les notions suivantes :

Définition 9 : *Un programme bot est un logiciel permettant d'accéder et de contrôler à distance la machine où il est installé. Une telle machine se nomme alors un bot ou un robot.*

Définition 10 : *Un botmaster ou contrôleur est la personne qui contrôle un bot.*

Définition 11 : *Un botnet est un réseau de bots contrôlés par le même botmaster.*

Cette section vise à détailler l'architecture générique d'un botnet, ses fonctionnalités ainsi que les implémentations réelles.

1.4.1 Architecture générale

La figure 1.4 représente les acteurs et le fonctionnement d'un botnet. L'attaquant, le contrôleur ou botmaster, contrôle un grand nombre de machines qui sont ainsi qualifiées de zombies ou robots (bot). Celles-ci peuvent être de diverses natures car tout équipement sur Internet capable de communiquer peut se retrouver compromis pour être transformé en zombie : ordinateurs, téléphones, serveurs, routeurs, imprimantes, modems... Cependant, la plupart d'entre eux sont des ordinateurs basiques de particuliers qui sont utilisés car ils constituent un réservoir énorme de machines peu ou faiblement défendues comparées aux équipements professionnels. Néanmoins, les ordinateurs d'entreprise sont aussi affectés par le phénomène notamment lors du déploiement d'un vers comme Conficker [20] en 2008. Cependant, les entreprises se doivent d'être plus réactives en mettant rapidement en œuvre des moyens de protection et de décontamination. Le canal de communication utilisé pour envoyer les ordres aux bots se nomme le canal de commande et contrôle (C&C).

Souvent, le déploiement du botnet est réalisé via la propagation d'un vers. Par exemple, pour Storm [31], on parlera du vers ou du botnet Storm.

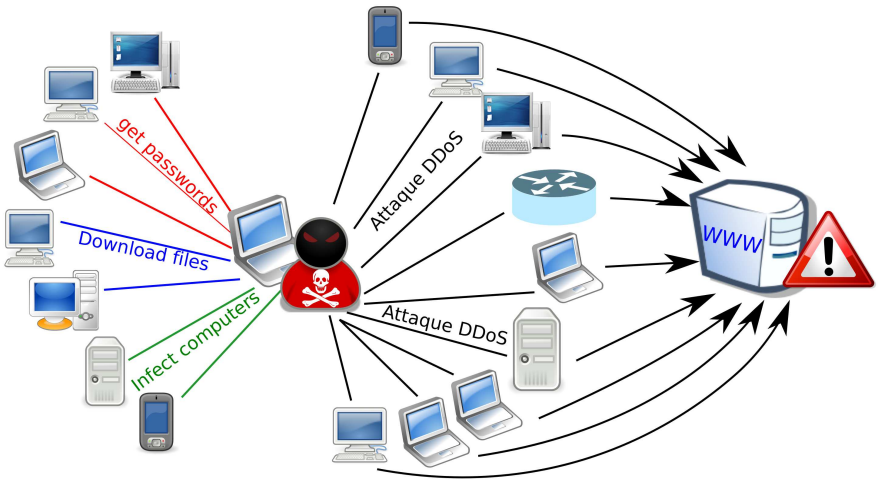


FIGURE 1.4 – Architecture générale d'un botnet

1.4.2 Menaces et impacts

La figure 1.4 met en lumière quelques actions possibles du botmaster parmis un panel indénombrable. Historiquement, Eggdrop [8], le premier robot IRC que l'on peut considérer comme point de départ du développement des botnets, ne fut pas créé à des fins malveillantes mais pour aider à l'administration des canaux IRC. Les premiers botmasters ayant des buts malveillants qui suivirent étaient alors plutôt motivés par l'aspect démonstratif et la reconnaissance par les autres pirates [45]. Rapidement l'intérêt économique [46] a pris le dessus et il devient possible aujourd'hui de louer des botnets [3].

Un botmaster qui a l'emprise sur des machines zombies peut installer différents modules ou applications. Les menaces des botnets sont donc diverses voire infinies. Les exemples les plus courants sont cités dans [62] et peuvent être divisés en deux catégories : les menaces internes ou celles envers les utilisateurs infectés par le bot et les menaces externes envers d'autres machines. De plus, un bref aperçu de l'impact financier des botnets conclura cette partie.

Menaces internes

Majoritairement, les menaces internes consistent à voler des informations en masse aux utilisateurs soit directement en téléchargeant certains fichiers ou données soit en les espionnant avec des logiciels espions (spyware en anglais) qui permettent entre autres de voir ce que font les utilisateurs sur leurs ordinateurs, sur quels sites ils naviguent ou ce qu'ils frappent au clavier grâce à un enregistreur de frappe (keylogger en anglais) [6]. Grâce à ces mécanismes, le vol massif de mots de passe ou de numéros de carte de crédit est une tâche aisée.

Un attaquant peut aussi profiter d'une machine zombie pour générer des revenus notamment en installant des publiciels (adware en anglais) [46] incitant l'utilisateur à cliquer sur certains liens ou directement en téléchargeant des publicités par exemple. Malgré une rémunération individuelle faible, le botnet permet alors de multiplier celle-ci par le nombre de bots utilisés.

Enfin, une fois la machine compromise, elle devient totalement contrôlable notamment pour supprimer des données, formater des disques durs... Ce dernier type d'action n'est pas discret et profite rarement au botmaster excepté dans le cas où l'utilisateur a découvert le bot et essaye de l'utiliser pour traquer le botnet. À l'inverse, de nombreux programmes bots essayent de se montrer le plus discret

nombre de zombies pour chaque campagne de spam	10 000 - 200 000
nombre moyen de messages envoyés par un botnet	160 millions en 2 heures
nombre moyen de messages que peuvent envoyer plusieurs botnets coordonnées	1 milliard en quelques heures

TABLE 1.2 – Les botnets et le spam

possibles en se "cachant" de l'utilisateur, du système et des logiciels de protection [45].

Menaces externes

Les menaces externes regroupent les attaques vers d'autres machines sur Internet et sont difficilement détectables car l'attaque est distribuée. En premier lieu, les botnets permettent de réaliser des attaques de déni de service distribuées (DDoS). Ce genre d'attaque fait partie des plus répandues et est aussi le vecteur de nombreuses autres attaques malgré leur léger déclin [33]. Plus précisément, une attaque de déni de service contre une machine est une action qui lui empêche de répondre normalement à celles qui essaieraient en vain de s'y connecter. Cette définition générale regroupe beaucoup d'attaques et déconnecter physiquement une machine du réseau en fait partie. Cependant, on s'accorde généralement à se limiter aux attaque de déni de service par saturation c'est-à-dire celles qui envoient beaucoup de requêtes à un serveur pour augmenter sa charge et ainsi le rendre inutilisable. L'intérêt d'avoir un botnet est donc de pouvoir envoyer massivement ces requêtes de différents endroits ce qui rend le filtrage beaucoup plus difficile. Cette application est représentée dans la partie droite de la figure 1.4. Avec cette méthode, des attaques générant plus de 40 GigaOctets de trafic ont été aperçues [33]. De plus amples détails sur les différents types de DDoS ainsi que les mécanismes de protection correspondants peuvent être consultés dans [66]. On a vu précédemment qu'un vers peut aussi être capable de faire de telles attaques mais la différence ici est que l'ordre est donné par un contrôleur (botmaster) alors pour les vers purs, personne ne contrôle le processus une fois démarré.

La deuxième attaque généralement réalisée par les botnets est l'envoi massif de courriels indésirables ou spam dont l'intérêt économique sera démontré en 1.4.2. L'étude [1] constate que 85% du spam était dû aux botnets en 2006 tout en mettant en évidence les capacités extraordinaires de tels réseaux. Celles-ci sont résumées dans le tableau 1.2 dont les chiffres parlent d'eux-mêmes. Par conséquent, de plus en plus d'études sur les campagnes de spam via les botnets ont été faites [32, 21] dans l'objectif de mieux comprendre les botnets en soi mais aussi de mieux combattre le spam. Par exemple, Xie *et al.* [32] proposent *AutoRE*, un système générant des signatures de spam à partir des noms de domaines inclus dans les adresses transmises via les spams. L'envoi de courriers indésirables se faisant de manière distribuée à large échelle, il est difficile de les bloquer grâce aux techniques classiques comme l'utilisation d'une liste noire. Les travaux dans [21] montrent par exemple que les campagnes de spam durent classiquement plusieurs jours voire plusieurs mois. Toutefois, pour éviter qu'une adresse IP soit identifiée comme un bot et qu'elle ne soit rajoutée dans une liste noire, chaque bot n'est impliqué que quelques heures seulement. La charge d'email à envoyer par chacun d'eux est elle aussi bien distribuée selon leurs capacités. Ces observations montrent clairement la dangerosité des botnets dans le cadre de l'envoi de courriels indésirables.

Enfin, un botnet est aussi une base solide pour chercher des vulnérabilités sur d'autres machines (balayage ou scanning) [62] ou lancer un nouveau malware, tel qu'un vers, car un botnet permet d'infecter rapidement de nombreuses machines en même temps. De plus, il peut chercher lui même à se "propager", c'est-à-dire trouver de nouvelles machines à infester, pour agrandir la population du botnet. Dans [53], les auteurs observent deux types principaux de propagation de botnets :

– ceux qui cherchent automatiquement à trouver de nouvelles machines ;

– ceux dont le déclenchement de la propagation est dépendant d'un ordre du botmaster.

Intérêts financiers

L'étude menée dans [46] met en lumière les intérêts financiers que l'on peut tirer d'un botnet et plus particulièrement via la diffusion de publicités : paiement par affichage, par clic ou par vente. Considérons le premier cas avec I_r affichages réalisés sur la machine d'un utilisateur réel, I_f ceux sur des machines compromises (bots), R_c la probabilité de conclure une vente suite à un affichage dans le cas d'un utilisateur réel et P le gain que rapporte une vente. Du point de vue de l'acheteur, le prix C_i d'un affichage (d'où est dérivé le revenu du botmaster) doit alors être au plus égal à :

$$C_i = \frac{R_c \times P \times I_r}{I_r + I_f}$$

Ainsi, quand le nombre d'affichages non réels I_f augmente, ce prix a tendance à diminuer et oblige de ce fait les annonceurs à réduire leur coûts.

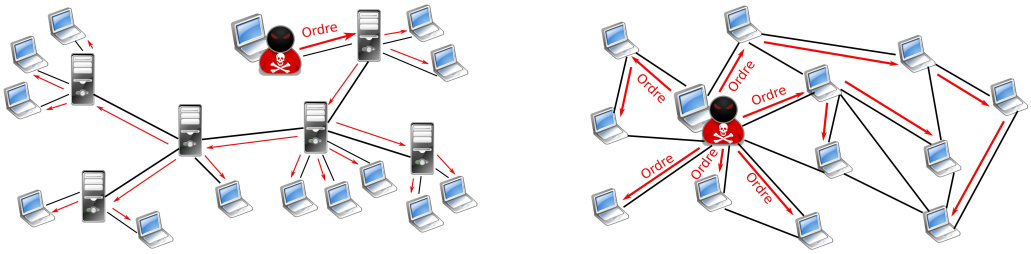
Cependant, quand le botnet occupe une part trop importante des affichages, cet équilibre n'est pas respecté, ce qui oblige le client payant la publicité à renégocier son contrat avec l'annonceur. Cette renégociation peut avoir lieu plusieurs fois de suite et déboucher sur la résiliation du contrat qui n'est bénéfique pour personne car plus aucun revenu n'est dégagé. Il y a donc un certain équilibre à respecter et c'est en jouant sur cet équilibre grâce à un réseau de machines virtuelles que les auteurs proposent de déstabiliser le botnet.

Un modèle économique plus évolué est exposé dans [28]. Sans rentrer dans les détails, il inclut la participation de pots de miel au botnet ce qui réduit son efficacité lors d'une attaque et donc son attrait. Ainsi, un attaquant à la recherche de la location d'un botnet ne paiera pas aussi cher si celui-ci contient des pots de miel.

Alors que les études précédentes avaient pour but d'établir un modèle économique, [29] évalue l'impact financier d'un botnet de manière pratique. En effet, Kanich *et al.* détournent le contrôle de Storm [31] pour lancer une campagne de spam proposant des produits pharmaceutiques. Les destinataires des courriels sont redirigés vers un site spécifique qui permet de comptabiliser réellement le nombre d'utilisateurs prêts à payer. Sur 350 millions de spams envoyés en 26 jours, seulement 28 ventes auraient pu aboutir, soit environ 2700 dollars. Cependant, seulement une partie du réseau de Storm a été détournée et en extrapolant le résultat précédent, il semble que la campagne de spam aurait pu dégager entre 7000 et 9500 dollars de bénéfice. De plus, en corrélant ces résultats avec l'évolution du nombre de machines infectées, Storm aurait permis de générer 3,5 millions de dollars par an. Enfin, compte tenu des prix du marché de la "location" (80 dollars pour envoyer 1 million d'emails [11]), les auteurs concluent que seuls les opérateurs de Storm eux-mêmes sont capables d'en tirer profit.

1.4.3 Botnets IRC

Historiquement, la renommée des botnets s'est faite grâce aux réseaux IRC (Internet Relay Chat) [8]. Ce protocole, bien que légèrement tombé en désuétude, est une messagerie instantanée. Chaque personne connectée à un serveur peut alors joindre certains canaux (channels en anglais) ou parler en privé avec une personne spécifique. Bien que ce système repose sur des serveurs, son fonctionnement assez simple permet facilement de déployer une architecture robuste composée de multiples serveurs. Ainsi, on peut considérer que les réseaux IRC sont pseudo centralisés tel que le montre la figure 1.5(a) où les serveurs forment un arbre de diffusion sur lesquels les clients, en l'occurrence des machines zombies,



(a) Architecture IRC (l'ordre du botmatser est relayé par les serveurs qui se chargent de les délivrer aux machines zombies)

(b) Architecture P2P (l'ordre du botmatser est relayé de machine zombie en machine zombie

FIGURE 1.5 – Les deux principaux types d'architecture des botnets

sont connectés. Lorsque l'attaquant envoie un ordre, celui-ci est relayé via les différents serveurs qui le transmettent également à leurs clients.

[59] donne un historique de tels botnets. Le vers PrettPark [4] (1999) semble être le premier à avoir installé un client IRC sur une machine infectée pour réceptionner des commandes. Ses actions se limitaient principalement à récupérer des données de l'utilisateur. De nombreux botnets plus évolués, notamment grâce à la capacité de pouvoir lancer diverses attaques, sont apparus au début des années 2000 [14, 65]. C'est à cette époque que les botnets se sont révélés dangereux. D'ailleurs, la détection et l'exclusion de machines zombies connectées sur des serveurs normaux a alors commencé. Ainsi, les attaquants se sont tournés vers la création de réseaux IRC dédiés.

[45] étudie en détails différents botnets IRC évolués dont les caractéristiques communes sont :

- leur modularité : il est facile pour un attaquant de coder un nouveau module pour un botnet ;
- leur capacité à récupérer des données sensibles : mots de passe, numéro de cartes de crédit, emails ;
- leur capacité à mener des attaques de déni de service (fonctionnalité "de base") ;
- la possibilité de trouver de nouveaux ordinateurs à infecter pour agrandir le botnet ;
- leurs méthodes pour éviter leur détection.

La modularité permet de créer rapidement de nouvelles variantes d'un bot [58]. Ainsi on estime à plus de 4000 les variantes de SDBot.

1.4.4 Vers de nouveaux botnets

Quelques temps après les botnets IRCs, les botnets HTTP apparurent à l'instar de BlackEnergy [12, 41]. Les commandes sont généralement postées à une adresse Web que le bot télécharge et celui-ci retourne des informations en se connectant à une certaine adresse également. Certains botnets construits sur la base de messageries instantanées, autre que IRC, auraient vu le jour [41]. Cependant, ils restent anecdotiques parce qu'ils étaient beaucoup moins souples et n'apportaient rien de plus par rapport aux botnets IRC. Le véritable changement intervient avec les botnets pair-à-pair (peer-to-peer ou P2P).

Contrairement à IRC, une architecture P2P se passe de l'utilisation de serveurs car les différents bots sont directement interconnectés entre eux. Cependant, ce nouveau type d'architecture est plus compliqué car ajouter ou enlever un bot du réseau, tout en préservant son efficacité, est plus difficile. Les ordres sont alors routés dans le réseau par les machines elles-mêmes comme illustré sur la figure 1.5(b). En réalité, chaque machine est associée à un identifiant unique sur lequel se base l'algorithme de routage. De la même manière que pour les réseaux IRC, un botnet peut s'appuyer sur un réseau P2P préexistant avec

des utilisateurs normaux ou alors créer son propre réseau.

Slapper [67], considéré comme le premier vers et botnet de ce type, est apparu en 2003. Le réseau P2P qu'il constituait était de type totalement maillé signifiant qu'un bot était connu par tous les autres, ce qui réduit son passage à l'échelle et limite sa protection car, en prenant le contrôle d'un seul nœud, il est possible de découvrir tout le réseau. Cependant, son routage aléatoire dans le réseau via plusieurs bots permet au contrôleur d'être indétectable.

Pour améliorer cela, le plus simple fut donc de se baser sur des protocoles P2P éprouvés comme Storm/Peacomm [43, 31] basé sur le protocole Kademlia [15]. Ce type de réseau P2P est en fait basé sur une table de hachage distribuée [1]. De plus, ces nouveaux botnets incluent des procédures de chiffrement pour se protéger.

Une approche hybride est proposée dans [42] où un certain nombre de pairs du réseau sont considérés comme sûrs, c'est-à-dire possédant une adresse IP statique et joignable (sans pare-feu). Ces premières machines forment alors une base solide sur laquelle peuvent se greffer les autres bots plus volatiles. Dans le même esprit Vogt *et al.* [36] proposent les super-botnets. Ils constatent que le point faible des botnets demeure le canal C&C dont la fragilité est accrue pour les très grands botnets trop visibles et détectables. De plus, les botnets de petite taille sont plus intéressant économiquement car plus facile à louer. Vogt *et al.* suggèrent ainsi de créer de multiples botnets indépendants et prouvent leur faisabilité en proposant un algorithme de routage des messages à travers celui-ci. Cet algorithme est basé sur de l'échange partiel d'information de routage entre botnets.

La force des réseaux P2P réside dans le fait que chaque bot ne connaît qu'une sous-partie du réseau (une liste de pairs), ce qui évite d'avoir un point critique dans le réseau (un serveur). De cette manière, tout le monde a le rôle de serveur et il est possible de rendre inutilisable le botnet en s'insérant puis en ayant un comportement anormal : non routage des messages et surtout pollution des autres nœuds pour les forcer à remplir leurs listes de pairs avec des faux bots (voir 1.5 pour plus de détails). Ainsi un modèle de conception avancé est évoqué dans [23] incluant les points suivants :

- une architecture P2P basée sur un nouveau réseau incluant le principe d'une relation de confiance entre 2 bots pour palier au problème précédent. Ainsi un bot se connecte prioritairement à d'autres qui ont gagné des points en ayant activement participé au routage de requêtes réelles de l'attaquant ;
- un chiffrement asymétrique permettant d'authentifier les requêtes de l'attaquant ;
- un masquage du trafic en l'encapsulant dans des protocoles standards (HTTP [7], FTP [11]...).

Enfin, [17] combine les différentes innovations des travaux précédents pour élaborer *lcbot* basé sur une structuration de bots en groupes et un routage aléatoire.

1.5 Détection et mesure des botnets

Le phénomène des botnets est devenu de plus en plus important et médiatique de telle sorte que le danger des botnets est directement corrélé à leur taille impressionnante. En y regardant de plus près, il est délicat d'estimer cette taille. Le tableau 1.3 reprend notamment quelques chiffres que l'on peut trouver. La colonne *Année* représente la date de publication du papier et donc pas forcément celle de l'étude (qui n'est pas toujours renseignée). On distingue le nombre de bots actifs à un moment donné du nombre total de bots différents durant l'ensemble de la période d'observation. Du fait que les botnets observés et les méthodes utilisées soient différentes, les chiffres ne sont pas les mêmes. En conséquence, pour mieux apprécier ceux-ci, la suite de cette section décrit les différentes méthodes possibles. Les références pour lesquelles aucun descriptif détaillé n'est exposé sont en général des rapports, ou des articles courts, plus

1. Chaque nœud du réseau indexe une partie des informations contenues dans ce réseau

Année	Référence	#botnets	#bots actif	#bots total
2004	[65]	100	2000-10000 (400 000)	800 000 - 900 000
2005	[62]	100	100 - 50 000	226 585
2006	[1]		200 000	6 -8 millions
2006	[53]	3	3000	
2008	[31]	2	5 000 - 40 000	
2008	[9]			1 500 000
2008	[32]			340 000
2009	[16]	1	16 500 - 23 000	400 000
2009	[21]			543 828

TABLE 1.3 – Taille des botnets

que des papiers de recherche et qui ne décrivent pas en détail la méthode utilisée. Les auteurs de [44] évoquent les difficultés pour mesurer la taille d'un botnet et particulièrement pour la définir.

Dans cette section, les différentes méthodes permettant d'évaluer la taille des botnets mais aussi de surveiller leurs activités seront évoquées. Cette tâche est fortement corrélée à la détection même du botnet et c'est pour cette raison que ces deux activités sont traitées ici. Cependant, les méthodes de détection pures ne seront pas détaillées, c'est-à-dire celles n'essayant pas d'évaluer le botnet dans son ensemble telle que les méthodes appliquées au niveau des systèmes de détection d'intrusions (Intrusion Detection System ou IDS) [40]. Aussi, la découverte du déploiement du botnet, par l'intermédiaire d'un vers ou de tout autre mécanisme, n'est pas approfondie ici, le lecteur intéressé pourra consulter [63, 69]. Enfin, un rapide tour d'horizon des méthodes de détection utilisées est disponible dans [22].

1.5.1 Observer le botnet

Dans un premier temps, les différentes méthodes d'observation des botnets sont décrites ici. Leurs utilisations dans les différents travaux de recherche seront détaillées dans les deux sous-sections suivantes.

La première approche pour étudier et comprendre comment marche un botnet consiste simplement à se laisser infecter par le virus qui le transporte et à voir comment se comporte la machine qui dans ce cas est un honeypot. Un honeypot ou pot de miel est une machine connectée à Internet et délibérément accessible pour pouvoir attirer les attaquants et les vers de manière à étudier les attaques menées.

Cette méthode est appliquée dans [62] où les communications IRC d'une variante de SDBot sont inspectées. Comme dans le cas d'un botnet IRC, celui-ci cherche à se connecter à un serveur, les premières informations récupérées sont donc les adresses IP ou les noms DNS [10] des serveurs. De plus, lors de la connexion, le bot peut aussi fournir un mot de passe ou les noms des canaux sur lesquels se connecter. Ce genre de technique est également utilisé dans [53] qui détermine aussi bien les ports mais aussi le langage utilisé en se basant sur la documentation publique de certains bots ainsi qu'en fouillant dans les traces du honeypot à la recherche des différentes commandes utilisées. Une fois le langage du botnet appris, le comportement du botmaster peut être examiné et le nombre de clients connectés à ce serveur permet d'estimer la taille du botnet. De plus, la liste des utilisateurs est partagée sur l'ensemble des serveurs ; en comparant celles fournies par chacun d'eux, [62] arrive à évaluer le nombre de serveurs utilisés et donc à mesurer plus justement la taille d'un botnet. En réalité, une fois que l'on participe activement au botnet en injectant ses propres commandes, la machine n'est plus seulement un pot de miel mais une machine d'expérimentation. Aussi, il est facile de directement contaminer une machine d'expérimentation. Les

travaux [31, 15, 16, 19] utilisent ce type de techniques pour étudier un botnet. [58] utilisa également la méthode du honeypot pour souligner la difficulté d'extraire des caractéristiques simples du canal C&C pour identifier un bot.

Enfin, de nombreuses méthodes collectent tout simplement du trafic et l'analysent. Ce n'est efficace que si le trafic collecté est important et concerne plusieurs machines. Cette approche est utilisée dans [52, 54, 50, 39, 18, 38, 32, 1, 21, 27].

1.5.2 Détection et mesures passives

Identifiants des bots

Concernant les botnets IRC, des méthodes assez simples basées sur les noms des bots dans des canaux IRC ont vu le jour. Elles permettent facilement de détecter la présence d'un ou plusieurs bots et donc de les compter. Dans [38], le nom d'un bot est repéré par l'utilisation d'expressions régulières traduisant des chaînes de caractères suspectes : références aux bots ("bot","l33t"), à la localisation (FR, USA), caractères spéciaux ([,], |) et utilisation de nombres. L'approche dans [18] détecte les botnets en mesurant la similitude entre les différents noms des utilisateurs de plusieurs canaux puisque les bots utilisent souvent des canaux IRC dédiés.

Utilisation du DNS

Dans [53], les auteurs recommandent de traquer les botnets à partir des DNS. Effectivement, le serveur IRC sur lequel se connecte un bot est généralement identifié par un nom de domaine, et non une adresse IP, pour permettre plus de flexibilité à l'attaquant. Ainsi, de nombreux serveurs DNS (environ 8000) sont sondés pour savoir si le nom de domaine du serveur IRC est dans le cache, ce qui indique dans ce cas qu'au moins un bot a utilisé ce serveur DNS. On peut ainsi avoir une vue globale de la répartition et de l'évolution des bots. Toutefois, une estimation précise de la taille du botnet n'est pas envisageable car il n'est pas possible d'évaluer le nombre de zombies qui se sont connectés à un serveur donné.

[54] repose aussi sur le DNS en demandant à l'autorité responsable du domaine de changer l'adresse IP correspondante de manière à router toutes les demandes de connexion au botnet vers une machine d'observation. L'objectif de cette étude est de montrer l'aspect diurne de l'activité d'un botnet qui s'explique par le fait que les machines compromises sont en grande partie des machines d'utilisateurs particuliers. Ce papier propose aussi le modèle correspondant (voir 1.6).

[50] étudie les requêtes DNSBL (DNS-based blackhole) utilisées par les serveurs mail pour savoir si l'adresse IP d'où est émis l'email est une source de spam. En fait, les attaquants vérifient que leurs bots ne font pas partie de telles listes noires avant de commencer à les utiliser. En regardant de plus près les requêtes DNSBL adressées, il est possible de distinguer celles légitimes, provenant de serveurs de mail, de celles des bots. Celles-ci peuvent être exécutées sur une seule machine ou de façon plus ou moins distribuée dans un botnet. Cependant, les auteurs relèvent deux propriétés clés qui permettent de faire la distinction :

- un serveur mail envoie des requêtes mais est aussi l'objet de requêtes contrairement à un bot ;
- les requêtes normales doivent avoir un trafic correspondant à celui des emails.

Dès qu'une ou plusieurs entités sont reconnues comme faisant partie du botnet, la liste des bots est facilement récupérée en regardant le contenu des requêtes, c'est-à-dire les IPs qui ont été vérifiées.

Il est clair que ces méthodes ne sont applicables que par des botnets utilisant des serveurs identifiés par des noms de domaines, ce qui exclut les nouvelles générations de botnets.

Activité du botnet

Un botnet actif tend à avoir des activités caractéristiques comme vu précédemment (envoi de spam, DDoS, balayage de ports...). C'est pourquoi [39] propose dans un premier temps de déceler les bots effectuant ce genre d'activités. Ensuite, cet article propose de remonter jusqu'au botmaster mais qui, dans le cas présent, se limite seulement au serveur IRC utilisé. Cette tâche commence par sauvegarder les flux réseaux concernant les bots puis cherche à trouver des couples d'adresses IP et de ports associés à plusieurs bots, ce qui constitue des serveurs IRC de contrôle potentiels. Ensuite, le trafic correspondant est considéré comme un botnet ou non en calculant sa similitude avec le trafic normal. Enfin, les auteurs mettent en évidence certains modèles périodiques dans les communications au sein d'un botnet. L'inconvénient majeur de cette méthode est la nécessité de se trouver au niveau d'un opérateur de Tier 1 pour corréler suffisamment d'informations. De ce fait, de nombreuses ressources pour capturer le trafic ou stocker ses caractéristiques sont nécessaires.

[52] utilise une méthode semblable puisque les canaux IRC de botnets potentiels sont d'abord découverts par rapport aux nombres d'utilisateurs qui semblent faire du balayage de ports. Pour raffiner la détection, des statistiques sur le nombre et la nature des messages échangés sont employées.

La reconnaissance de balayages de ports peut aussi se faire via un réseau de pots de miels et permet de calculer la population totale du botnet par extrapolation, en comparant les sources du balayage détectées par chacune des deux moitiés du réseau de pots de miel [19].

L'envoi de courrier indésirable est aussi une activité prépondérante des botnets et les campagnes de spam sont donc une source d'informations importante pour mieux appréhender les botnets. Dans [32], Xie *et al.* découpent les URLs incluses dans les emails pour les classer. À partir de chaque groupe, une expression régulière est générée et ceux dont les emails ont été envoyés en rafale sont associés à du spam. Leurs observations montrent que les campagnes sont de courte durée (5 jours). En observant plus de 7000 campagnes distinctes en 3 mois, environ 340 000 adresses IP ont envoyé du spam. Enfin, l'étude est complétée en mettant en évidence deux phases du botnet : une première phase de balayage pour chercher de nouvelles machines à infecter puis le lancement des campagnes de spam. Alors que cette technique montre un taux de faux positifs très réduit, [21] constate qu'elle peut entraîner un nombre de faux négatifs très élevé car le critère d'un envoi en rafale sur 5 jours n'est pas significatif. Dans ce papier, les traces sont collectées via un relais MTA (Mail Transfert Agent) utilisé par les bots pour éviter d'être eux même détectés. Les campagnes sont identifiées selon les URLs utilisées mais aussi selon d'autres identifiants généralement insérés dans les emails (numéro de téléphone, identifiant skype...). Même si le nombre d'adresses IP ayant envoyé du spam est comptabilisé (plus d'un demi million), le but du papier est plutôt d'étudier la durée des campagnes de spam. En bref, bien qu'un bot individuel envoie du spam en rafale, le botnet lui-même peut participer à la même campagne de spam généralement pendant plusieurs jours et notamment jusqu'à 99 jours. Le travail [27] est complémentaire de [32] car il vise à classer ou non les emails comme spam, non plus d'après le URLs contenues, mais plutôt grâce aux entêtes utilisées et notamment leurs types et leur ordre d'apparition.

Comme les techniques précédentes se basent uniquement sur l'analyse d'activités malveillantes, elles risquent de considérer des botnets qui n'en sont pas réellement. C'est notamment le cas pour le balayage de ports qui peut aussi être effectué par un vers. A l'inverse, *BotMiner* [30] inspecte aussi bien le trafic de commande du botnet que celui des attaques correspondantes. En fait, le trafic des attaques est analysé d'un côté ainsi que le trafic "normal" pour tenter de trouver des machines qui partagent le même type de communication. Ainsi, deux jeux de clusters sont identifiés et mis en relation pour essayer de détecter un botnet.

1.5.3 Détection et mesures actives

Les méthodes précédentes sont passives et ne cherchent pas à injecter de commandes dans les botnets. Leurs seules interactions avec celui-ci est leur participation au réseau via un pot de miel par exemple. Ce type de méthode fonctionne bien pour des réseaux centralisés tel que IRC car, une fois connecté, l'accès à l'information est simple. Néanmoins, il est quand même possible sur ce type de réseau de cacher des informations, notamment en rendant invisible les bots sur les canaux. De plus, les botnets décentralisés de type P2P sont rapidement apparus et ont focalisé l'attention des chercheurs dans ce domaine. Alors que la première étape reste la même, c'est-à-dire faire tourner le bot sur une machine dédiée ou un honeypot qui aurait été contaminé, il n'est pas aisé de mesurer le botnet car le bot que l'on contrôle n'a qu'une vue partielle du réseau.

[31] propose une première approche pour contourner ce problème dans le cadre de l'étude du botnet Storm. La méthode utilisée, le crawling, exploite la connaissance partielle de plusieurs nœuds. En quelques mots, le bot est exécuté puis récupère une liste de pairs. Ensuite, des commandes sont injectées dans le botnet pour récupérer la liste des pairs de chacun d'eux et ainsi de suite. Un attaquant ne peut pas supprimer un tel mécanisme car il permet au réseau de se développer et de se renforcer. Enfin, comme Storm a utilisé pendant un temps un réseau partagé avec des utilisateurs normaux (Overnet), Holz *et al.* ont espionné le réseau en introduisant des sybils [72], c'est-à-dire en créant des pairs avec des identifiants spécifiques et, dans le cas présent, répartis dans la table de hachage distribuée de telle sorte que chaque requête soit ensuite routée jusqu'à un nœud sybil. Ainsi, les pairs qui publient ou recherchent du contenu corrélé à Storm peuvent être comptabilisés (entre 5000 et 6000). D'un autre côté, la simple méthode de crawling permet d'évaluer le nombre de pairs sur le réseau dédié de Storm (Stormnet) (entre 25 000 et 40 000).

La technique du crawling n'est possible que si les pairs peuvent recevoir des requêtes, ce qui est impossible lorsqu'ils sont derrière un pare-feu ou un routeur NAT (Network Address Translator). Partant de ce problème, [15] introduit la méthode PPM (Passive P2P Monitor) plus passive mais qui reste cependant active puisqu'elle nécessite d'injecter des requêtes. Dans un premier temps, des sybils sont introduits dans le réseau, observent le trafic et reçoivent ainsi des requêtes. Les pairs à l'origine de celles-ci peuvent être comptabilisés. Ce mécanisme est simple mais se base sur le fait que, en maintenant les sybils longtemps actifs, ils seront prioritairement choisis. Cependant, un certain nombre d'adresses peuvent être usurpées (IP spoofing). Dans un second temps, les nœuds sybils renvoient une requête pour voir si un nœud est vraiment réel. Lorsqu'un nœud derrière un pare-feu envoie une requête, le pare-feu est capable d'identifier le destinataire et donc de laisser passer les réponses en sens inverse. Pour déterminer si un nœud est derrière un pare-feu, les auteurs proposent donc d'envoyer la même requête d'un autre nœud du réseau. Si une réponse est émise alors il n'y a pas de pare-feu. D'autres problèmes sont aussi considérés dans cet article et résolus grâce à l'utilisation assez longue dans le temps des sybils qui permettent d'avoir des données plus détaillées sur les autres nœuds (ils peuvent être testés à différents moments). Comme le montre la figure 1.6 reprise de l'article, la méthode PPM permet de découvrir un nombre de nœuds beaucoup plus important par rapport à la méthode de crawling. Enfin, cette méthode est validée analytiquement en montrant qu'un nœud du réseau est quasi sûr d'être identifié (98%) si celui-ci a envoyé au moins 200 messages.

A l'origine Storm utilisait Overnet et [16] introduit une méthode pour différencier les bots dans ce réseau. Grâce à la technique du crawling, deux phénomènes sont corrélés pour les repérer :

- node-id aliasing qui met en évidence les adresses IPs dont l'identifiant varie. Ceci est typique de Storm qui le fait à chaque redémarrage contrairement à un client classique ;
- Ip-address aliasing qui observe les identifiants du réseau P2P partagés par plusieurs adresses IP. Ceci est aussi classique des premières versions de Storm dont l'algorithme de génération aléatoire

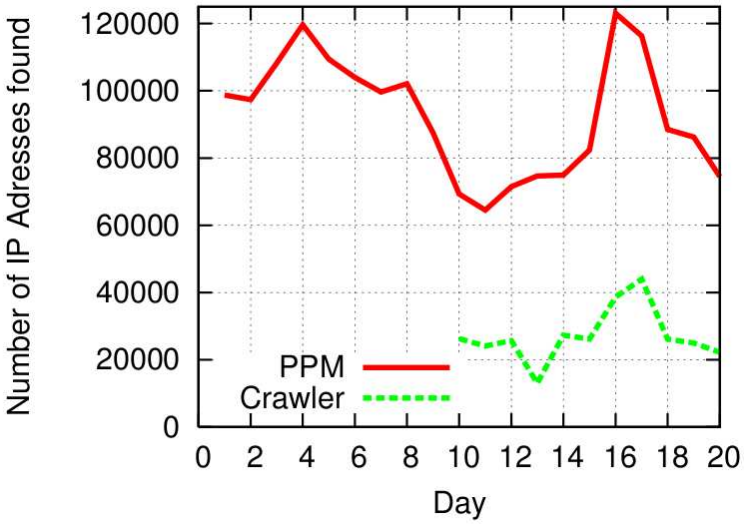


FIGURE 1.6 – Nombres de machines infectées par Storm découvertes selon la méthode classique de crawling ou PPM [15]

des identifiants était biaisé.
Comme on peut le remarquer, cette technique n'est applicable qu'aux premières versions de Storm et n'est possible que grâce à une faille dans le développement de ce dernier.

1.6 Modèles

1.6.1 Propagation d'un vers ou d'un botnet

[73] énonce un modèle mathématique permettant d'évaluer le nombre de machines infectées par un vers et en l'occurrence CodeRed. Il est dérivé du modèle infectieux de Kermack-Mckendrick [81]. Etant donné une population de victimes potentielles de taille N, le but est de trouver le nombre de machines infectées $I(t)$ au temps t. La capacité d'un vers à se propager est le taux d'infection noté $\beta(t)$ et est dépendant du temps car les observations montrent que la propagation d'un vers a tendance à surcharger les infrastructures et ralentit donc sa propagation. La figure 1.7 schématise les différents états possibles d'une machine. Au départ elle est susceptible d'être infectée puis elle peut le devenir. Ensuite, la machine peut être désinfectée. Cependant, si le vers ou la faille qu'il exploite est découvert à temps, des contre-mesures peuvent êtres mises en place et dans ce cas, la machine est directement protégée sans passer par l'état infecté. On considère donc les quantités suivantes au temps t :

- $S(t)$, le nombre de machines potentiellement infectables considéré comme inconnu ;
- $P(t)$, le nombre de machines protégées ;
- $D(t)$, le nombre de machines désinfectées ;

On obtient alors :

$$\frac{dI(t)}{dt} = \beta(t)[N - D(t) - I(t) - P(t)]I(t) - \frac{dD(t)}{dt}$$

Bien entendu $P(t)$ et $D(t)$ sont chacun dépendant de paramètres humains (réactivité) mais aussi de $I(t)$

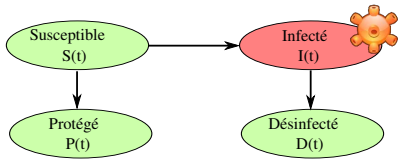


FIGURE 1.7 – Modèle infectieux d'un vers

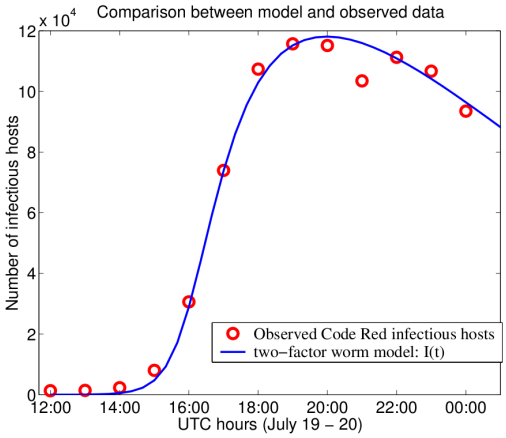


FIGURE 1.8 – Comparaison entre le modèle de [73] et le nombre réel de machines infectées par CodeRed

et $S(t)$. De plus amples explications sur la résolution de cette équation sont consultables dans [73]. Ce modèle, quant il est bien paramétré, est réaliste comme le montre la figure 1.8 provenant de l'article en question et où la courbe du modèle est proche de la réalité. Les auteurs ont dérivé un autre modèle dans [68] qui ajoute une étape de quarantaine obligatoire pour toutes les machines pendant un temps très court. Ils explorent alors les différentes actions à prendre (protection, désinfection, mise en quarantaine) dans le cas de l'apparition d'un nouveau vers et mesurent leurs effets.

Alors que CodeRed vise des serveurs Web, il n'est pas influencé par l'activité diurne humaine car ces derniers sont en permanence connectés. Les bots sont quant à eux majoritairement installés sur des ordinateurs "familiaux" qui se retrouvent éteints la nuit. Ainsi [54] étudie la propagation d'un botnet, mais plus généralement d'un vers, selon l'activité diurne. Par ailleurs, les auteurs ont aussi pris en compte les différents fuseaux horaires pour avoir une modélisation précise et aussi réaliste d'après leurs expérimentations. Par ailleurs, ils montrent que, selon l'heure à laquelle la propagation commence, les effets ne sont pas les mêmes.

Partant du constat que de nombreux vers profitent de réseaux P2P préexistants pour se propager, Yu *et al.* [24] ont établi un modèle mathématique associé et étudient notamment le nombre de machines infectées selon plusieurs paramètres importants des réseaux P2P :

- le nombre de pairs au total,
- la connectivité des pairs (nombre de pairs connectés à un seul),
- le type de topologie P2P (structurée [2] ou non),
- la vulnérabilité d'une machine (probabilité).

Le modèle est vérifié une fois de plus en le comparant avec une expérimentation réelle. De plus, les tests mettent en évidence une propagation épidémique plus rapide grâce à l'utilisation des réseaux P2P et aussi une plus grande résistance des réseaux P2P structurés.

Alors que les modèles précédents englobent déjà les effets des contre mesures comme la désinfection, [25] incorpore encore d'autres paramètres significatifs dans un modèle stochastique. Par exemple, pour limiter sa détection, un botnet peut ne pas chercher à se propager en permanence comme précédemment expliqué. Ainsi, la probabilité qu'un bot soit actif ou non est prise en compte dans ce nouveau modèle.

2. Un réseau P2P est dit structuré quand le choix des pairs d'un nœud est dépendant de son emplacement dans le réseau, c'est-à-dire son identifiant (table de hachage distribuée)

De plus, le comportement de l'utilisateur, vis à vis de l'installation ou non du bot, est aussi introduit sans oublier le caractère instable des connexions dans les réseaux P2P. Les auteurs étudient leurs modèles selon ces paramètres et cherchent aussi à déterminer quel taux de désinfection (rapidité) est nécessaire pour vraiment contenir la propagation du botnet.

1.6.2 Structure d'un botnet

Contrairement à la partie précédente qui mettait l'accent sur la propagation d'un botnet, et qui s'apparente à celle d'un vers, cette partie vise plutôt à étudier sa structure une fois déployé mais aussi lors de l'exécution de mesures de défense. L'article [37] introduit deux stratégies génériques de défense : aléatoire, qui vise à supprimer des bots au hasard, ou ciblée, qui chercher à supprimer les bots qui ont une place clé dans le réseau, c'est-à-dire ceux par lesquels de nombreux messages sont routés.

Dans ce papier, Dagon *et al.* proposent quatre types de mesures pour évaluer la structure d'un botnet :

- la capacité permettant de mesurer la taille du botnet et qui correspond au nombre de bots composant la plus grande partie composante connexe du réseau.
- la bande passante disponible qui mesure le débit que peut utiliser le contrôleur pour mener une attaque. Elle est définie comme suit :

$$B = \sum_{i=1}^{3} (M_i - A_i) P_i W_i \tag{1.1}$$

  où i représente les 3 classes possibles de connexion (1 : modems, 2 : DSl/cable, 3 : très haut débit), P est la distribution des bots dans ces classes, M la bande passante disponible, A le débit utilisé pour l'usage normal et W un poids dépendant de l'activité diurne.
- l'efficacité est synonyme de rapidité à transmettre des informations car elle mesure le nombre de sauts moyens entre deux bots quelconques. Elle est basée sur la longueur géodésique inverse :

$$l^{-1} = \frac{1}{N(N-1)} \sum_{v \in V} \sum_{w \neq v \in V} \frac{1}{d(v,w)} \tag{1.2}$$

  où V est l'ensembles des bots, $N = |V|$ et $d(a,b)$ est la distance minimale entre a et b (le nombre de sauts).
- la robustesse mesure la connectivité du réseau et donc sa redondance (nombre de chemins alternatifs) synonyme de meilleure résistance aux déconnexions. Elle est définie comme le degré moyen de la transitivité locale :

$$\gamma = \frac{1}{N} \sum_{v \in V} \gamma_v, \ \gamma_v = \frac{E_v}{C_{k_v}^2} \tag{1.3}$$

  où E_v est le nombre de connexions et k_v le nombre de bots du graphe limité au bot v et ses voisins. Grâce à des études précédentes [13], les effets de l'activité diurne sont quantifiés ainsi : $P = [0.3, 0.6, 0.1]$ et $W = [0.0625, 0.1875, 0.75]$. Plusieurs modèles de topologies de botnets sont explorés également :
  - graphe aléatoire reposant sur le modèle d'Erdös-Rényi [9] où chaque bot est connecté avec une probabilité égale à tout autre bot.
  - le modèle de Watts-Strogatz [7] basé sur le phénomène des petits mondes où chaque nouveau bot se connecte aux bots existants (ou au moins à une partie de ceux-ci).
  - le modèle "scale-free" de Barabási-Albert [6] où la probabilité qu'un bot soit connecté à un autre diminue au fur et à mesure de la croissance du botnet. Ce type de modèle crée des topologies avec des nœuds centraux ayant une grande connectivité et des nœuds "terminaux" avec peu de connexions.

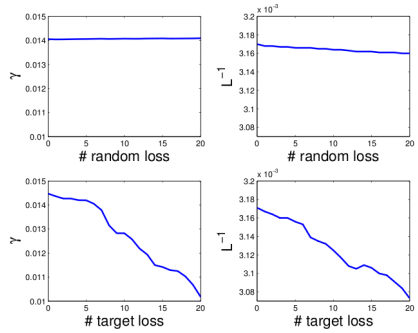


FIGURE 1.9 – Effet des défenses (aléatoire ou ciblée) dans un réseau P2P non structuré

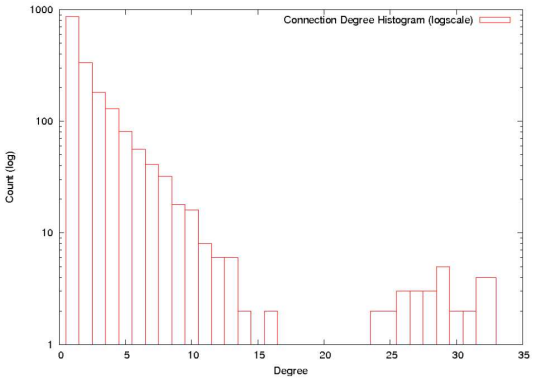


FIGURE 1.10 – Distribution des bots du vers Nugache selon leur degré de connectivité

– les modèles P2P non structurés (équivalents au modèle de Barabási-Albert) et structurés (équivalents au modèle d'Erdös-Rényi)

Sans rentrer dans les détails des études menées sur ces modèles, la figure 1.9 montre que dans un réseau de type P2P non structuré (modèle de Barabási-Albert)), les déconnexions ciblées de bots, c'est-à-dire ceux très connectés, sont très efficaces. De plus, la distribution des bots selon leur connectivité pour le botnet Nugache [5] illustre l'équivalence de ces réseaux P2P avec le modèle théorique de Barabási-Albert (voir figure 1.10). Enfin, de manière générale, un attaquant qui disposerait d'un botnet de 50 000 membres aurait en permanence au moins 1 GO de bande passante disponible pour mener des attaques.

[26] reprend les différents modèles précédents et les instancie avec deux réseaux P2P réels : Gnutella [2] (non structuré) et Overnet [6] (structuré). De plus, l'article définit d'autres métriques :

– l'atteignabillité $N_k(x)$ est définie comme le nombre de bots atteignables à un nombre maximal k de sauts à partir du bot x :

$$N_k(x) = \cup_{i=0}^{k} \Gamma_i(x),\ \Gamma_k(x) = \{y \in V,\ d(x,y) = k\} \tag{1.4}$$

Plus cette valeur est grande pour des k faibles, plus le réseau est performant car un grand nombre de bots sont proches.

– l'ensemble des chemins les plus courts pour une longueur donnée l est l'ensemble des couples de nœuds (bots) séparés par cette distance minimale :

$$L_l(u,v) = \{(u,v),\ u,v \in V,\ d(u,v) = l\} \tag{1.5}$$

Cette métrique est complémentaire de la précédente puisque plus cet ensemble contient d'éléments pour des l faibles, plus un message est transmis à de nombreux bots en quelques sauts seulement.

– le diamètre du graphe ou de la topologie, qui est une mesure de la taille du botnet comparable à la longueur géodésique. Ici le diamètre de la topologie est le minimum entre tous les chemins les plus courts entre deux nœuds quelconques.

Une fois de plus, la première stratégie de défense utilisée est de type aléatoire, c'est-à-dire que dès qu'un bot est découvert on le désinfecte. Cependant, deux types de stratégies ciblées sont introduites. La première consiste à récupérer la liste des pairs d'un nœud et ainsi désinfecter ces nouveaux nœuds (global info-based). La seconde est une approche équivalente à celle proposée dans [37] puisqu'elle cherche à déconnecter en priorité les bots à forte connectivité (tree-like), c'est-à-dire dont le nombre de pairs est élevé. Avec les métriques définies ci-dessus, les conclusions sont semblables à [37] où les

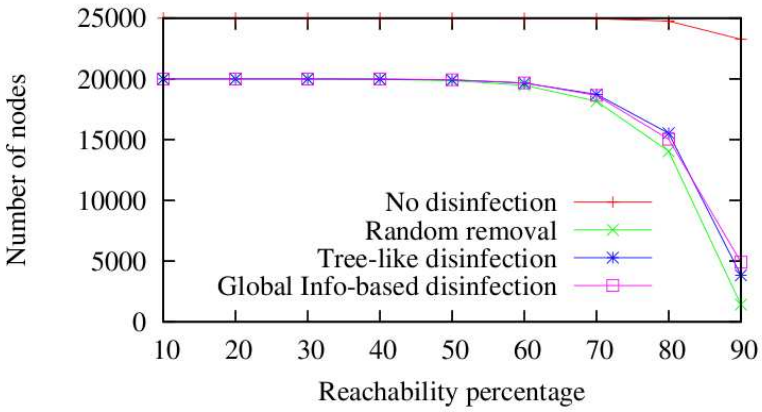


FIGURE 1.11 – Effets similaires des défenses (aléatoires ou ciblées) dans Overnet

réseaux comme Gnutella sont les plus résistants à la stratégie de défense aléatoire alors que les réseaux structurés, comme Overnet, sont quant à eux extrêmement résistants aux déconnexions ciblées. Enfin, comme le montre la figure 1.11, l'atteignabilité est quasi identique quelque soit la stratégie de défense mise en place dans le cas d'Overnet. Les auteurs de [37] avaient d'ailleurs remarqué à juste titre que des défenses plus compliquées mais probablement plus efficaces, tel que l'introduction de pollution dans les réseaux structurés (index poisoning en anglais) [49], sont sûrement réalisables. Ce genre de technique a d'ailleurs été expérimenté avec succès sur le vers botnet Storm [31] (voir section 1.5).

1.7 Bilan

Après avoir rappelé les fondements théoriques des virus, ce chapitre a montré les aspects pratiques de ces derniers et a mis en évidence les plus dangereux d'entre eux. Les vers sont, à l'heure actuelle, capables en quelques heures voire quelques minutes de contaminer des millions de machines sur l'ensemble de la planète. En parallèle ils sont souvent amenés à installer un programme de bot sur les machines infectées et constituent alors une véritable armée opérée par un ou plusieurs attaquants motivés aujourd'hui par l'intérêt économique. Il existe diverses méthodes pour les étudier mais leurs limites ont souvent été démontrées puisque que comme dans toute guerre, l'attaquant et le défenseur font preuve d'imagination chacun leur tour. C'est ainsi que les botnets P2P sont en train de submerger les botnets IRC.

Les techniques de détection et d'évaluation les plus efficaces pour lutter contre ce phénomène sont basées sur deux principes : l'introduction d'une ou plusieurs machines dans le botnet et l'injection ou la transformation des commandes dans ce dernier. Alors que la première étape est une chose facile, la seconde est beaucoup plus ardue puisqu'elle nécessite de faire de la rétro-ingénierie sur le protocole utilisé. Ce domaine d'application est introduit dans le chapitre suivant.

Enfin, la plupart des modèles proposés jusque là se limitaient au déploiement même du botnet et peu de travaux se sont intéressés à son efficacité une fois déployé car il est important d'estimer la taille générale du botnet mais aussi le nombre de machines auxquelles un ordre peut être transmis et à quelle vitesse. Mes travaux s'attaqueront à cette problématique et sont complémentaires à ceux précédemment cités.

2

Rétro-ingénierie de protocoles

Sommaire

2.1 Introduction 43
2.2 Généralités 44
2.2.1 Les protocoles réseaux 44
2.2.2 Définition 47
2.2.3 Applications 49
2.3 Découverte des messages : types et syntaxe 50
2.3.1 À partir de traces réseaux 50
2.3.2 En observant la machine hôte 52
2.4 Apprentissage de la machine à états 55
2.5 Le rejeu 56
2.5.1 Cas de figure simple 56
2.5.2 Problème 57
2.5.3 Solution 58
2.5.4 Autres applications 59
2.5.5 Définition formelle 59
2.6 Bilan 60

2.1 Introduction

En informatique, la rétro-ingénierie [28] consiste à étudier et observer l'implantation d'un logiciel pour en extraire les spécifications et les choix d'implantations. L'objectif est alors de comprendre comment ce logiciel s'exécute :

– quels sont les modules de l'application ?
– quel est le rôle de chacun des modules ?
– comment les modules interagissent-ils entre eux ?
– quel est le pseudo-code ou code source de l'application ?

Cette liste n'est pas exhaustive et les approches proposées ne cherchent pas forcément à répondre à toutes les questions à la foi. Le dernier point est assez intéressant car de celui-ci découlent les applications pratiques telles que la mise au point d'une application clone ou modifiée (notamment en enlevant les

parties relatives à la protection du logiciel) ou la découverte de vulnérabilités, dans l'application dans l'objectif de les corriger ou de les exploiter.

Il existe aujourd'hui de nombreux outils permettant de faire de la rétro-ingénierie dont les plus courants sont les désassembleurs tels que IDA [7]. Wine [1] est une mise en pratique assez spectaculaire de rétro-ingénierie puisque son but est de fournir un équivalent de l'interface de programmation (Application Programming Interface ou API en anglais) de Windows sous les systèmes Unix/Linux et qu'il est aujourd'hui capable de faire tourner de nombreux logiciels Windows sous de tels systèmes. La technique la plus courante pour se prémunir de la rétro-ingénierie est l'obscurcissement (obfuscation en anglais) qui consiste à rendre le code moins compréhensif tout en conservant les mêmes fonctionnalités et le même comportement du programme. Cela se fait généralement de manière automatique par l'intermédiaire d'outils comme `Stunnix C++ Obfuscator` [3] pour les langages C et C++. Des techniques de chiffrement peuvent également être employées. Une approche plus théorique de ces problèmes est abordée dans [25]. Les exécutables compliqués sont généralement analysés grâce à des débuggers permettant de découvrir la traces d'exécutions du programme (appel de fonctions). Ainsi, des techniques dites anti-debugging cherche à détecter l'utilisation d'un tel logiciel ou vise à l'induire en erreur (voir [17] pour une courte introduction sur ce sujet).

La rétro-ingénierie des protocoles est un domaine particulier de la rétro-ingénierie en informatique. Un protocole est généralement décrit par un document normatif ce qui permet ensuite de développer des applications utilisant ce protocole. Cependant, il arrive souvent que cette spécification n'existe pas ou seulement partiellement, notamment dans le cas de protocoles propriétaires comme ceux utilisés par le fameux logiciel de VoIP Skype [4], ceci dans le but de garder le contrôle sur le protocole et son utilisation. La rétro-ingénierie des protocoles permet alors de contourner ce problème. Hormis dans cette introduction, ce chapitre va exclusivement traiter de ce type de rétro-ingénierie.

Les techniques de rétro-ingénierie sont donc appropriées pour étudier les protocoles inconnus tels que ceux utilisés parfois par des logiciels malveillants comme les vers ou les botnets. Le chapitre précédent a clairement mis en évidence la nécessité de mieux les comprendre pour évaluer plus précisement le risque associé et pour mieux s'en défendre. La rétro-ingénierie est donc une étape essentielle dans la compréhension d'un malware communiquant. À titre d'exemple, un grand nombre de méthodes d'analyse et de protection du chapitre précédent nécessitait l'injection de messages qui n'est réalisable que si le protocole est en partie compris et maîtrisé.

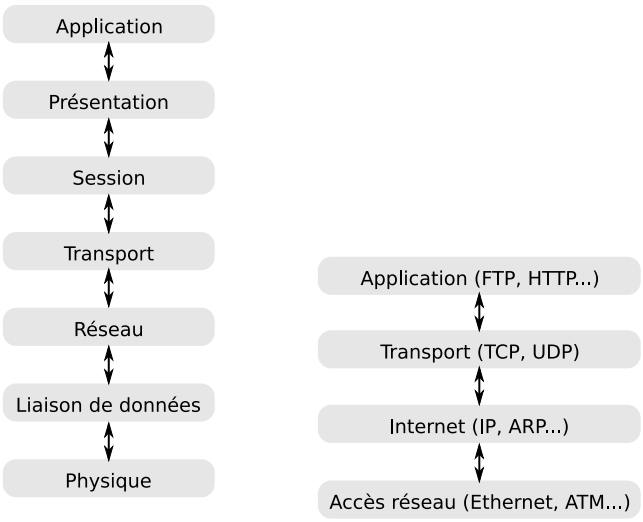
2.2 Généralités

2.2.1 Les protocoles réseaux

Un protocole de communication est un ensemble de règles permettant à deux machines d'échanger des informations. Ces règles définissent la syntaxe des messages échangés, leur enchaînement ainsi que la sémantique qui y est associée.

Dans les réseaux informatiques, le modèle OSI (Open Systems Interconnection) représenté sur la figure 2.1(a) est considéré comme trop détaillé. Ainsi, un protocole est plus souvent associé à une couche du modèle Internet aussi appelée TCP/IP. Ce modèle est représenté sur la figure 2.1(b). Les protocoles des couches basses sont suffisamment documentés et ne nécessitent donc pas de rétro-ingénierie ; seuls les protocoles de la couche application sont concernés.

Dans la suite de ce chapitre, un message correspond au contenu (payload) TCP ou UDP. La dénomination *unité de données applicative* est aussi généralement utilisée (ADUs ou Application-level Data



(a) Le modèle OSI (b) le modèle Internet

FIGURE 2.1 – Les différents modèles en couches

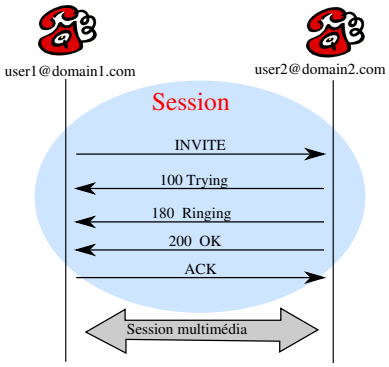


FIGURE 2.2 – Un exemple de session du protocole SIP

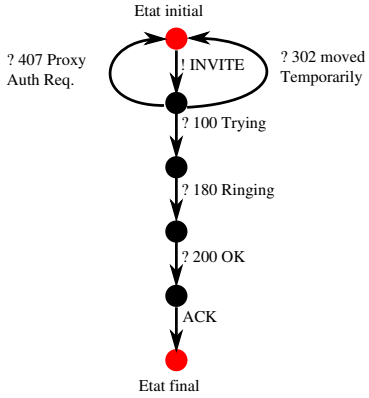


FIGURE 2.3 – Une sous-partie de la machine à état de SIP

Units en anglais). Une session est une série de messages entre deux machines dans le but d'accomplir une tâche spécifique.

La figure 2.2 montre un exemple de session du protocole SIP [6] pour établir la session multimédia permettant à *user1* de communiquer avec *user2*. Le premier message initie la demande d'appel, les 3 messages suivant envoyés par l'appelé servent à acquitter ce message, informer que le téléphone sonne et finalement que l'utilisateur a décroché. Le dernier message consiste à acquitter ces informations.

Syntaxe des messages

La syntaxe des messages est généralement définie à l'aide de la forme de Backus-Naur augmentée (ABNF) [2]. Un extrait de la syntaxe des messages SIP est donné dans la figure 2.4.

Sans entrer dans les détails, les premières lignes spécifient qu'un message commence par une première ligne (*start-line*) qui peut être une requête (*request-line*) ou une réponse (*status-line*). La suite est constituée d'aucun ou de plusieurs en-têtes, une fin de ligne (retour chariot et saut à la ligne suivante).

```
generic-message = start-line
                  *message-header
                 CRLF
                 [ message-body ]

start-line = Request-Line / Status-Line

Request-Line = Method SP Request-URI SP SIP-Version CRLF

Request-URI = SIP-URI / SIPS-URI / absoluteURI

absoluteURI = scheme ":" ( hier-part / opaque-part )
```

```
hier-part = ( net-path / abs-path ) [ "?" query ]

net-path = "//" authority [ abs-path ]

abs-path = "/" path-segments

token = 1*(alphanum / "-" / "." / "!" / "%" / "*"
        / "_" / "+" / "`" / "'" / "~" )

separator = "(" / ")" / "<" / ">" / "@" /
            "," / ";" / ":" / "\" / DQUOTE /
            "/" / "[" / "]" / "?" / "=" /
            "{" / "}" / SP / HTAB
```

FIGURE 2.4 – Extrait de la syntaxe de SIP

Enfin, le corps du message (*message-body*) est optionnel.

Un message est en fait constitué de plusieurs champs qui peuvent prendre différentes formes. Bien qu'il n'y ait pas de taxonomie officielle pour ces derniers, ce paragraphe va en proposer une dérivée des différents travaux sur la rétro-ingénierie de protocole. Les différents termes introduits seront ensuite repris au cours de ce chapitre. La figure 2.5 résume les différents types possibles. Certains protocoles peuvent envoyer des messages textuels, comme SIP [6], qui ont l'avantage de pouvoir être interprétés assez facilement par un humain. D'autres sont binaires comme DNS [10]. Bien entendu, il arrive que les deux types de données cohabitent comme dans le cas d'HTTP [7], protocole textuel mais qui peut aussi contenir des données binaires lorsqu'il est par exemple utilisé pour transmettre une image dans une page Web. Parmis ces champs, certains ont une longueur fixe et d'autres une longueur variable. Ces derniers sont généralement classés comme ci-après :

- des indicateurs de format permettant de préciser le format du message. Plusieurs indicateurs peuvent être combinés successivement dans le cas de communications encapsulées ;
- des identifiants des machines (adresses, ports...) ;
- des positions dans le message :
  - des pointeurs indiquant directement où trouver une information ;
  - des longueurs indiquant soit la longueur totale du message soit celle d'une sous-partie comme celle d'un champ (cela permet dans ce cas de retrouver la position de ceux-ci) ;
  - des décalages (offset en anglais) qui indiquent où commence chaque champ à partir d'une position de départ (généralement le début du message) ;
  - des compteurs indiquant le nombre de champs du message entier ou d'une sous-partie et qui est donc utile lors de l'analyse du message et le classement des informations qu'il contient ;
- des cookies qui sont des identifiants partagés entre les différents protagonistes pour identifier une session ;
- des mots-clés qui appartiennent à la grammaire du protocole comme par exemple *INVITE*, *ACK* pour SIP ;
- des champs divers qui transportent des informations dépendantes du protocole.

Automate

La manière dont les messages doivent s'enchaîner, c'est-à-dire quel message a besoin d'être envoyé lors de la réception d'un autre, correspond à un automate ou une machine à états finis. Par exemple, la figure 2.3 est une représentation très minimale de quelques états possibles pour le protocole SIP. Chaque transition correspond à un type de message préfixé par sa direction :

- ? : message reçu
- ! : message émis

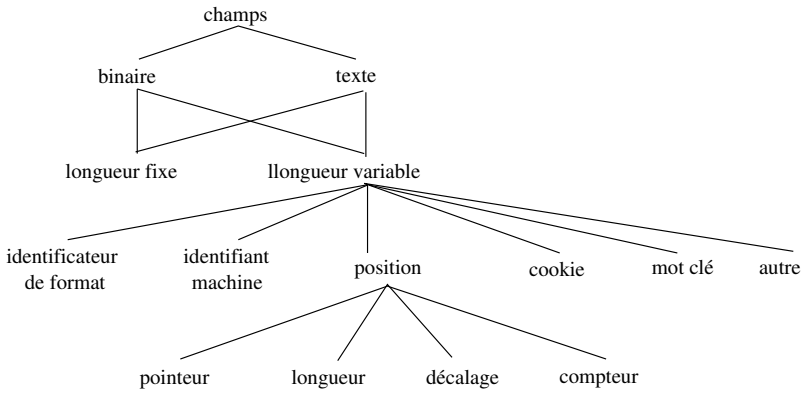


FIGURE 2.5 – Classification des différents types de champs d'un message

Cet exemple reprend exactement les messages de la figure 2.2 ainsi que deux autres possibilités :

- *Proxy Auth Req.* informant l'utilisateur qu'il doit s'identifier, celui-ci réémet alors un message *INVITE* avec son authentification incluse (une clé secrète par exemple) ;
- *Moved Temporarily* informant l'utilisateur que la personne qu'il cherche à contacter se trouve à une autre adresse, celui-ci réémet alors un message *INVITE* à l'adresse qu'il vient de recevoir.

La définition d'un type de message varie selon la granularité souhaitée mais il est facilement compréhensible qu'il ne peut être dérivé qu'à partir de champs étant des mots clés ou très peu variables. Le cas suivant est une requête HTTP pour télécharger une page Web :

```
GET /index.html HTTP/1.1 Host: loria.fr
```

Ce message peut alors être identifié comme une requête *GET* ou une requête de type *GET HTTP/1.1* si on désire introduire la version. Cependant, ajouter à la définition du type le chemin d'accès à la page ou le domaine concerné ne semble pas judicieux dans le contexte de la rétro-ingénierie.

2.2.2 Définition

La rétro-ingénierie de protocoles se définit comme la tâche qui consiste à retrouver la manière dont fonctionne un protocole à partir d'exemples d'exécution. Cette définition reste assez vague car elle regroupe plusieurs méthodes schématisées sur la figure 2.6. Tout d'abord, les exemples d'exécution ont différentes formes comme des traces réseaux stockées dans un fichier (tcpdump [2]) ou directement capturées sur un réseau local voire sur Internet. D'un autre côté, si on dispose d'un accès à une machine exécutant le protocole à étudier, alors on peut aussi analyser la machine : accès à la mémoire, aux fichiers, librairies utilisées, appels de fonction... Il est bien entendu possible de combiner ces différentes sources d'information. En effet, il est tout à fait imaginable de tester le protocole d'une nouvelle application en essayant ses différentes fonctionnalités et en inspectant les messages engendrés.

Le cœur de la rétro-ingénierie se divise en trois aspects qui sont mis en lumière par un exemple de commande ICMP (Internet Control Message Protocol) [12] où un utilisateur tente de faire un ping grâce à une requête *ECHO*. La destination n'étant pas accessible, la réponse est un message de type *DESTINATION UNREACHABLE*. La première étape consiste donc à retrouver le type des messages échangés

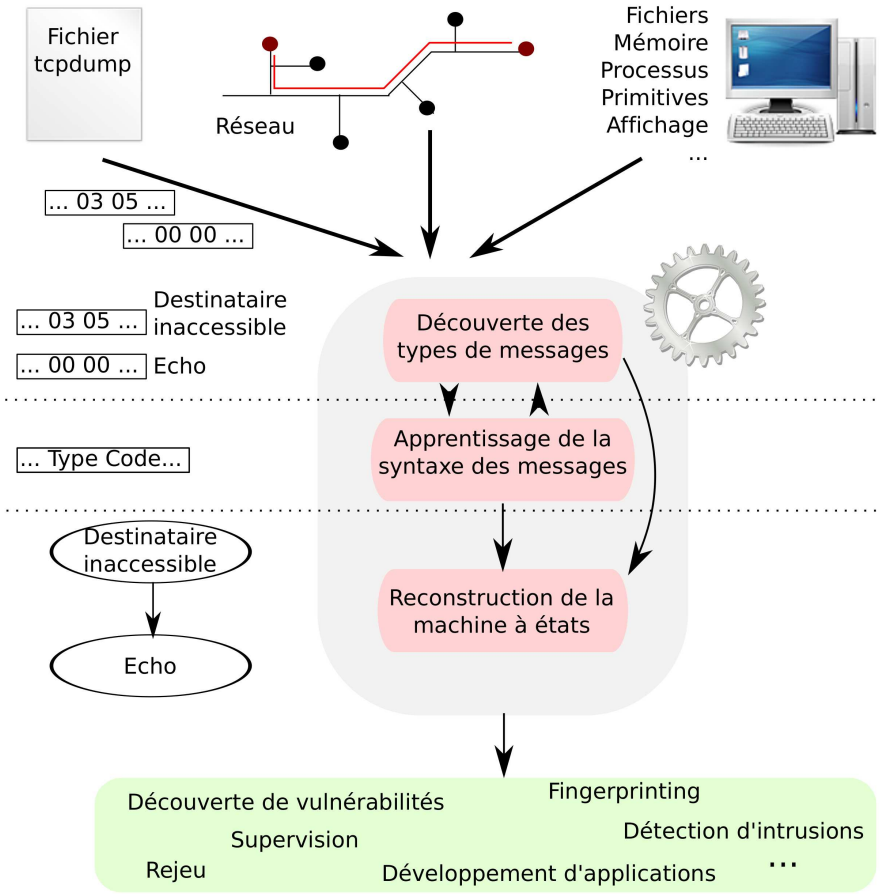


FIGURE 2.6 – La rétro-ingénierie de protocoles illustrée par un exemple sur un *ping* ICMP [12]

car tous les protocoles reposent sur plusieurs types de messages. Une fois ces types identifiés, il est alors possible d'en extraire la syntaxe comme pour le cas d'ICMP où les deux premiers octets correspondent au type du message ainsi qu'à un code. Par exemple le message *x0305* indique effectivement que le destinataire n'est pas atteignable et plus particulièrement que le routage du message a échoué. La dernière étape vise quant à elle à reconstruire la machine à états du protocole. Les types de messages ou une spécification plus précise (contenu d'un champ par exemple) sont nécessaires pour cette construction.

On notera que la découverte des types des messages et l'apprentissage de leurs syntaxes peuvent être simultanés. En effet, la syntaxe du protocole peut aussi servir à en découvrir le type car elle peut révéler l'existence d'un ou plusieurs champs d'où il est dérivé.

Historiquement, les travaux de Lee et Sabnani [27] introduisent la notion de rétro-ingénierie de protocoles de communication mais elle diffère de celle introduite ici. Leur but était de détecter si le comportement d'un programme était conforme à celui défini dans les spécifications du protocole concerné. Cependant, dans la problématique étudiée ici, aucune spécification n'est disponible.

2.2.3 Applications

Les applications de la rétro-ingénierie sont nombreuses et variées dont les plus notables sont décrites ci-après.

Développement d'applications "normales"

Il est évident que les spécifications d'un protocole dérivées grâce à la rétro-ingénierie permettent de développer une application l'utilisant. Ce fut notamment le cas pour des applications libres clones comme par exemple :

- le projet Samba [5] qui a développé un client libre pour le protocole *SMB* de partage *Windows* ;
- le protocole de messagerie instantanée MSN développé initialement par Microsoft puis utilisé ensuite dans de nombreux clients alternatifs [8] ;
- OSCAR, le protocole créé par AOL pour AIM et ICQ [7] ;
- Skype, logiciel de VoIP P2P bien connu, fut quant à lui aussi l'objet de tentatives de rétro-ingénierie [4].

Découverte de vulnérabilités

Grâce à la machine d'états du protocole, il est alors possible de voir s'il n'existe pas des moyens de parvenir dans un état d'erreur ou de générer une série de tests pour chercher des vulnérabilités [56]. La connaissance de la syntaxe du protocole permet de malformer volontairement des messages pour essayer de découvrir des vulnérabilités. Ces techniques sont regroupées sous le terme de *fuzzing* [55] à l'instar de [34].

Détection d'intrusion et supervision

La plupart des systèmes de détection d'intrusion comme Snort [76] nécessitent d'inspecter les paquets de manière précise et d'en extraire des informations selon le protocole utilisé. Par exemple, une vérification élémentaire cherche à déterminer si les messages envoyés respectent bien le protocole. De même, si on souhaite superviser le réseau et notamment obtenir des informations détaillées sur les services utilisés (ne fonctionnant pas forcément sur les ports standards) et leurs utilisateurs, on doit dans un premier temps connaître les protocoles utilisés. C'est dans cette optique que GAPAL [14] a été proposé. GAPAL permet de mettre au point automatiquement des analyseurs de protocole, pour en extraire les différentes informations, et ce à partir d'une spécification. Cette dernière doit cependant être fournie ou être dérivée grâce à la rétro-ingénierie.

Autres applications

Il existe encore deux autres types d'applications importantes :

- le rejeu vise à rejouer une communication précédente. Plus précisement, le but est de réussir à se faire passer pour une entité implémentant le protocole sans vraiment le connaître. Il est donc nécessaire pour cela d'avoir extrait quelques éléments de la syntaxe pour éviter d'envoyer des messages malformés. Cependant, la section 2.5 mettra en évidence que le rejeu peut être considéré en soi comme de la rétro-ingénierie ;
- le fingerprinting cherche à identifier des équipements d'après le contenu des messages envoyés. Le chapitre 3 lui est entièrement dédié.

– l'analyse du fonctionnement d'un malware pour comprendre la manière dont il interagit avec d'autres entités. Par exemple, la rétro-ingénierie d'un programme bot permet de savoir de quelle manière les ordres lui sont envoyés.

2.3 Découverte des messages : types et syntaxe

Une étape clé dans la rétro-ingénierie de protocoles est la découverte automatique des types de message et si possible la syntaxe associée. Comme précédemment mises en évidence, certaines techniques se basent uniquement sur les traces réseau alors que d'autres observent la machine hôte en même temps. Ces deux approches seront étudiées dans cette section.

2.3.1 À partir de traces réseaux

Initialement, la rétro-ingénierie de protocoles se faisait manuellement et était donc une tâche difficile et longue à réaliser. Le projet Samba [5] a demandé 12 ans d'effort pour avoir une spécification utilisable. De ce fait, des méthodes automatiques sont rapidement apparues. Beaucoup d'entre elles et notamment les premières se sont basées sur des travaux en bio-informatique visant à établir des liens entre les différents gènes selon leur ressemblance grâce à des techniques d'alignement de séquences. [6] s'inspire de ces travaux en considérant que chaque paquet réseau correspond à une séquence. Deux types d'alignements sont exploités :

– un alignement local dont le but est d'extraire le type général du message ;
– un alignement global visant à retrouver la syntaxe du message.

L'alignement local cherche à trouver deux sous-séquences similaires avec un score maximal et est notamment implanté par le biais de l'algorithme de Smith-Waterman [28]. Le score est calculé à partir des transformations nécessaires pour passer d'une séquence à une autre. Chaque transformation – insertion, substitution ou suppression – dévalorisera alors le score. D'un autre côté, quand aucune transformation n'est nécessaire, le score sera augmenté. Il mesure donc la similarité entre deux messages. Il devient alors possible de regrouper les messages selon leurs ressemblances.

La seconde étape d'alignement global, basée sur l'algorithme de Needleman-Wunsch [29], est assez similaire mais ne peut s'appliquer qu'à des messages identiques et donc de même type. En effet, elle considère les messages intégralement et ne se limite donc pas à chercher une sous-séquence seulement mais bel et bien une séquence globale. Par exemple, l'exemple suivant présente deux messages *HTTP* [7] réduits :

```
message 1 : GET /index.html HTTP/1.1 Host: loria.fr
message 2 : GET / HTTP/1.0 Host: www.google.com
```

On obtiendra alors l'alignement global suivant :

```
message 1 : GET /index.html HTTP/1.1 Host: ____lo_ria._fr
message 2 : GET /__________ HTTP/1.1 Host: www.google.com
```

Pour aligner les séquences, des trous ont été introduits et correspondent à des insertions ou des substitutions selon le message considéré. De plus, certains caractères sont différents d'un message à l'autre. Un consensus est ensuite dérivé où l'ensemble des trous et des caractères différents sont remplacés par un caractère joker. Dans l'exemple précédent, le consensus est :

```
GET /?????????? HTTP/1.1 Host: ??????????????
```

La syntaxe des messages HTTP est alors partiellement dévoilée en ayant notamment mis en valeur les champs fixes (HTTP/1.1) ou les mots clés du protocole (GET, Host) et les champs variables (domaine, chemin d'accès).

PEXT [16] propose d'identifier les messages de même type grâce à un algorithme de classification assez simple : la classification ascendante hiérarchique [27]. En quelques mots, la méthode considère tout d'abord chaque message comme un groupe à part entière. Les deux groupes les plus proches sont alors réunis en un seul et ainsi de suite jusqu'à ce que la distance entre les groupes les plus proches soit supérieure à un certain seuil. La distance entre deux groupes est la distance minimale entre deux messages de chacun d'eux. Cette dernière est quant à elle fonction de la plus longue séquence de bits ou d'octets partagés en se basant sur un algorithme proche de ceux évoqués lors des techniques d'alignement de séquences. La reconstruction de la machine à états est aussi abordée dans ce papier et sera présentée dans la section suivante.

L'objectif de Discoverer [8] est de plus haut niveau puisque cette application ne cherche pas seulement à découvrir les différents types de messages mais aussi leurs formats. La première étape de Discoverer vise à délimiter les champs binaires et les champs de texte. Les octets de texte correspondent à des valeurs ASCII (American Standard Code for Information Interchange) ou Unicode [1], le reste étant considéré comme binaire. Un ou plusieurs champs textuels se retrouvent entre deux marqueurs binaires. Enfin, chaque segment de texte est divisé selon la présence ou non de séparateurs (espace, tabulation...). Chaque octet binaire est considéré comme un champ à part entière puisque ce type de découpage est impossible. L'étape suivante regroupe les messages ayant la même structure, c'est-à-dire le même enchaînement de champs de texte et de champs binaires. Cette première classification incorpore aussi la notion de direction introduite dans la structure des messages car dans les protocoles clients-serveur, la nature des messages envoyés par l'un ou l'autre est souvent très différente. Pendant cette phase, les champs dynamiques et statiques peuvent aussi facilement être repérés.

A partir de ce point la sémantique des champs est étudiée pour repérer ceux de type *longueur*, *décalage* et *cookie* grâce à la méthode exposée dans [21] que les auteurs ont étendue (voir la section 2.5.3). Les *décalages* sont notamment repérés en comparant la différence entre la valeur numérique de deux champs par rapport à la différence de placement des champs suivants.

Grâce à cette sémantique et à la différenciation binaire/texte, les messages partageant exactement la même séquence sont rassemblés dans un groupe. Si aucune sémantique ne peut être associée, le contenu est directement comparé. Le nouveau format obtenu est donc de la forme :

```
format : champ*
champ : type, variable, sémantique
type : binaire | texte
variable : statique | dynamique
sémantique : aucune | longueur | décalage | cookie
```

Ainsi Discoverer est capable d'inférer les formats des messages. Cependant, des messages de même format peuvent être de type distinct. On peut citer le cas du protocole SMTP [1] :

```
commande 1 : MAIL john@loria.fr
commande 2 : RCTP oscar@inria.fr
```

Ces deux commandes sont bien de types différents (MAIL et RCTP) mais partagent le même format : deux champs variables sans sémantique dans le cadre des définitions précédentes. Les identificateurs de formats (IF) visent à différencier les deux types de messages. Ici le premier champ est un IF. Sans trop entrer dans les détails, un champ est assimilé à un IF lorsqu'un champ dynamique a un nombre de

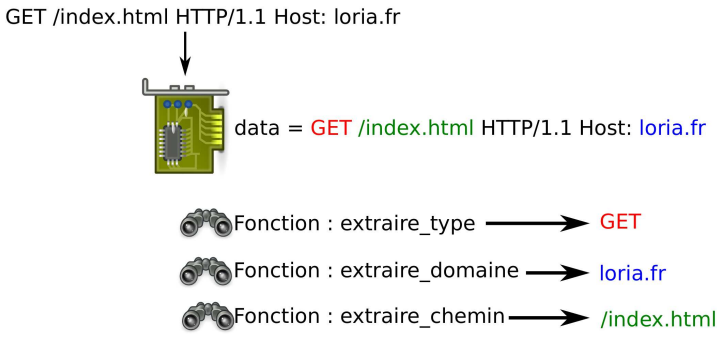


FIGURE 2.7 – Analyse par tainting

valeurs possibles réduit. Ensuite pour chacune de ces valeurs, le groupe est séparé en deux. Dans le cas de l'exemple ci-dessus, il faut bien entendu que les traces capturées soient assez conséquentes de manière à observer plusieurs messages MAIL et RCTP.

Après ces différentes phases de classification, Discoverer découvre beaucoup plus de formats de messages qu'il en existe réellement. Cela provient de la classification initiale qui impose aux messages d'un même groupe d'avoir exactement la même structure. Une ultime phase consiste donc à regrouper les groupes (ou formats) en utilisant l'alignement de séquences [29] non plus octet par octet mais champ par champ ou deux champs sont égaux si et seulement si ils sont de même type (binaire ou textuel) et s'ils partagent la même sémantique ou ont au moins une valeur commune.

Les deux principaux inconvénients de cette méthode sont l'absence de sémantique associée aux champs binaires (chaque octet est un champ à part entière) ainsi que l'utilisation supposée de séparateurs prédéfinis (espaces, tabulations...), ce qui représente une condition assez forte.

Dans [19], l'approche avancée est semi-automatique puisqu'elle demande à l'utilisateur d'être capable d'identifier manuellement au préalable le champ qu'il souhaite extraire du message, ce champ peut alors correspondre au type du message. La méthode d'apprentissage employée consiste à générer des exemples représentant le champ extrait. Ils correspondent en fait à des conditions sur les octets du message. Par exemple, le champ délimité par l'utilisateur peut toujours être précédé par un octet égal à *00*. Des contre-exemples sont générés également. L'ensemble de ces exemples servent alors à extraire ce champ dans un autre message et l'utilisateur confirme si la réponse est bonne ou mauvaise. Ce processus continue ainsi de manière incrémentale jusqu'à ce que l'utilisateur considère que le système est assez efficace.

2.3.2 En observant la machine hôte

Caballero *et al.* [15] constatent que les traces réseaux manquent de sémantique et la difficulté pour associer une sémantique aux différents champs d'un message a été mise en évidence au paragraphe précédent. Le système exposé dans ce papier se nomme *Polyglot* et l'approche consiste à analyser dynamiquement la manière dont un message est traité. La première étape doit extraire une trace d'exécution système à partir d'un message et de l'analyse du programme qui le traite grâce à une technique de *tainting* [14]. Cela demande de marquer chaque paquet arrivant sur l'interface réseau de la machine et à propager ce marquage sur chaque résultat d'opération appliquée sur les données de celui-ci. Une opération peut être arithmétique, logique ou un déplacement de données. Le marquage indiquera quelle partie des don-

nées originales sont utilisées. Par exemple, lorsqu'un champ est extrait du message et stocké dans un registre, ce registre se retrouve marqué. Le cas simple d'une requête HTTP et la manière dont elle peut être traitée sont proposés sur la figure 2.7. En fait différents champs sont extraits et le but de l'analyse par *tainting* est de garder en mémoire d'où proviennent ces différents champs.

De manière résumée et en reprenant la nomenclature des types de champs précédemment introduite, *Polyglot* procède de la manière suivante. Pour les champs de positions tels que les pointeurs ou les décalages, il suffit de voir qu'une donnée marquée est employée pour effectuer un déplacement (comme par exemple une boucle pour avancer dans les données). Les données pointées par ces positions sont logiquement considérées comme des champs de longueurs variables. Intuitivement, un protocole utilisant des séparateurs ou des mots clés va effectuer des comparaisons qui sont comptabilisées par *Polyglot*. Dans le cas d'une requête HTTP comme sur la figure 2.7, l'application bouclera sur les octets en comparant leur contenu aux différents séparateurs possibles (espace, tabulation...). Enfin, tout autre donnée marquée mais non traitée par une de ces techniques correspond à un champ de longueur fixe.

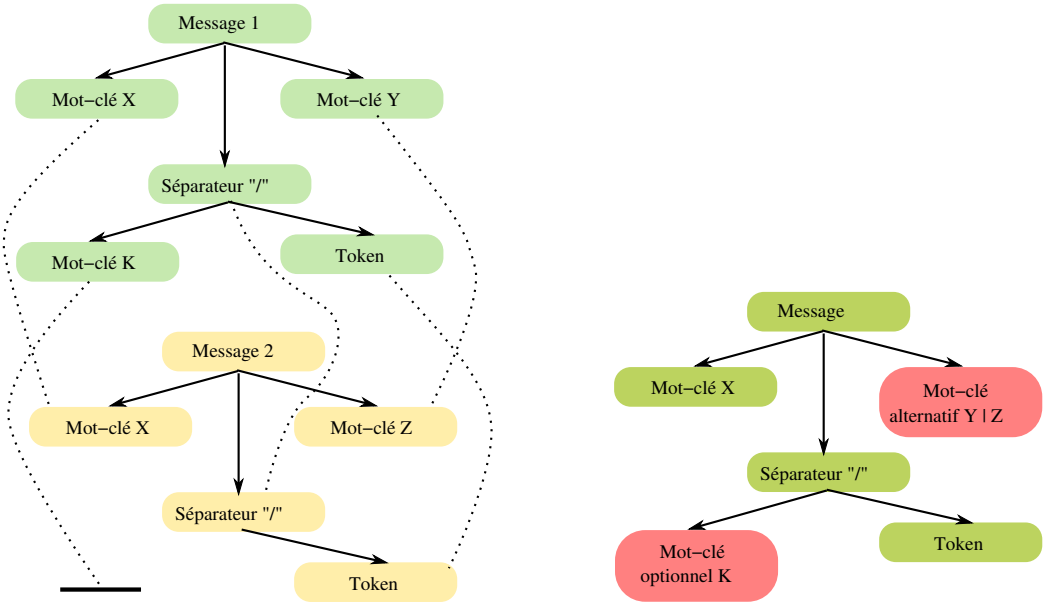
Cette première approche reste toutefois limitée à l'analyse d'un message à la fois et aucune méthode de classification ne permet de faire des regroupements comme avant pour essayer d'inférer les différents types de messages ainsi que leurs formats.

AutoFormat [10] a pour ambition de retrouver, en plus des différents champs, la structure hiérarchique qui leur est associée c'est-à-dire d'être capable de retrouver une spécification semblable à celle fournie par une grammaire ABNF. Effectivement, sur l'exemple ci-dessous extrait de la grammaire du protocole HTTP, il est clair que le champ *Request-Line* est divisé en plusieurs autres telle une relation hiérarchique :

```
Request = Request-Line
          *(( general-header | request-header | entity-header ) CRLF ) CRLF
          [message-body]
Request-Line = Method SP Request-UTI SP HTTP-Version CRLF
```

Il existe aussi une relation séquentielle entre certains champs comme sur la dernière ligne de cet exemple. De plus, les champs interchangeables, tout en respectant la grammaire, respectent une relation dite parallèle à l'instar des différentes entêtes (headers) dans l'exemple. La méthode appliquée est proche de la précédente (*Polyglot*) mais introduit en plus la notion de pile d'exécution. La trace d'exécution correspond aux différentes opérations sur les données marquées (*tainting*) tout en conservant en mémoire une représentation de la pile à l'exécution. Ensuite, toutes les actions successives avec la même pile sont regroupées en une seule pour créer un nœud. Dans le cas contraire (pile différente), un nœud à part est créé et rattaché à un autre précédemment créé qui contient en partie les même données ou, à défaut, au nœud racine. Par exemple, si le texte "GET /index.html" a été déplacé à un endroit de la mémoire et ensuite seulement "GET", le nœud induit par le second déplacement sera un fils de celui généré par le premier car "GET" est une donnée marquée contenue dans "GET /index.html". Après un affinement secondaire de l'arbre, un historique des piles d'exécution est adjoint à chaque nœud et correspond à l'évolution de celle-ci à travers l'ensemble des ancêtres du nœud en question. Deux nœuds partageant un historique similaire sont supposés parallèles car ils se sont exécutés dans le même contexte. À partir de ces relations hiérarchiques et parallèles, les relations séquentielles sont alors facilement extraites.

AutoFormat est donc capable d'extraire ces différentes structures sur un message pour notamment en dériver une structure hiérarchique. Le seul inconvénient est qu'il n'est pas pour l'instant capable d'inférer la syntaxe du protocole puisque seul un message est utilisé. C'est dans cette optique que Wondracek *et al.* ont proposé [12]. En utilisant des techniques similaires à *Polyglot* [15], la sémantique associée aux champs est mise en exergue. Les auteurs arrivent notamment à extraire deux types de champs : les chemins d'accès à des fichiers et les données communes à une requête et à sa réponse, ce qui est seulement possible en utilisant de multiples messages. Une structure hiérarchique est aussi induite en repérant



(a) Alignement de deux messages

(b) Arbre général

FIGURE 2.8 – Déduction du format général d'un message

les données marquées (tainting) qui sont des sous-parties d'autres données marquées elles aussi. Un arbre général est déduit en exploitant les techniques d'alignement de séquences [29] sur plusieurs arbres. L'alignement est d'abord appliqué sur les feuilles des arbres d'après leurs sémantiques. Par exemple, des champs de longueurs correspondent si leurs tailles sont identiques, si les champs qu'ils délimitent correspondent aussi. Le score de chaque alignement est propagé aux nœuds pères et ainsi de suite pour obtenir un score global. L'exemple donnée dans la figure 2.8 repris de [12] donne un aperçu du fonctionnement de l'alignement et surtout de son résultat. Au premier niveau (le plus bas), un *token* (champs sans sémantique associée) est délimité par le caractère séparateur "\\" mais le premier message comporte en plus le mot-clé K, l'alignement va donc insérer un blanc (suppression) à cet endroit. En appliquant ce processus récursif en remontant l'arbre, il apparaît que la structure des messages est identique exceptée pour les mots clés Y et Z, le résultat de l'alignement dans ce cas là est donc une substitution. L'étape finale consiste alors à traduire cet alignement en format de messages où une suppression concorde avec quelque chose d'optionnel tandis qu'une substitution représente une alternative comme présenté dans la figure 2.8(b)

Cui *et al.* [11] s'inspirent des travaux précédents basés sur l'analyse par *tainting* tout en mettant en lumière leurs limitations. Tout d'abord, les messages ne comportent pas uniquement des champs mais aussi des données de taille variable, comme par exemple un fichier multimédia. Deuxièmement, les valeurs des différents champs peuvent être contraints entre eux à l'instar des vérifications par somme de contrôle (checksum en anglais) ou les numéros de séquences et d'acquittements TCP. Ces informations sont essentielles pour produire efficacement des signatures pour les attaques de type 0-jour (ou 0-day en anglais) [35]. Le système proposé, *Tupni*, recherche les différentes séquences de données en cherchant les boucles permettant de les lire, c'est-à-dire des boucles telles que chaque itération traite une nouvelle partie du message. En se basant sur la trace d'exécution d'une application et notamment les différentes conditions testées, Tupni déduit les différentes contraintes par rapport aux données marquées, c'est-à-dire le message original.

Enfin, on peut citer *ReFormat* [13] qui, pour la première fois, cherche à inférer la syntaxe de messages

chiffrés. Intuitivement, les fonctions de déchiffrement font appel à de nombreuses opérations arithmétiques et logiques. *ReFormat* commence donc par détecter les phases où ces instructions sont grandement utilisées, ce qui correspond à la phase de déchiffrement. Puis, un retour à la normale indique que le programme commence à traiter le message déchiffré. Une deuxième phase s'appuie sur *AutoFormat* [10] mais il est nécessaire de savoir où a été stocké le message déchiffré. Par conséquent, les auteurs suggèrent d'observer le temps de vie des zones de la mémoire tampon c'est-à-dire le temps pendant lequel elles sont allouées. *ReFormat* repère l'endroit où le message déchiffré est localisé car il doit se trouver dans des zones qui sont en vie pendant la phase de déchiffrement et la phase de traitement du message.

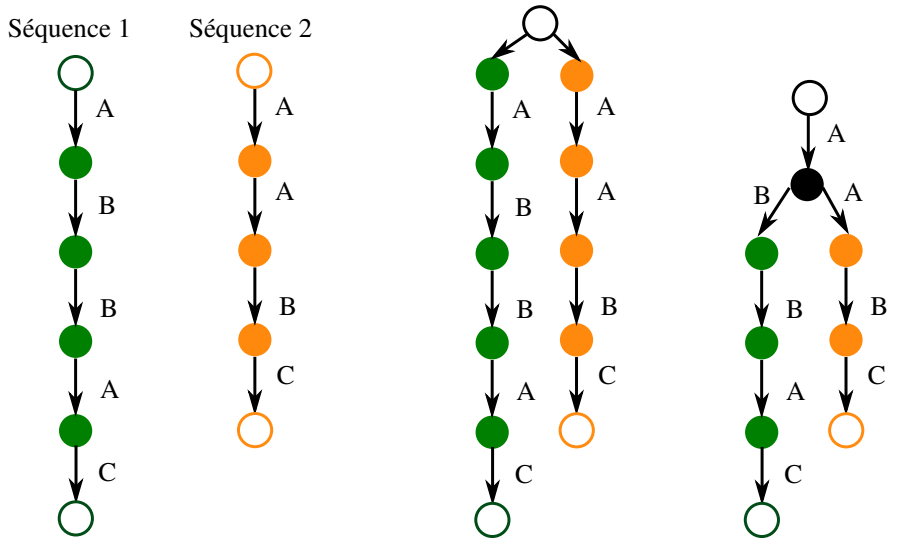
2.4 Apprentissage de la machine à états

Peu de travaux s'attellent à définir la machine à états du protocole alors que la plupart se focalisent sur l'inférence des types de message ou de la syntaxe de ces derniers. Supposons que chaque message soit associé à un type grâce à une des méthodes précédemment détaillées, il est donc possible d'observer l'enchaînement des types de messages. Grâce à ces exemples d'exécution, le but est de reconstruire l'automate le plus petit tout en étant cohérent avec les exemples. A proprement parler, ce problème n'est pas spécifique au domaine de la rétro-ingénierie et est fortement lié aux problèmes d'inférence de grammaire d'un langage à partir d'exemples. Ce problème est connu pour être NP-complet [31] et l'ensemble des algorithmes pour le résoudre sont des approximations qui demandent souvent des contre-exemples [26]. Dans ce contexte, il faudrait être capable de distinguer les enchaînements qui aboutissent à un état acceptable et inversement. Ensuite, de nombreux travaux permettaient de tester l'automate en temps réel [30, 29]. Dans ce cas, cela correspondrait à envoyer activement des messages vers l'application et en observer le comportement résultant pour en déduire la machine à états. Ce type d'approche est aussi difficile à mettre en œuvre. Les méthodes les plus récentes sont basées sur des arbres et plus particulièrement des arbres APTA (Augmented Prefix Tree Acceptor en anglais) [24]. Par exemple, sur la figure 2.9(a), deux séquences de messages sont représentées puis sont regroupées dans un arbre sur la figure 2.9(b) tout simplement en rajoutant un nœud racine. Enfin, l'arbre APTA de la figure 2.9(c) est obtenu en groupant les préfixes communs. Les méthodes actuelles cherchent alors à simplifier ces arbres en regroupant des états. Nombre d'entre elles sont décrites dans [24].

La rétro-ingénierie peut donc s'appuyer sur ces travaux pour produire la machine à états. Deux papiers détaillent ces aspects de la rétro-ingénierie [9, 16]. Prospex [9] repose sur les travaux [15, 12] pour inférer le format des messages mais propose dans un premier temps de découper une session entière en plusieurs messages. En effet, il peut arriver qu'un message soit découpé en plusieurs paquets réseaux et il est alors plus simple d'étudier l'ensemble de la communication c'est-à-dire la session. Pour découper une telle session, Prospex repère à partir de quel moment, une réponse est émise. Par exemple, si après avoir lu deux paquets, une réponse est envoyée alors ces deux paquets correspondent à un message. Ensuite, un algorithme de classification est appliqué pour trouver les différentes classes ou types de messages. Celui-ci repose sur une mesure de distance entre deux messages, elle-même basée sur plusieurs caractéristiques :

- les formats des messages ;
- le comportement de l'application lors du traitement d'un message : fonctions appelées, librairies chargées...
- ce que l'application produit en sortie : messages envoyés, données écrites sur le système de fichiers.

Selon les caractéristiques étudiées, la similitude est une fois de plus mesurée grâce à la technique de l'alignement de séquences [29] ou alors grâce à l'indice de Jaccard [30] qui est le quotient entre l'intersection et l'union de deux ensembles. Dans le cas présent, les librairies chargées peuvent constituer un



(a) Séquences initiales (b) Groupement des séquences (c) Arbre APTA

FIGURE 2.9 – Construction de l'arbre APTA

tel ensemble. Une fois les messages classés, un arbre APTA est construit et un algorithme de regroupement d'états est appliqué. Sans entrer dans les détails, il est nécessaire que chaque état soit étiqueté. En supposant que tous sont des états d'acceptation, l'arbre se réduirait à un seul état. Chaque état est donc étiqueté par les conditions nécessaires pour y parvenir.

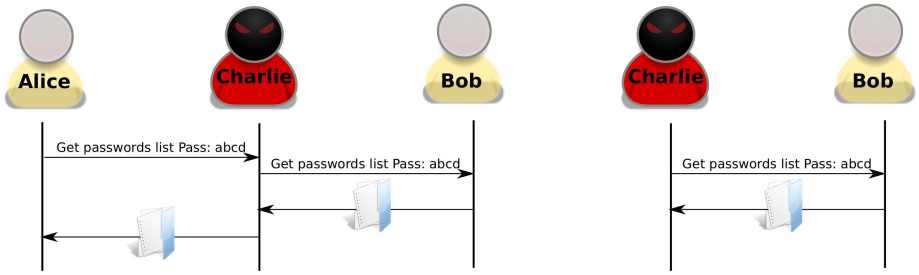
PEXT [16] construit quant à lui l'automate général d'une application qui peut utiliser plusieurs connexions comme c'est le cas pour FTP [11]. La méthode commence d'abord par découper la séquence de messages selon les différents flux puis recherche les séquences de messages identiques ou similaires qui ont pu être observées. Ce découpage crée donc des états qui peuvent inclure plusieurs messages. Pour chaque état, il existe donc une séquence de messages, les états identiques sont regroupés. L'automate ainsi défini est alors transformé notamment pour rendre les transitions déterministes. La méthode proposée est donc simple mais risque d'être mise à mal avec des protocoles complexes où la même séquence de messages peut apparaître plusieurs fois mais dans des contextes différents, ce qui n'est pas pris en compte ici.

2.5 Le rejeu

2.5.1 Cas de figure simple

Le rejeu consiste à pouvoir communiquer avec une autre machine à partir d'observations d'échanges entre deux entités réelles. Il n'est pas forcément nécessaire de retrouver toute la syntaxe d'un protocole ou sa machine à états. Le but du rejeu n'est pas unique et peut prendre différentes formes.

A titre d'exemple, l'attaque par rejeu peut permettre d'outrepasser certains droits en usurpant l'identité d'une tierce personne qui possède réellement ces droits. Sur la figure 2.10(a), Charlie commence par faire une attaque de type *l'homme du milieu* appelée plus communément en anglais *man in the middle*. Plusieurs choix s'offrent à lui selon la situation ; par exemple, il pourra espionner les communications



(a) Dans un premier temps, Charlie, l'homme du milieu, intercepte la communication

(b) Dans un second temps, Charlie souhaite mettre à jour sa liste de mots de passe, il envoie donc le message qu'il a capturé précédemment

FIGURE 2.10 – Attaque par rejeu simple

sur le réseau local de Bob ou Alice ou utiliser l'ARP spoofing pour se faire passer pour Alice auprès de Bob et inversement. De ce fait il est capable d'observer la requête d'Alice et le fichier confidentiel que Bob lui envoie. Quelques jours après, Charlie veut mettre à jour la liste de mots de passe qu'il a réussi à dérober auparavant. Cependant, l'attaque de *l'homme du milieu* échoue [47], il décide alors de mimer Alice et de renvoyer la requête qu'il avait capturée sans aucune idée de son sens exact ou de sa syntaxe. Finalement, Bob lui envoi le nouveau fichier de mots de passe.

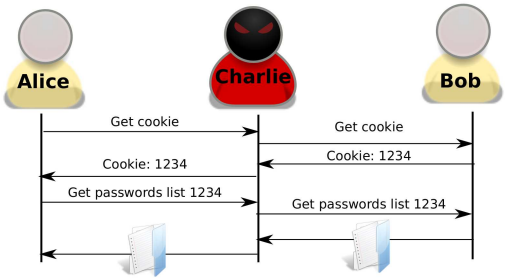
2.5.2 Problème

L'attaque précédente est extrêmement simple et peut facilement être évitée grâce à l'utilisation d'un cookie qui permet d'identifier une session. Comme illustrée sur la figure 2.11(a), Alice va dans un premier temps récupérer un cookie auprès de Bob qu'elle pourra alors inclure dans la requête suivante pour obtenir les mots de passe. Bob vérifiera alors que c'est bien la même session grâce au cookie. Lors de l'attaque sur la figure 2.11(b), Charlie rejoue les paquets mais ne tient pas compte de la réponse de Bob. En conséquence, il demande le fichier avec un mauvais cookie que Bob détecte facilement.

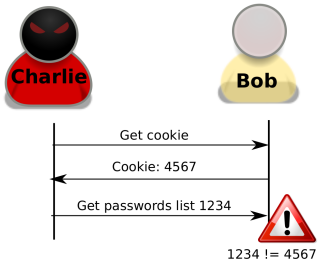
Dans cet exemple, il suffirait donc de repérer la présence d'un identifiant (le cookie) et de remplacer sa valeur par celle reçue dans le premier message. Dans le cas d'une application réelle, les modifications à apporter peuvent être beaucoup plus nombreuses et en définir la liste exhaustive est impossible :

- le message peut contenir par exemple des informations relatives à des couches réseaux inférieures (adresses IP, ports) voire le domaine réseau de la machine ;
- toute sorte d'identifiants de session comme les cookies ;
- des compteurs à l'instar des numéros de séquence TCP ;
- l'heure courante ;
- ...

TCPReplay et TCPRewrite [2] permettent de modifier des paquets puis de les rejouer mais il est nécessaire de fournir manuellement les règles de réécriture. Un cas plus complexe mais bien réel est celui du protocole de transfert de fichiers FTP [11]. Une première connexion est nécessaire pour échanger les commandes et une deuxième pour transférer les données identifiée par un numéro de port sur lequel se connecter. Cette information est donc envoyée par le client ou le serveur selon le mode (passif ou actif) par l'intermédiaire de la première connexion dite de commande. Pour rejouer un échange FTP du côté



(a) Dans un premier temps, Charlie, l'homme du milieu, intercepte la communication



(b) Lors du rejeu, Charlie n'envoie pas le bon cookie, l'attaque échoue

FIGURE 2.11 – Protection de l'attaque par rejeu par l'utilisation d'un cookie

de l'hôte recevant le numéro de port, l'extraire du message de commande est donc obligatoire.

2.5.3 Solution

Dans le cadre de l'amélioration de leur honeypot, C. Leita *et al.* ont conçu ScriptGen [23], un outil permettant de générer automatiquement des réponses à des requêtes sans avoir à priori connaissance du protocole utilisé. Comme de nombreux services sont attaquables et pas forcément implantés sur la machine du pot de miel, ce dernier doit les émuler grâce à des scripts spécifiant leur fonctionnement. Écrire un script pour chaque service est alors une tâche ardue, notamment dans le cas de protocoles non documentés, d'où l'intérêt de ScriptGen. Son fonctionnement se base sur [6] où l'alignement de séquences local aide à détecter les différents types de messages. Une deuxième étape consiste à identifier les régions semblables dans un message par rapport à la présence ou non de celles-ci, aux types de données (ascii, binaire...), leurs valeurs habituelles ou leurs variabilités. Contrairement à la recherche d'un consensus comme dans [6], le but est de trouver ici les messages de même type qui font à peu près la même chose. Par exemple, ScriptGen arrive facilement à observer différents types de téléchargements selon le type de fichier demandé (html, gif...). La limitation actuelle de cette approche comme toutes celles basées sur l'apprentissage à partir d'un jeu de données est la complétude de celui-ci.

Le problème souligné dans la section précédente met en évidence la nécessité de chercher dans les messages les champs statiques, qui n'auront pas besoin d'être modifiés, et les champs dynamiques, qui nécessiteront des modifications pour exécuter un rejeu correct. *RolePlayer* [21] a pour ambition de résoudre ce problème. Dans ce cas, un message désigne le contenu du segment TCP ou UDP. Tout d'abord, il y a un certain nombre d'informations classiques communes à beaucoup de protocoles comme les adresses IP les ports TCP ou les noms d'hôtes. Ces différents éléments sont extraits de la configuration de la machine où va être rejouée la communication mais aussi fournis par l'utilisateur en ce qui concernent les machines distantes. *RolePlayer* entreprend alors de repérer ces différentes informations dans les messages échangés et de les découper en fonction. Chaque paire de segments de message est alors testée en utilisant la méthode d'alignement déjà proposée dans [6]. Ainsi, il peut repérer des champs redondants comme les cookies. De plus, les protocoles spécifient souvent un champ contenant la longueur du message, [21] recherche ce type de champ en utilisant la taille totale du message et les tailles des différents champs qu'il a détectées précédemment.

2.5.4 Autres applications

Bien que l'exemple de l'attaque par rejeu soit simple mais suffisant pour expliquer le principe du rejeu et ses problèmes, le domaine d'application du rejeu est bien entendu plus étendu dans [21]. Pour continuer dans le domaine de la sécurité, lors de l'apparition d'un nouveau vers sur Internet, il peut être intéressant de voir si celui est totalement nouveau ou dérivé d'un autre plus ancien. Dans ce cas, essayer de rejouer un dialogue de l'ancien vers avec le nouveau est une investigation possible. De plus, il peut alors être intéressant de commencer le rejeu jusqu'à atteindre un point où il devient impossible (les réponses diffèrent trop de la trace capturée) et dans ce cas, passer la main à un pot de miel de haute interaction [51]. Grâce à ce mécanisme, la charge du pot de miel est réduite et le traitement des attaques est accéléré.

D'un autre côté, lors de la découverte d'une faille dans une application, le rejeu permet de tester cette faille avec d'autres versions du logiciel en rejouant simplement l'attaque contre celui-ci. De plus, rejouer l'attaque est une manière de vérifier si la correction de la faille est vraiment efficace. Plus généralement, si on dispose d'un jeu d'attaques, l'efficacité d'un IDS comme Snort [76] peut être évaluée avant de le déployer sur son réseau.

De plus, effectuer une batterie de tests par rejeu peut s'appliquer à n'importe quel domaine tel que l'étude du comportement d'un serveur face à une charge élevée de requêtes valides. Supposant un protocole donné, le rejeu peut aussi servir à valider si une nouvelle application le mettant en pratique fonctionne correctement à partir d'une capture réalisée grâce à un logiciel préexistant.

Enfin, le rejeu permet de créer rapidement une petite application clone grâce à quelques exécutions d'une autre. Reprenons l'exemple réduit du chargement d'une page Web :

```
message 1 : GET /index.html HTTP/1.1 Host: loria.fr
message 2 : GET / HTTP/1.0 Host: www.google.com
```

[21] doit réussir à détecter les différents champs statiques et dynamiques d'autant plus que l'utilisateur peut lui renseigner un certains nombres d'informations comme l'adresse qu'il a tapé dans la barre de recherche. Il est alors clair qu'une requête est de la forme :

```
GET chemin HTTP/1.1 Host: domaine
```

Une application simple sera donc capable de remplir les champs *chemin* et *domaine* à partir d'une adresse donnée par l'utilisateur.

Ces exemples d'applications montrent clairement que le rejeu est assimilable à la rétro-ingénierie de protocole car il permet de détecter des éléments de syntaxe du protocole et ses applications sons similaires à celles de la rétro-ingénierie de protocole. De la même manière, une bonne connaissance du protocole par l'intermédiaire de cette dernière aide à effectuer un rejeu plus efficace. La limite entre ces deux domaines reste donc floue. Une différence notable reste la construction de la machine à états car le rejeu se limite à rejouer une même séquence vu auparavant alors que la machine à états doit permettre de jouer des séquences à partir d'un mélange d'autres.

2.5.5 Définition formelle

J. Newsome *et al.* [20] ont proposé une définition formelle du rejeu. Ils définissent les éléments suivants :

– un ensemble d'états du processus S ;

- un état $s \in S$ contient toutes les informations systèmes à propos du processus : pointeur sur l'instruction suivante, valeurs des registres, de la mémoire...
- un ensemble d'entrées sur l'interface réseau c'est-à-dire les messages possibles I ;
- $P : S \times I \rightarrow S$ représente l'exécution d'un programme P à partir d'un état $s \in S$ et d'une entrée $i \in I$ résultant dans un état final $\sigma \in S$. Cet état final correspond aussi à la sortie sur le réseau c'est-à-dire aux messages envoyés ;
- un ensemble de postconditions Q dont l'intérêt va être détaillé ci-dessous.

Supposons qu'un processus sur la machine A exécute le programme P à partir de l'état initial s_a. i_a lui est alors transmis et son état devient σ_a tel qu'il existe $q \in Q$ avec $q(\sigma_a) = Vrai$ où $Q = \{B|B : S \rightarrow \{Vrai, Faux\}\}$. Cela correspond à trouver une condition vraie après que l'hôte A soit passé dans le nouvel état. Considérons maitenant une deuxième machine B exécutant le même programme P auquel on transmet la même entrée i_a, le rejeu réussira si l'état final de B, σ_b, satisfait la même postcondition que celle de σ_a, c'est-à-dire $q(\sigma_b) = Vrai$. D'après les indications des sections précédentes, il est clair qu'il est très peu probable d'obtenir ce résultat car, en règle général, l'état initial de B n'est pas le même que A (adresse IP, cookie...) et ainsi $q(P(s_b, i_a)) = Faux$. De manière formelle, le problème du rejeu consiste donc à trouver i_b tel que $q(P(s_b, i_b)) = Vrai$ avec $q(\sigma_a) = Vrai$. Les postconditions sont donc essentielles dans ce problème et sont à définir selon les cas d'applications. La fonction $\Phi : S \rightarrow Q$ génère une postcondition à partir d'un état d'entrée. La plus basique est la suivante où la postcondition est vraie pour des états totalement identiques :

$$\Phi(s) = (q(x) : s == x?)$$

Il ne faut pas oublier qu'un état regroupe tous les éléments du système et une telle fonction est inapplicable. A l'inverse, une définition plus générale imposerait que seule la sortie (le ou les messages envoyés) soit la même :

$$\Phi(s) = (q(x) : sortie(s) == sortie(x)?)$$

Enfin, le rejeu sert aussi à tester et vérifier la découverte d'une vulnérabilité en réussissant à reproduire l'attaque. Supposons p la propriété transgressée dans s, elle doit alors l'être dans x.

$$\Phi(s) = (q(x) : (s \Rightarrow \neg p) \Rightarrow (x \Rightarrow \neg p)?)$$

Grâce à ce modèle, J. Newsome *et al.* ont implanté *Replayer* [20], un outil de rejeu basé sur un haut niveau d'abstraction. Une des suppositions fondamentales est d'avoir accès au programme utilisant le protocole et pas seulement des traces d'exécution. *Replayer* commence alors par désassembler le programme pour le traduire en GCL (Guarded Command Language) [32], un langage de haut niveau basé sur la logique. Le programme s'écrit alors sous forme de formules pour identifier quelles sont les valeurs attendues pour produire une sortie identique. Ces valeurs peuvent être fixées, provenir du système ou être directement liées à une entrée utilisateur.

2.6 Bilan

Ce chapitre a mis en évidence les nombreuses applications de la rétro-ingénierie de protocole ainsi que ses nombreuses difficultés. Deux types majeurs d'informations peuvent être découverts :

- les différents types de messages qui symbolisent les genres principaux des opérations réalisables ainsi que les réponses associées à un protocole donné ;
- la syntaxe des messages qui, en plus du type des messages, met en avant les différents champs qui le composent et qui peuvent être vus comme les options du message.

Approches	Cas d'utilisation	Avantages	Inconvénients
Analyse des traces réseaux	– installation interdite ou risquée d'un logiciel utilisant le protocole – captures de traces réelles possibles	– acquisition / installation facultative d'un logiciel → moins de risques, moins coûteux – captures d'utilisations réelles souvent assez complètes → pas besoin de tester manuellement toute les fonctionnalités de l'application	extraction de la syntaxe difficile et souvent partielle
Analyse par tainting	installation possible d'un logiciel utilisant le protocole + instrumentation d'une machine	extraction plus complète (sémantique, structure), praticable avec des messages chiffrés	– coût plus elevé : acquisition/installation/tests d'un logiciel utilisant le protocole + instrumentation de la machine – plus risqué car comportement non connu à l'avance

TABLE 2.1 – Techniques de rétro-ingénierie de protocoles

Détecter le type des messages est une étape essentielle pour avoir un aperçu général du fonctionnement du protocole. La machine à états est d'ailleurs souvent induite à partir de ces derniers. La syntaxe introduit une granularité plus fine qui devient intéressante lorsque l'on souhaite utiliser le protocole en tant que tel. Cependant, les méthodes de rejeu montrent que pour mimer une application réelle, découvrir l'ensemble de la syntaxe n'est pas forcément nécessaire.

Par ailleurs, il existe aussi deux types de données à partir desquelles sont inférées ces informations :

– les traces réseaux ;
– les traces d'exécutions systèmes.

Comme le montre le tableau récapitulatif 2.1, le deuxième type a permis l'élaboration des outils les plus performants mais requiert d'avoir une application implantant le protocole étudié tournant sur une machine spécialement instrumentée. Dans le cas de l'étude d'un malware, cette approche est plus difficilement réalisable car le malware tentera de détecter une telle instrumentation pour l'induire en erreur. De plus, exécuter un malware comporte évidemment des risques.

Récupérer des traces réseaux est plus facile et beaucoup plus discret. Le plus simple consiste à écouter le trafic d'une machine infectée ce qui pose une fois de plus le problème d'installer et d'exécuter un malware. Cependant, il est possible d'imaginer qu'une fois qu'une machine compromise est détectée, le trafic est rapidement capturé avant sa désinfection. De même un opérateur a la capacité de récupérer ce type de trafic sans devoir installer le malware puisque celui-ci peut déjà sévir sur les machines de ses clients.

3

Fingerprinting

Sommaire

3.1 Introduction 63
3.2 Méthodes de fingerprinting 64
3.2.1 Fingerprinting passif 64
3.2.2 Fingerprinting actif 65
3.2.3 Fingerprinting semi-passif 66
3.3 Applications 66
3.3.1 Systèmes d'exploitation 67
3.3.2 Protocole 69
3.3.3 Équipement, pile protocolaire 75
3.3.4 Comptage 77
3.4 Bilan 78

3.1 Introduction

Le *fingerprinting* est un terme anglais difficilement traduisible dans le contexte des réseaux. Une traduction parfois rencontrée dans la littérature est *prise d'empreinte*. Effectivement, le fingerprinting vise à récupérer l'empreinte, aussi dénommée fingerprint ou signature, d'une donnée pour retrouver l'identité correspondante.

Une fois de plus, ce domaine n'est pas purement informatique comme le montre la racine du mot relatif à la prise d'empreintes digitales pour identifier une personne. En ce qui concerne l'informatique, le fingerprinting n'est pas cantonné aux réseaux puisque ce concept fut appliqué à différentes sources de données telles que des documents quelconques pour identifier leur provenance et surtout contrôler leur diffusion. Un exemple courant est l'adjonction d'une signature spécifique dans des contenus multimédias protégés par les droits d'auteurs [35] pour vérifier la légalité d'une copie.

Bien entendu, ce chapitre ne couvre que le fingerprinting dans le cadre des réseaux informatiques, domaine dans lequel de nombreux travaux ont été réalisés. Dans ce cadre, la définition suivante est donnée :

Définition 12 : *Le fingerprinting est la capacité à identifier une machine à distance.*

	Actif	Passif
Système d'exploitation	[41, 6, 21, 12, 14, 1, 7, 22]	[40, 2, 12, 37, 29, 22]
Protocole, Application	[6, 5]	[34, 38, 31, 32, 10, 36]
Équipement	[13, 4, 23, 22]	[14, 4, 24, 8, 22]
Comptage		[39, 30, 9]

TABLE 3.1 – Les approches traditionnelles de fingerprinting

Cette définition est volontairement vague car de nombreuses techniques de fingerprinting existent et l'identification d'une machine peut se révéler sous différentes formes selon le contexte dont les principales sont :

- identification unique (comptage du nombre de machines),
- détection du système d'exploitation,
- détermination du ou des protocoles utilisés,
- identification de la version du logiciel ou de l'équipement utilisé.

En outre, le fingerprinting repose sur la définition d'une signature :

Définition 13 : *La signature d'une machine est la représentation d'un sous-ensemble de données qu'elle a émises ou qu'elle a reçues et qui permettent de l'identifier.*

L'identification d'une machine peut prendre différentes formes comme auparavant. De plus, les données peuvent correspondre intégralement ou en une partie à un paquet, ou à un ensemble de paquets. En outre, ce chapitre va mettre en évidence la variété des représentations correspondantes : valeurs fixes, probabilités, chaînes de Markov, arbres syntaxiques, graphes... Le but de ce chapitre est donc de préciser ces définitions selon les cas étudiés et les différents travaux en essayant d'apporter une vue globale à ce domaine.

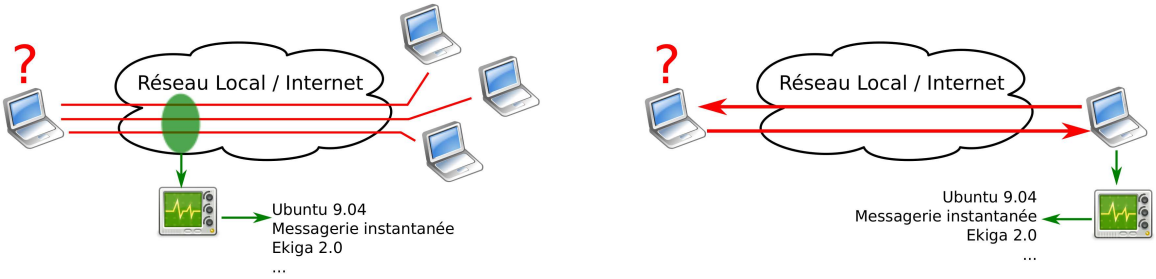
Le rapport entre le fingerprinting et les malwares se révèle sous différentes formes. Premièrement, un malware peut s'attaquer à un type de machines bien précis que le fingerprinting peut chercher à identifier pour les protéger ensuite. Deuxièmement, le fingerprinting peut être appliqué à un malware afin de l'identifier plus précisement. Identifier la version d'un malware se révèle intéressant lors de la mise en place de protections contre celui-ci ou tout simplement pour l'étudier. Par exemple, la présence d'une version prédominante est significative d'une meilleure robustesse de celle-ci.

3.2 Méthodes de fingerprinting

Deux classes de fingerprinting sont généralement distinguées : active et passive. Un récapitulatif des publications s'appuyant sur des méthodes de ce type est donné dans le tableau 3.1. Dans tous les cas, une machine dite d'observation capturant du trafic est nécessaire.

3.2.1 Fingerprinting passif

Le fingerprinting passif consiste à capturer des échanges de messages sur le réseau puis à identifier les machines impliquées comme le montre la figure 3.1(a). Cette technique est non intrusive puisqu'aucune interaction additionnelle n'est nécessaire. Cette "discrétion" est son principal avantage car elle évite à celui qui réalise cette opération d'être démasqué voire, par la suite, attaqué. Cependant, les techniques



(a) Approche passive (b) Approche active

FIGURE 3.1 – Les deux approches traditionelles de fingerprinting

passives sont exclusivement dépendantes du trafic observé et si une machine n'envoie que quelques paquets de temps en temps, son identification risque d'être très difficile. Il est alors souvent nécessaire de capturer le trafic pendant un temps relativement long pour obtenir un nombre de paquets suffisants à analyser. De même, si le système de fingerprinting est établi sur la base d'un modèle de communication bien précis, c'est-à-dire celui observé pendant la capture, et que celui-ci évolue avec le temps, il ne pourra pas forcer les machines à réexécuter le modèle précédent car il ne doit pas interagir.

3.2.2 Fingerprinting actif

Contrairement au fingerprinting passif, les méthodes actives requièrent d'interagir avec la machine à identifier pour ensuite pouvoir analyser les réponses émises par celle-ci comme le montre la figure 3.1(b). La difficulté de telles approches réside donc principalement dans la détermination des requêtes à envoyer et dans l'analyse des réponses obtenues. Une technique ne sera efficace que si pour une même requête, les réponses obtenues de différentes machines diffèrent. Bien entendu, une seule requête ne permet pas d'avoir une identification précise mais permet de réduire l'espace de recherche et ainsi de choisir plus précisement les requêtes suivantes.

L'inconvénient majeur de cette approche est l'obligation d'envoyer des messages, ce qui implique la nécessité de comprendre le protocole et de pouvoir générer des messages acceptables, même si l'introduction volontaire d'erreurs dans la définition du message est une méthode de fingerprinting possible. En effet, les réactions des hôtes face à de tels messages sont de bonnes indications sur leur identification mais cela nécessite quand même de respecter en grande partie la grammaire du protocole pour ne jouer que sur une partie de celle-ci.

Bien entendu, l'injection de messages rend la machine qui fait le fingerprinting détectable. Ainsi, le challenge principal est la réduction du nombre de messages à envoyer tout en gardant une bonne précision. Pour cela, chaque requête envoyée doit identifier très clairement le plus de machines possibles.

Le tableau 3.1 met en évidence le peu de ce type de fingerprinting pour identifier un protocole. En effet, pour utiliser ce type de méthode, la connaissance du protocole employé est obligatoire alors que dans le cas présent il doit justement être identifié. En réalité, ces méthodes envoient un grand nombre de requêtes de différents protocoles pour déterminer laquelle est acceptée par l'entité à identifier, ce qui les rend très facilement détectable.

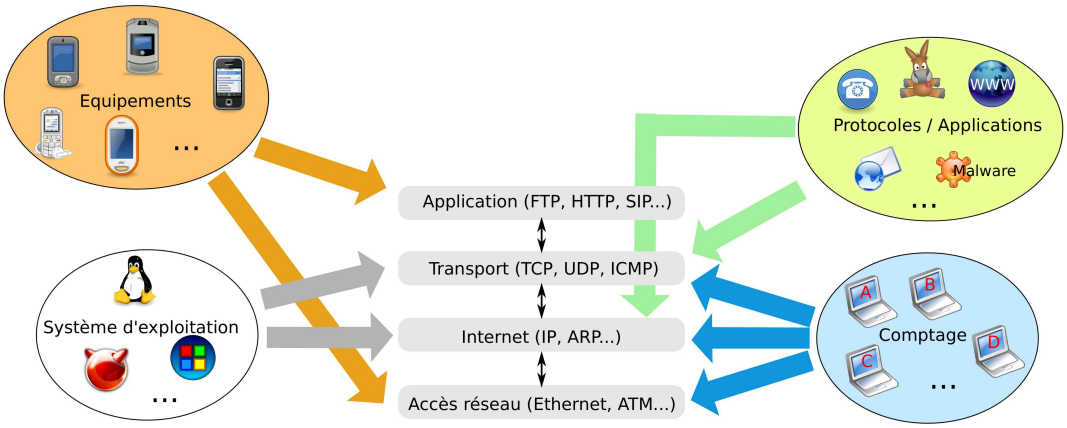


FIGURE 3.2 – Différents types de fingerprinting à différents niveaux de la pile TCP/IP (Beaucoup de méthodes se basent également sur les délais entre les messages qui dépendent en partie de la couche *Accès Réseau*)

3.2.3 Fingerprinting semi-passif

A côté de ces approches traditionnelles, Kohno *et al.* évoquent une technique intermédiaire qu'ils qualifient de semi-passive [30]. Dans ce cas, la machine d'observation peut interagir mais seulement après avoir été elle-même contactée par la machine à identifier. Cette approche est reprise dans [16]. Pour pouvoir mettre en œuvre ce genre de techniques, il est préférable d'offrir un service qui pourrait intéresser les machines à identifier. Cependant, un autre procédé vise à s'immiscer au milieu de la communication de deux machines pour pouvoir modifier les messages transmis (attaque *man in the middle*, opérateur réseau).

3.3 Applications

Jusqu'à présent, aucune définition de l'identification d'une machine n'a été donnée. Selon le but visé, celle-ci peut prendre différentes formes qui vont être introduites dans la suite de cette section. La figure 3.2 représente les grands types de fingerprinting et le niveau d'information que chacun d'eux utilise. Ainsi, les caractéristiques que l'on souhaite identifier sont de diverses natures :

- système d'exploitation ;
- protocoles utilisés ;
- logiciel ou équipement utilisé ;
- un "identifiant" unique par machine.

Il est important de bien comprendre que les flèches de la figure 3.2 indiquent quelles sont les informations extraites par les différentes techniques et non ce qu'elles cherchent à déduire.

3.3.1 Systèmes d'exploitation

L'identification du système d'exploitation d'une machine distante se révèle très important pour évaluer les risques d'attaques car certains systèmes sont plus vulnérables et plus visés par celles-ci. Cette approche est d'autant plus intéressante lorsqu'elle est capable de préciser la version exacte du système d'exploitation ainsi que les mises à jour installées. Bien entendu, c'est aussi un formidable outil de repérage de victimes potentielles pour les attaquants. De plus, le fingerprinting permet de déceler une machine essayant de se faire passer pour un autre système tel qu'un honeypot. Enfin, dans le cas de systèmes propriétaires, il peut aider à repérer les machines l'utilisant et qui ne sont apparemment pas censées avoir obtenu une licence d'utilisation.

Inférence de règles à partir d'observations

Les systèmes d'exploitation fournissent les couches réseaux nécessaires aux applications pour communiquer. Leurs différences d'implantation sont donc significatives du système d'exploitation utilisé. Comer *et al.* [41] proposent alors de découvrir ce dernier en envoyant des messages bien précis pour observer le comportement en résultant. En fait, ce sont les couches réseaux directement gérées par le système d'exploitation comme illustré sur la figure 3.2. Par exemple, la complexité et l'évolution du standard de TCP [12] a occasionné des différences d'implantation. Deux types de différences sont recherchées :

- celles induites par la marge de liberté de la spécification du protocole ;
- celles émanant du non respect du protocole.

Dans le premier cas, la spécification impose, par exemple, seulement un minimum de 10 ms pour le timer de retransmission [3]. Pour explorer le second cas, la méthode cherche à générer des cas très particuliers comme lorsque la fenêtre de réception décroît jusqu'à un, *Solaris 2.1* se trompe alors dans les numéros de séquences TCP. Le fingerprinting de systèmes d'exploitation se base donc généralement sur le contenu des entêtes des protocoles des couches *Internet* et *Transport* du modèle TCP/IP.

La première approche passive apparue en 1997 [40] souligne d'importantes divergences dans l'implantation du contrôle de congestion malgré l'apparition de nombreux standards. Pour rappel, l'émetteur a une fenêtre d'émission représentant le nombre de paquets pouvant être émis sans acquittement mais qui est également limitée par une fenêtre de congestion comme le montre la figure 3.3. Paxson *et al.* révèlent que les mécanismes habituellement préconisés pour éviter la congestion et définir cette fenêtre ne sont pas forcément totalement et identiquement implantés (slow start, congestion avoidance... [12]). De plus, selon les versions des piles TCP, la retransmission des paquets est gérée différemment de même que la gestion des segments TCP arrivant dans le mauvais ordre.

Actuellement, les outils *NMAP* [6] et *p0f* [2] sont les plus répandus dans le domaine de la découverte des systèmes d'exploitation. p0f est capable de le faire de manière passive. Les auteurs distinguent quatre classes de machines selon le rapport qui existe entre celles-ci et une autre machine dans le réseau :

- celles qui se connectent à la machine, les connexions entrantes (paquets SYN) ;
- celles sur lesquelles la machine se connecte, les connexions sortantes (retour d'un paquet SYN+ACK) ;
- les machines sur lesquelles on ne peut pas se connecter (retour d'un paquet RST) ;
- toutes les autres machines dont on peut observer les communications ;

Pour chaque paquet, *p0f* observe les différentes valeurs des champs et les compare à une base de données de fingerprints car selon le système d'exploitation, elles sont initialisées à différentes valeurs. A titre d'exemple, p0f récupère les informations suivantes pour les paquets SYN :

```
win:TTL:MSS:DF:wscale:sackOK:nop:taille
```

3. Un paquet est considéré comme perdu lorsqu'aucun acquittement n'est reçu après un certain délai

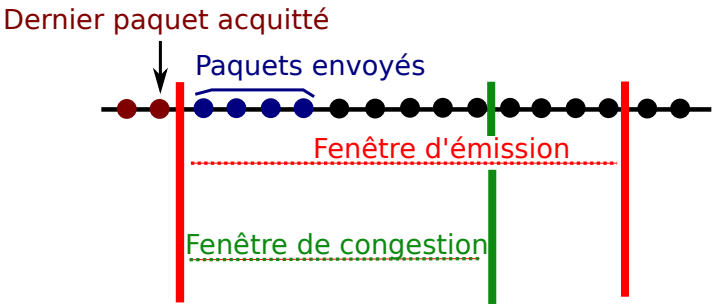


FIGURE 3.3 – Fenêtres TCP de l'émetteur

où *win* représente la taille de la fenêtre de réception, *TTL* le time-to-live... Sans entrer dans les détails, la signature *5840 :64 :1460 :1 :0 :1 :1 :60* correspond à un noyau Linux 2.4. Enfin, *p0f* a beaucoup évolué et propose aujourd'hui des fonctions plus avancées comme la découverte de pare-feux ou de routeurs NAT.

NMAP [6] utilise quant à lui des méthodes actives et inclut plus de fonctionnalités tel que le balayage de ports. Dans le cadre du fingerprinting, *NMAP* envoie 16 requêtes spécifiques (TCP, UDP et ICMP). Les réponses reçues sont alors analysées individuellement mais aussi conjointement car les valeurs de certains champs TCP peuvent par exemple être incrémentées linéairement pour certains systèmes. Ces tests (environ une trentaine dans la version 5) sont effectués aussi bien sur les protocoles de transport que sur IP par rapport à une base de données de fingerprints. *Xprobe* [1] cherche quant à lui à engendrer moins de trafic et repose essentiellement sur des requêtes ICMP.

NMAP est aujourd'hui un outil très répandu et complète parfaitement *p0f* dans le cas où la machine à identifier ne communique pas puisqu'il se charge de sonder ce type de machines. Cependant, il est aisé de repérer un utilisateur de *NMAP* d'autant plus que celui-ci est souvent obligé en premier lieu d'effectuer un balayage de ports. C'est dans cette optique que [21] mesure l'efficacité des différents tests selon différents systèmes d'exploitation et conclut que seulement trois ou quatre requêtes sont suffisantes pour obtenir un bon taux de découverte. De plus, seuls des paquets valides établissant le 3-way handshake sont générés.

C'est aussi la technique mise en œuvre par *SinFP* [12] qui n'utilise que trois requêtes pour identifier un système d'exploitation. Son intérêt est d'avoir une base de signatures beaucoup plus petite. En effet, des masques de déformation, définis par des expressions régulières, sont appliqués sur les signatures car pour un même système d'exploitation, il peut y avoir certaines variations dûes à l'environnement réseau. SinFP suggère aussi d'utiliser ces même signatures pour du fingerprinting passif en les adaptant, c'est-à-dire en les transformant, car dans ce cas, capturer exactement les bonnes requêtes n'est pas évident.

Apprentissage

Les systèmes précédents reposent sur un ensemble de signatures dont la mise au point est une tâche ardue. Ainsi, FIG [14] génère automatiquement les signatures adéquates. Pour fonctionner, cette méthode nécessite une phase de test où de nombreuses requêtes sont envoyées, elles constituent l'ensemble initial des requêtes et doit représenter de manière assez exhaustive les différentes possibilités. Toutefois, des tests exhaustifs seraient beaucoup trop longs et la plupart d'entre eux inutiles. Les auteurs proposent donc de générer ces requêtes en modifiant uniquement les champs sémantiquement riches (comme les flags TCP par exemple). Ensuite, grâce aux réponses induites et à des techniques d'apprentissage automatique, les requêtes permettant de distinguer efficacement les différents systèmes d'exploitation sont extraites.

Ainsi, à partir de 305 requêtes initiales possibles, FIG en déduit que seuls 66 sont propices.

L'approche passive énoncée dans [37] se base sur une classification naïve bayésienne [17]. Cette technique s'appuie sur les probabilités conditionnelles de manière à déterminer la probabilité d'avoir la classe C_k pour un vecteur v : $P(C_k|v)$. Sachant que v est composé de différentes valeurs caractéristiques du paquet (time-to-live, flag SYN...), cette probabilité se calcule en fonction de celles observées pendant la phase d'apprentissage :

$$P(C_k|v) = P(C_k) \times \Pi_j P(v_j|C_k)$$

La classe considérée comme juste est celle coïncidant avec la plus forte probabilité. Cette approche est dite *naïve* puisque les composantes des vecteurs caractéristiques sont supposées indépendantes ce qui n'est généralement pas le cas. Il est donc nécessaire dans un premier temps de calculer les valeurs $P(v_j|C_k)$ pendant l'apprentissage.

Les réseaux neuronaux sont une autre voie possible étudiée par Sarraute *et al.* [7]. Lors d'une phase d'entraînement, ils construisent un réseau multi-couche de neurones qui prend ensuite en entrée les résultats d'une requête sur une machine à tester. Leur première méthode repose sur des requêtes RPC (Remote Procedure Call) [9] qui permettent de découvrir les services distants proposés par une machine Windows. Cette technique permet d'identifier précisement la version de Windows mais aussi une édition précise (familiale, professionnelle) ainsi que les mises à jours principales (service packs). De plus, un réseau de neurones est bâti grâce aux signatures de *NMAP* où chaque réponse, voire type de réponse reçue (valeurs des champs TCP), est associée à un neurone. Ainsi, il n'y a plus besoin de définir des règles précises (basées sur les valeurs des différents champs) puisque le réseau de neurones se charge d'apprendre une fonction équivalente. Les travaux dans [29] mettent aussi en œuvre les réseaux neuronaux mais se basent pour sa part sur les signatures de *p0f*.

3.3.2 Protocole

Identifier un protocole est un autre aspect du fingerprinting visant des buts variés. Par exemple, il est intéressant pour des raisons de sécurité de repérer l'installation et l'utilisation de logiciels prohibés au sein d'une entreprise. En effet, Skype fut banni des réseaux universitaires [3] malgré son utilisation croissante, principalement du fait de ses aspects P2P et propriétaires qui empêchaient de mesurer réellement les risques de sécurité. Sans forcément interdire l'utilisation d'un logiciel et selon les besoins à un moment donné, il peut s'avérer opportun de limiter le trafic annexe par rapport au trafic principal. Par exemple, une entreprise de vente en ligne doit en priorité privilégier son trafic Web plutôt que le trafic P2P ou les messageries instantanées de ses employés. Ainsi, le fingerprinting d'applications ou de protocoles sont des termes équivalents.

Dans cette partie, le fingerprinting pur (classification des protocoles) sera distingué de la classification de trafic qui est corrélée mais qui présente une granularité moins fine. A titre d'exemple, le tableau 3.2 montre qu'une catégorie de trafic regroupe plusieurs protocoles. Cependant, la dernière catégorie, *malware*, peut contenir des protocoles standards tels qu'introduits dans le chapitre 1 et un protocole de messagerie instantanée comme IRC peut donc se retrouver également dans cette catégorie. De plus, la granularité de ces catégories peut varier selon l'utilisation visée, de la plus simple (bénin / malware) à la plus compliquée (chaque protocole comme dans la section suivante).

De manière générale, le fingerprinting de protocoles donne un aperçu des services employés et de leurs utilisateurs. Cette supervision permet ensuite de modifier le réseau pour mieux supporter la charge par exemple. Cela permet aussi d'établir simplement des statistiques pour savoir si un service est encore utilisé ou si les ressources associées peuvent être libérées (réseau en partie dédié, abonnement...).

Catégorie de trafic	Exemples de protocoles
Web	http
P2P	Gnutella, BitTorrent, Kazaa, FastTrack, Kademelia/Kad, eDonkey
Email	pop3, imap, smtp
Services réseaux	dns, snmp, ldap, ntp, netbios
Messagerie instantanée	IRC, MSN, AIM, Jabber
Jeux	HalfLife, Age of Empire
Base de données	sqlnet, oracle
Streaming	mms, quicktime, shoutcast
Malware	vers, attaques, botnet
Transfert de grands volumes (*Bulk*)	ftp

TABLE 3.2 – Granularité de l'identification de trafic

Bien que la IANA [3] a défini des ports standards pour les applications, il est très facile d'utiliser un port classique tel que celui pour le Web (80) pour transmettre tout autre type de données. De même certains protocoles se partagent les mêmes ports et de nombreuses applications assignent un port de façon dynamique à chaque utilisation.

Pour distinguer les différents trafics, il existe deux types de méthodes :

- les méthodes par paquet qui tentent de classer chaque paquet indépendamment [34, 38] ;
- les méthodes qui traitent un ensemble de paquets à la fois, comme par exemple ceux qui forment un flux [34, 36, 31, 18, 32, 25, 33, 10, 19, 20, 22, 26, 27, 18, 17, 28, 2].

Dans le premier cas, il est possible d'extraire le contenu du paquet pour l'analyser [38]. Dans le second cas, il est aussi possible de prélever le contenu du flux ou tout du moins partiellement [34, 22, 18]. Cependant, de nombreuses caractéristiques plus générales sont étudiées [36, 31, 18, 32, 25, 10, 19, 26, 27, 17, 28, 2] :

- taille d'un paquet ;
- temps entre deux paquets ;
- taille du flux dans son ensemble ;
- entêtes des paquets ;
- durée du flux ;
- ports utilisés ;
- etc...

Plusieurs de ces caractéristiques sont dérivées selon leurs valeurs statistiques : moyenne, écart type, minimum, maximum...

Pour rappel, un flux se caractérise au minimum par une adresse IP source et destination. Cependant cette définition se retrouve également au niveau des protocoles de transport et un flux se caractérise alors par le 5-uplet suivant : <protocole,adresse IP source, adresse IP destination, port source, port destination>. Concernant TCP, un flux est facilement délimité grâce aux paquets de contrôle permettant d'ouvrir et de fermer une connexion. Dans le cas d'UDP, la spécification d'un délai maximal entre deux paquets est obligatoire pour repérer la fin d'un flux. Cette méthode est aussi appliquée avec TCP lorsqu'une connexion ne s'est pas terminée normalement.

Classification de trafic

Moore *et al.* décidèrent d'inspecter plus en détails les flux de données [34]. Ainsi la méthode proposée consiste à analyser chaque communication avec un degré de détail variant du plus simple jusqu'au plus compliqué quand c'est nécessaire. Par exemple, la suite de tests cherche d'abord à déterminer s'il

existe une signature connue concordant avec le premier paquet. Dans le cas contraire, il faut rechercher une signature qui correspond aux premiers octets du flux TCP voire chercher dans l'ensemble du contenu (payload). Cette étude met en évidence l'obligation de rechercher l'information dans le contenu même du paquet pour réussir à différencier certaines applications tout en soulignant l'impact sur la complexité et le besoin d'avoir une base de signatures.

De ce fait, des techniques d'apprentissage ou de classification supervisées sont apparues car elles ne nécessitent pas d'élaborer manuellement des signatures. Elles apprennent directement à identifier le trafic après avoir été entraînées grâce à un jeu de données où le trafic est supposé connu. La contrainte essentielle réside donc dans la complétude de ce jeu de données ainsi que dans sa justesse quant à l'étiquetage des flux ce qui requière une intervention manuelle préalable.

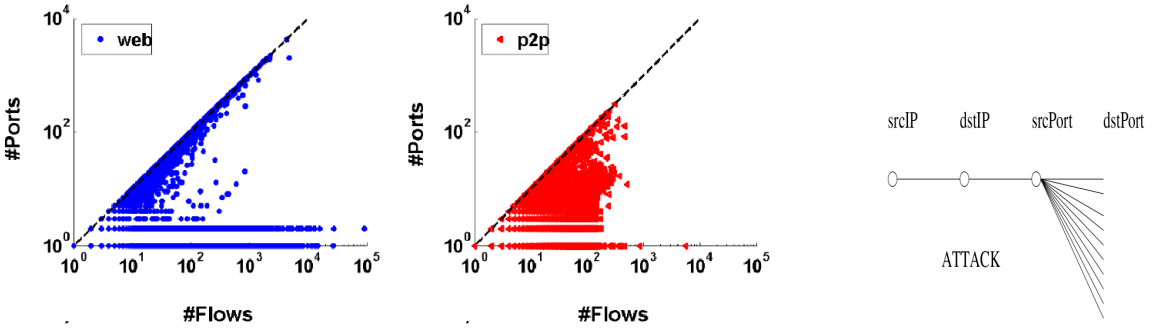
Moore *et al.* introduisent l'utilisation de classification naïve bayesienne dans [31] se basant exclusivement sur les caractéristiques des flux. L'approche est améliorée par l'usage de deux mécanismes. Le premier estime chaque probabilité conditionnelle $P(c|y)$ d'avoir la classe c selon le discriminant y grâce à une distribution gaussienne pour lisser celle-ci. Le second, dérivé de [12], sélectionne les meilleurs discriminants, c'est-à-dire les champs des en-têtes à analyser. L'objectif est de se cantonner à des discriminants qui séparent bien les différentes classes et, en même temps, ne pas avoir deux discriminants dont le résultat est redondant. Cette sélection se fait via des mesures entropiques qui vérifient qu'un discriminant ajoute de l'information, c'est-à-dire qu'il permet de mieux distinguer deux classes différentes, et que deux discriminants sont suffisamment indépendants. L'étude [18] tire parti des caractéristiques classiques des flux et applique la méthode des K-means [4].

Les méthodes non supervisées, sans phase d'apprentissage, ont aussi vu le jour et classent les différents flux sans savoir exactement à quoi ils correspondent. Ce fut notamment le cas de [36] qui visait à discerner quatre grandes classes de trafic en utilisant différentes caractéristiques parmis lesquelles : taille moyenne d'un paquet, durée d'un flux, temps entre deux paquets, nombre de paquets... L'approche est quant à elle validée grâce à deux techniques de classification répandues dont celle des plus proches voisins et les résultats furent encourageants. Ainsi, de nombreuses recherches ont suivi en utilisant d'autres caractéristiques, d'autres méthodes ou aussi en essayant d'affiner les différentes catégories. Comme [31], [32] s'appuie sur un classificateur bayésien mais qui se différencie par son aspect non supervisé et donc sa capacité à apprendre automatiquement les classes du trafic.

En réalité, beaucoup d'algorithmes de classification furent employés et [25] met en exergue la multitude de caractéristiques possibles ainsi que des performances similaires pour différents algorithmes. De ce fait, le papier se focalise sur les différences de temps de calcul entre eux et souligne l'intérêt de réduire le nombre de caractéristiques qui n'influent que très peu sur les performances.

BLINC [33] se singularise grâce à l'introduction de nouvelles caractéristiques basées sur les relations inter-machines permettant de tester ensuite différentes heuristiques afin de classer les flux. Ainsi, la première métrique calcule la popularité d'un hôte, c'est-à-dire le nombre de machines avec lesquelles il communique. Par exemple, un protocole P2P implique d'interagir avec de nombreux participants pendant un court intervalle de temps. Par ailleurs, les groupes de machines interagissant entre-eux sont définis comme des communautés telles que les communautés de machines de même domaine qui implique, la plupart du temps, le même service offert. Aussi, le nombre de ports associés à chaque machine permet d'identifier le rôle de celle-ci comme le montre la figure 3.4(a) où un point représente chaque IP selon le nombre de flux et le nombre de ports. Comme illustré dans le cas du P2P, les points sont répartis car tout le monde joue le rôle de serveur et de client alors que pour le Web, certains sont les clients et utilisent beaucoup de ports différents (points proches de la diagonale) et d'autres sont les serveurs utilisant un ou plusieurs ports (lignes horizontales). Enfin, une base de données de petits graphes basés sur les adresses IP et les ports aide ensuite à identifier le comportement d'une machine. A titre d'exemple, un balayage de port d'une machine se schématise par le graphe représenté sur la figure 3.4(b).

Bien que [10] tire aussi parti de diverses techniques d'apprentissage, son principal intérêt est l'utili-



(a) Différence entre un serveur Web et un client P2P au niveau des ports utilisés

(b) Graphe d'interaction représentant un balayage de ports

FIGURE 3.4 – Exemples de caractéristiques utilisées dans BLINC

sation d'un nouveau type de données à étudier, les *Application Rounds*, aussi bien différents des paquets que des flux. Ils sont constitués de deux séries de messages, ceux envoyés et ceux reçus juste après, et représentent en fait les différentes négociations intervenant généralement au commencement d'une communication. Chaque série de messages unidirectionnels s'appelle un *Talk*. Sur ces différents ensembles, plusieurs métriques (nombre de messages, délais, tailles...) servent d'entrée aux algorithmes d'apprentissage.

Alors que les méthodes supervisées citées précédemment ont l'avantage de pouvoir préciser exactement la classe d'un trafic et pas seulement de regrouper les flux entre eux, Erman *et al.* remarquent que celles-ci sont fortement contraintes [19] :

- obtenir un jeu d'apprentissage avec beaucoup de flux étiquetés est difficile ;
- certaines classes ne sont pas forcément connues à priori et d'autres types de trafic peuvent apparaître.

C'est pourquoi, la méthode énoncée dans cet article est dite non supervisée puisqu'elle considère une phase d'apprentissage où des groupes de flux sont créés. Ces derniers peuvent contenir des flux étiquetés ou non et le groupe lui même se voit étiqueté avec le type de flux le plus fréquent. Cependant, un groupe peut ne contenir aucun flux connu. Ainsi, lors de la phase de test, chaque nouveau flux est associé au cluster le plus proche et peut donc être considéré comme du trafic inconnu. De plus, les auteurs ont conçu un système de classification grâce à différents jalons qui sont spécifiés par le nombre de paquets transmis dans le flux (2,4,8,16 ou 32).

Au vu du nombre important de méthodes et de caractéristiques employées, [20] combine différentes méthodes. Enfin, [11] est un bon état de l'art des différentes techniques car en plus d'évaluer leurs performances de classification, il mesure la complexité de chacune d'elle.

Classification de protocoles

Les démarches précédentes essayent de classer le trafic selon une granularité plus ou moins fine. La sous-section courante se focalise essentiellement sur la classification de protocoles qui n'a pas été présentée avant. Cependant, beaucoup des techniques mises en œuvre et des caractéristiques des flux utilisées sont semblables.

Tout d'abord, les techniques actives telles que NMAP [6] ou AMAP [5] se connecte à un port ouvert (détecté souvent grâce à un balayage de ports) avant d'envoyer des requêtes de différents protocoles pour déterminer celle qui est comprise par la machine à identifier indiquant ainsi le protocole utilisé. NMAP

[6] peut également n'ouvrir qu'une connexion TCP sans envoyer de requêtes. En effet, dans certains cas il peut arriver qu'un serveur se présente lui même à son correspondant grâce à un message d'accueil. Dans ce cas, il est parfois possible d'identifier le logiciel utilisé.

Les premières approches passives s'efforçaient, comme dans la section précédente, d'établir manuellement des signatures. Par exemple, [38] se focalise sur la détection des flux P2P de différents types (Gnutella, BitTorrent...) dans les réseaux très haut débit. La technique présentée est efficace en terme de précision et rapidité grâce à des tests "simples" et ordonnés pour rapidement étiqueter un flux. La seule ombre au tableau est la nécessité d'une étape manuelle d'étude des protocoles pour identifier les formats correspondants et générer une signature.

En conséquence, établir des signatures est primordial mais est une tâche fastidieuse lorsqu'elle est manuelle. ACAS [22] est un système permettant de dériver automatiquement une signature pour chaque protocole étudié à partir d'un jeu d'apprentissage et d'une méthode de classification appropriée, telle qu'AdaBoost, [23] appliquée sur le contenu du paquet : plus de 99% des flux sont correctement identifiés grâce aux 64 premiers octets seulement, sans chercher à en extirper des informations précises (entêtes, champs...). De plus, les signatures sont relativement stables puisque six mois après, elles permettent d'identifier encore 94% du trafic.

Les travaux de la référence [26] emploient la méthode des K-means [4] en se basant uniquement sur la taille des cinq premiers messages et leurs directions pour distinguer les protocoles. En effet, les premiers paquets transmis lors d'une communication s'apparentent à la négociation de ses paramètres alors que les paquets suivants sont plutôt dépendants du comportement de l'utilisateur. Bien que la méthode des K-means demande de renseigner le nombre de clusters, les auteurs prônent une méthode non supervisée en fixant arbitrairement ce nombre à une valeur arbitrairement élevée. Ainsi, même si différents clusters partagent le même protocole, chacun d'eux restera consistant, c'est-à-dire ne contiendra qu'un seul type de flux d'autant plus que les caractéristiques de ce dernier peuvent varier selon les actions faites par l'utilisateur. Toutefois, en augmentant artificiellement le nombre de clusters et en autorisant plusieurs clusters de même type, la précision du classement se retrouve surévaluée puisque les clusters sont plus précis voire, dans le cas extrême, ne contiennent qu'un flux. De manière à affiner le choix du nombre de clusters, les travaux qui succédèrent dans [27] mettent en pratique le calcul d'un coefficient mesurant la qualité du regroupement pour éviter de trop éclater les clusters. Par ailleurs, l'utilisation des modèles de Markov [13] cachés, qui tient compte de l'ordre d'arrivée des messages, fut également testée tout en ramenant le nombre minimal de paquets à seulement quatre pour obtenir une bonne précision.

Bien que l'approche détaillée dans [18] nécessite une phase d'apprentissage, elle permet d'identifier un flux connu, comme vu auparavant, ou d'identifier un nouveau trafic encore inconnu. De la même manière que [22], seuls les 64 premiers octets de chaque flux sont utilisés dans le but de créer des clusters de sessions. D'après les auteurs, la distribution de ces octets caractérise suffisamment les protocoles mais son exploitation et son stockage ont besoin de beaucoup de ressources car il faut être capable de les comparer rapidement et aussi de les fusionner entre elles pendant la phase d'apprentissage. Ainsi plusieurs formalisations équivalentes sont suggérées :

- la première considère chaque distribution de manière indépendante et elles sont comparées grâce à l'entropie relative ;
- l'utilisation de processus de Markov qui établissent des graphes probabilistes entres les différentes valeurs des octets et une mesure dérivée de l'entropie relative pour comparer deux processus différents ;
- l'alignement de séquences [28] s'illustre une fois de plus ici et permet d'établir un graphe pondéré de sous-chaînes représentant les enchaînements possibles pour un protocole. Par exemple, la figure 3.5 schématise différentes possibilités de flux SSH [3] possibles où les nœuds les plus sombres sont ceux qui sont les plus courants.

Bien sûr chacune des méthodes précédentes est complétée par un mécanisme de regroupement dont les

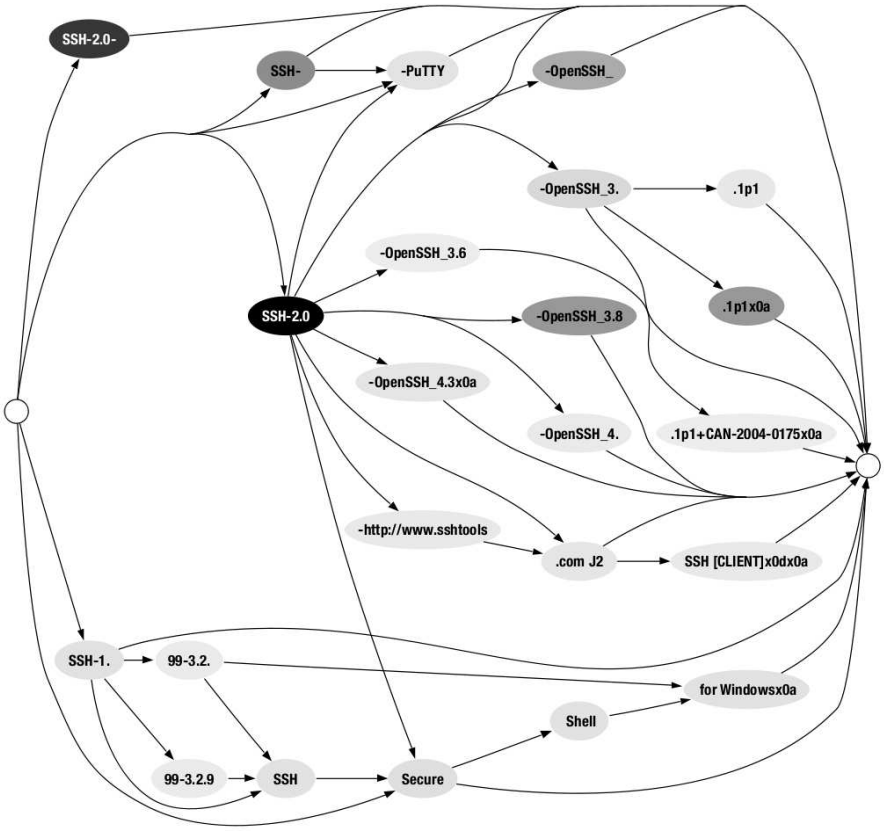
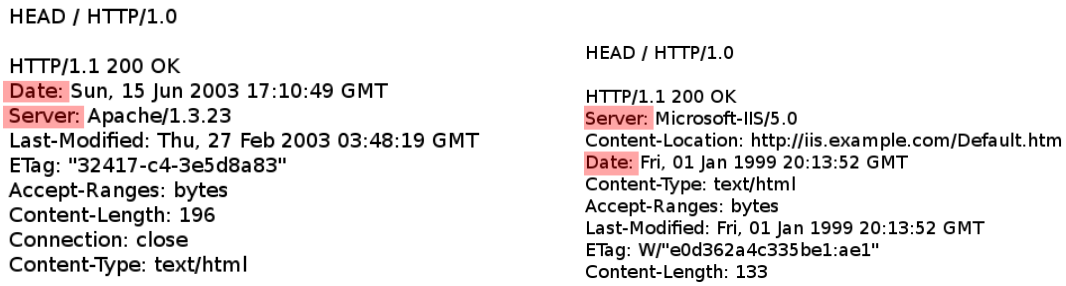


FIGURE 3.5 – Graphe de sous-chaînes possibles pour le protocole SSH

détails peuvent être consultés dans [18]. Pendant la phase d'apprentissage, des *cellules* sont créées et chacune d'elle regroupe les flux partageant le même protocole de transport, le même port et la même adresse de destination. Selon les auteurs, cette définition de cellule est suffisante pour regrouper les sessions d'un même protocole. De plus, tout flux partageant le même port de destination et la même machine source pendant un temps très court sont aussi regroupés dans la même cellule. Une fois cette première classification générale créée, les cellules sont représentées grâce aux différents formalismes précédents et comparées entre-elles de manière à les regrouper. Après cette phase d'apprentissage, chaque nouveau flux est alors comparé à chaque groupe de manière à déterminer sa *distance* vis à vis de chaque protocole pour lui en attribuer un. L'apparition d'un nouveau protocole est alors repérée si la distance minimale calculée est anormalement élevée.

La taille des paquets, notamment pour les premiers, est une statistique significative que Crotti *et al.* [17] complètent par le temps entre chaque paquet tout en exploitant leur ordre d'arrivée. Ainsi chaque flux unidirectionnel se définit comme une séquence ordonnée de paires $P_i = \{s_i, \Delta_i\}$ où s_i est la taille du i^e paquet et Δ_i le temps écoulé entre son arrivée et celle du suivant. Lors de la phase d'apprentissage, une densité de probabilités est calculée pour chaque paire P_i. L'ensemble de ces densités forme la signature du protocole. Ensuite, les distances entre la distribution d'un nouveau flux et chaque signature sont évaluées en fonction des valeurs P_i mais aussi en fonction d'un filtre gaussien qui permet de lisser les délais des paquets et ainsi de limiter les erreurs de classification dues aux variations du RTT dans le réseau. La deuxième innovation dans ce papier est la définition d'un seuil par protocole représentant la distance maximale autorisée pour détecter un nouveau protocole. Ainsi, il existe autant de seuils que de protocoles



(a) Apache 1.3.23 (b) IIS 5.0

FIGURE 3.6 – Différence d'ordonnancement des champs *Date* et *Server* dans les entêtes HTTP

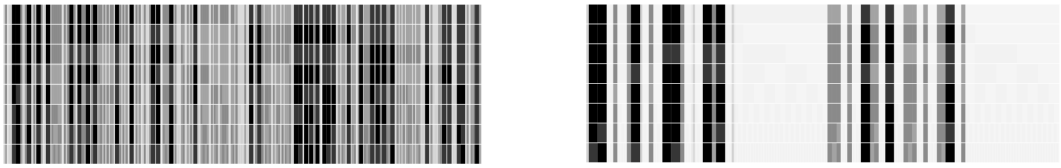
et ne sont pas fixés manuellement mais automatiquement grâce au même filtre gaussien appliqué sur les valeurs P_i de manière à calculer ensuite la moyenne globale et l'écart type.

Bien entendu, la définition de nouvelles caractéristiques fut aussi un autre axe de recherche, comme la taille du contenu (hors entêtes) ou la taille totale du flux à l'instar de [28] qui compare l'efficacité de différents algorithmes de clustering. Une autre possibilité est l'utilisation de chaînes de Markov à l'instar de [2] se basant sur les séquences de paquets de contrôles (SYN, ACK, SYN-ACK...). Ainsi, à partir d'un ensemble de flux d'un protocole, une chaîne de Markov est construite dont les états correspondent aux types de paquets de contrôles et les transitions à la probabilité d'enchaînement de ces derniers. Ensuite, une probabilité est calculée pour chaque nouveau flux en parcourant la chaîne de Markov selon les paquets rencontrés tout en multipliant les probabilités rencontrées.

3.3.3 Équipement, pile protocolaire

Étant donné un protocole, de nombreuses variantes d'implantations sont possibles. Par exemple, bien que le protocole HTTP soit défini dans le RFC [7], chaque navigateur utilise sa propre implantation (pile protocolaire). Ces différences peuvent être dues ou non à des erreurs de programmation ou des libertés permises par les spécifications (volontairement ou par manque de clarté). Le fingerprinting d'équipements ou de piles protocolaires a donc pour but d'identifier quelle version d'un protocole implanté est utilisée. Une version s'identifie par le nom du logiciel ainsi que la version utilisée. Cependant, certaines implantations sont spécifiques à des équipements (téléphone SIP [6] par exemple) dont le nom et la version identifient la version du protocole. En réalité, les fingerprintings de systèmes d'exploitation et d'équipements sont similaires à la différence près que le premier se limite à l'identification des protocoles implantés directement par les systèmes d'exploitations (couche Internet ou Transport) alors que le second opère au niveau de la couche application. Logiquement, ces méthodes de fingerprinting reposent sur les données échangées au niveau de la couche application mais aussi au niveau de la couche d'accès au réseau comme le souligne la figure 3.2. Effectivement, cette couche est généralement opérée par des pilotes spécifiques selon les équipements comme par exemple dans le cas de réseaux sans fil.

En premier lieu, ce type de fingerprinting aide à faire un inventaire des équipements réseaux déployés. Pour une entreprise, c'est très important puisque ça lui permet de voir si le matériel actif sur son réseau concorde bien à celui qu'elle a déployé. Ainsi, elle peut contrôler l'insertion d'ordinateurs ou d'autres équipements étrangers. De plus, elle peut contrôler si les machines sont bien mises à jours ou localiser certains équipements potentiellement vulnérables lors de la publication d'une vulnérabilité. Ainsi, elle peut rapidement réagir et mettre à jour ces machines en priorité. Finalement, les attaques par usurpation



(a) Cisco-Linksys WRT54g (b) HostAP

FIGURE 3.7 – Tests des piles protocolaires sans fil 802.11 grâce aux bits de contrôle

peuvent être plus facilement détectées si l'attaquant n'utilise pas exactement la même pile protocolaire.

Il arrive que l'identifiant de l'équipement même soit contenu dans les informations échangées notamment lors de l'établissement d'une session entre deux hôtes. Par exemple, les serveurs HTTP [7] incluent dans les entêtes leurs identifiant [4] comme le montre la figure 3.6. Cette technique est très simple mais n'est valable que si aucune technique de masquage n'est employée. En effet, il suffit de ne pas envoyer cette entête voire même d'en envoyer une fausse pour se faire passer pour un autre type de serveur. Il est important de noter que ces techniques de masquage représentent aussi des moyens légitimes lorsqu'il s'agit de cacher de l'information sans intérêt aux utilisateurs normaux mais cruciale pour les attaquants. Effectivement, lors de la découverte d'une vulnérabilité envers un programme spécifique, les attaquants vont en priorité chercher de tels programmes pour cibler leurs attaques.

Pour palier ces problèmes, des méthodes plus élaborées sont nécessaires. L'exemple de la figure 3.6 repris de [4] illustre par exemple des différences dans l'ordre des entêtes *server* et *date*. De plus, en envoyant des requêtes spécifiques les réponses peuvent différer. Comme dans les approches vues dans le cas du fingerprinting de système d'exploitation, ces requêtes sont soient malformées soit peu courantes. En effet, les erreurs d'implantation ou le manque de spécifications exactes se présentent souvent pour ces cas. L'outil [4] observe que lors de l'envoi d'une requête *DELETE*, pour supprimer une ressource sur le serveur et donc généralement interdit, les codes d'erreurs transmis varient.

Alors que les réseaux sans fil 802.11 se sont popularisés ces dernières années, leur sécurisation n'a cessé d'être améliorée. De manière générale, le premier pas pour pénétrer un tel réseau est de se faire passer pour un utilisateur légitime en lui volant son "identité" sur ce réseau. C'est dans cette optique que [13] cherche à identifier les points d'accès en se basant uniquement sur les bits de contrôle de trame. Sur la figure 3.7, chaque ligne représente un de ses bits et chaque colonne un test effectué dont la largeur dépend de la fréquence de réponse. Plus une case est noire, plus elle est fréquente. La différence est flagrante entre un point d'accès Cisco-Linksys WRT54g 3.7(a) et HostAP 3.7(b).

Le protocole SIP [6] est largement employé aujourd'hui notamment pour la gestion des sessions de VoIP. En parallèle à l'essor de cette technologie, les attaques se sont développées et notamment le SpIT (Spam over Internet Telephony) équivalent multimédia du spam (appels vocaux, vidéos...). Ce problème est traité dans [23] introduisant du fingerprinting actif grâce à deux types de messages :

- des messages malformés (par exemple avec un numéro de version de protocole impossible) entraînant différents types de réponses ;
- les requêtes *OPTIONS* pour récupérer la liste des opérations permises et qui varient selon les équipements.

De plus, de nombreux champs dans l'entête sont optionnels et leur présence ou non sont des indices utiles pour le fingerprinting de même que l'ordre dans lequel ils apparaissent.

Un point faible souvent adopté dans les techniques de fingerprinting est l'utilisation de générateur de nombres aléatoires qui sont souvent mal implantés. En effet, il arrive souvent que la distribution de ces nombres ne suive pas une loi de distribution uniforme. Ainsi Scholz *et al.* [24] parviennent à identifier les différents équipements SIP à partir des identifiants d'appel devant normalement être déterminés

aléatoirement à chaque nouvel appel émis.

Toujours dans le cadre de SIP, [8] présente une méthode reposant sur la construction de l'arbre syntaxique d'un message par rapport à la grammaire ABNF du protocole. Chaque élément de l'arbre est un champ terminal (valeur définie) ou non terminal (répétition, séquence...). Plus précisement, lors de la phase d'entraînement du système, les arbres syntaxiques provenant du même équipement sont comparés entre eux de manière à identifier les champs invariants ou dynamiques. Ensuite, les différences entre les arbres de deux équipements distincts sont identifiées. Il devient alors possible de faire de la discrimination des messages basés sur ces différences.

Dans le cadre du fingerprinting actif, FIG [14], précédemment introduit dans la section 3.3.1, est aussi capable d'identifier spécifiquement les versions implantées d'un protocole toujours en testant de nombreuses requêtes possibles pour en dégager le sous-ensemble qui permet d'identifier efficacement les équipements.

Enfin, Shu *et al.* introduisent un modèle formel pour le fingerprinting dans [22]. Bien qu'ici, il soit testé avec TCP et corresponde plutôt à du fingerprinting de systèmes d'exploitation, il est clairement applicable à tout autre protocole. Le modèle repose sur la définition de machines à états finis étendues paramétrées où chaque transition entre deux états dépend d'une partie du contenu d'un paquet. Grâce à cette formalisation, le but est de trouver une séquence d'entrée (messages reçus) qui permet de générer deux sorties différentes (messages émis) pour deux machines à états à discerner.

3.3.4 Comptage

Le comptage de machines est un autre genre de fingerprinting qui a pour objectif de différencier chaque machine unique. Dans le pire des cas, deux machines identiques ayant exactement les mêmes logiciels installés doivent être différentiables. La technique la plus simple consiste tout simplement à compter le nombre d'adresses IP différentes mais est totalement inefficace pour deux raisons majeures :

- l'usurpation d'adresses IP ou l'utilisation d'adresses IP dynamiques impliquant une surévaluation du nombre de machines ;
- le partage d'une adresse IP publique via un routeur NAT entraînant une sous-évaluation du nombre de machines.

L'intérêt de ce fingerprinting est le comptage précis des machines permettant de mieux gérer un réseau, notamment pour prévoir les ressources à allouer. Par ailleurs, les attaques par usurpation d'adresses IP sont plus facilement détectées. De même, si une signature est attribuée à chaque machine, sa traque et le suivi de sa localisation sur Internet sont envisageables.

Pour résoudre le problème des machines derrière un NAT, Bellovin suggéra d'utiliser le champ *Id* de l'entête IP [39]. Ce champ permet d'identifier de manière unique chaque paquet mais sa génération est généralement implantée via un compteur. Ainsi, un hôte est identifié en repérant des séquences linéaires de ce champ.

Une technique plus générale consiste à étudier les gigues des horloges TCP voire ICMP [30]. En effet, ces paquets contiennent un timestamp à partir duquel la machine d'observation est capable de dériver la gigue de l'horloge qui mesure son décalage avec le temps réel. En réalité, c'est l'évolution de ce décalage au cours du temps qui permet de différencier chaque machine. Cette approche s'applique également à la distinction des machines virtuelles de machines réelles. Fink élabora une technique similaire dans [16]. Plus récemment, [9] se focalise sur la distinction d'équipements sans fil 802.11 de manière totalement passive en se basant sur le délai entre deux paquets servant à sonder la présence d'un point d'accès. En fait, les auteurs observent que ce temps dépend bien entendu du pilote sans fil, du système d'exploitation mais aussi de la gigue de l'horloge ou encore du bruit environnant.

Approches	Avantages	Inconvénients
Signatures manuelles	– simple à appliquer – souvent bien testée et éprouvée	– signatures spécifiques → bases de données devant être mises à jour – nombre de signatures élevées devant être optimisé – création manuelle des signatures
Apprentissage (signatures automatiques)	– génériques → rapidement applicables dans différents contextes – pas de mises à jour manuelles – peu d'interventions humaines	– nécessité d'avoir un jeu d'apprentissage complet et correctement étiqueté – nombreux choix de méthodes souvent plus complexes que de simples signatures à comparer – paramétrage parfois difficile

TABLE 3.3 – Techniques de fingerprinting

3.4 Bilan

Le fingerprinting est un domaine vaste qui ne vise pas forcément à obtenir toujours le même type d'identification. Ce chapitre a présenté les applications majeures ainsi que les techniques associées. Il est important de remarquer que les caractéristiques étudiées sont souvent similaires (taille d'un paquet, d'un flux, temps entre deux paquets) malgré la mise en valeur de nouvelles plus récemment. Cependant, les réelles différences se situent au niveau des représentations de ces données et de leur traitement. Le tableau 3.3 récapitule les deux grandes classes de fingerprinting que l'on peut trouver actuellement. Malgré leur simplicité, les bases de signatures manuelles sont aujourd'hui généralement employées du fait de leur simplicité d'utilisation à l'instar de p0f [2] ou NMAP [6].

Alors qu'historiquement, l'objectif souhaité était la classification des machines selon leur système d'exploitation et la classification du trafic, celui-ci a plus récemment évolué vers le fingerprinting d'équipements. En fait, un objectif principal est la distinction du trafic normal et du trafic malveillant qui pouvait, par le passé, être réalisée à un niveau assez macroscopique (classification de trafic, de protocoles). Aujourd'hui, les malwares ne nécessitent plus forcément de protocoles spécifiques. De plus, les attaquants ont fait d'énormes progrès dans la dissimulation de leur trafic. Par exemple, une machine qui envoyait du spam (beaucoup de flux similaires) ou effectuait une attaque de déni de service (temps inter-paquets très courts) était facilement repérable. Actuellement, avec le développement des botnets, ces actions sont distribuées et deux alternatives sont envisageables pour améliorer leur détection :

– avoir une vue plus globale sur le réseau ce qui oblige les différents acteurs à collaborer mais aussi à échanger et stocker beaucoup de données ;
– associer une signature précise au malware.

La seconde méthode est clairement moins coûteuse, réalisable grâce au fingerprinting d'équipements. Par exemple, dans le cas de l'envoi de Spam, les programmes bots utilisent généralement leur propre implantation d'un client SMTP souvent plus légère pour gagner en rapidité et en discretion. Le fingerprinting ne doit plus seulement être capable de détecter du trafic SMTP mais aussi la version utilisée, c'est-à-dire le logiciel bot. Ce petit exemple montre l'intérêt du fingerprinting d'équipements dans le

domaine de la sécurité. Néanmoins, ce type de fingerprinting demeure pour le moment très peu exploré.

Enfin, de nouveaux types de fingerprinting émergent tels que [15] où l'objectif est de déterminer automatiquement le langage d'utilisateurs réels dans des communications VoIP chiffrées.

Deuxième partie

Contribution

4

Modélisation des botnets

Sommaire

4.1 Introduction 83
4.2 Cas d'étude 84
4.2.1 Supervision des réseaux et services 84
4.2.2 Supervision d'un pot de miel distribué 85
4.2.3 Techniques existantes 86
4.3 Terminologie 86
4.3.1 Notations 86
4.3.2 Métriques 87
4.4 Modèles Internet Relay Chat 88
4.4.1 Topologie Internet Relay Chat 88
4.4.2 Premier modèle 89
4.4.3 Second modèle 92
4.4.4 Délais 97
4.5 Modèles P2P 97
4.5.1 Totalement maillé, Slapper 97
4.5.2 Table de hachage distribuée, Chord 99
4.5.3 Délais 104
4.6 Bilan 104

4.1 Introduction

Le chapitre 1 a présenté les botnets, ces réseaux de machines compromises contrôlées par un attaquant. Leurs tailles en font de redoutables moyens d'attaque dont personne ne peut douter aujourd'hui au vue des nombreuses attaques réalisées et observées. Cependant, les connaissances actuelles se limitent principalement à des observations réelles qui ont l'avantage d'être réalistes mais l'inconvénient d'être limitées à un botnet particulier à un moment précis. Pour pouvoir étudier les botnets de manière plus générique et selon de nombreux paramètres pour diversifier les cas, l'élaboration de modèles est une étape primordiale.

Les modélisations actuelles introduites dans la section 1.6 sont de deux types :

– celles visant à modéliser le déploiement du botnet, c'est-à-dire sa propagation sur Internet [54, 25] ;
– celles cherchant à étudier les performances d'un botnet une fois déployé [37, 26].

Le déploiement d'un botnet est comparable à la propagation d'un vers qui fut étudiée dans de nombreux travaux (voir section 1.6). L'étude des performances d'un botnet déployé est un domaine encore peu exploré mais dont les avantages sont multiples. En premier lieu, dans le cadre de l'élaboration de contre-mesures face à un botnet, de tels modèles aident à mieux estimer sa puissance potentielle et ainsi prévoir son impact. Par exemple, des outils introduits dans la section 1.6 parviennent à acquérir des informations sur un botnet réel (nombre de machines infectées, architecture sous-jacente...) pouvant alors servir à instancier un modèle et ainsi à prévoir les effets néfastes lors du lancement d'une attaque. Par ailleurs, il devient possible d'expertiser la robustesse du botnet et de voir si le fait de déconnecter certains bots est vraiment efficace. A première vue, le modèle le plus simple considérerait que l'efficacité et la robustesse d'un botnet sont linéaires par rapport à sa taille. Bien entendu, cette idée est bien trop simple et nécessite d'être affinée car rien n'assure que tous les bots puissent être contactés et, qui plus est, en même temps.

Outre l'aspect malveillant d'un botnet, ses performances restent extraordinaires alors que le passage à l'échelle reste le point faible de nombreuses applications communicantes. S'inspirer de tels malwares peut ainsi contribuer à une meilleure conception des applications. Cependant cela implique d'être sûr des performances réalisables pour déterminer si elles sont suffisantes dans le cas d'une application donnée. Ce chapitre commencera donc par introduire une telle application où la mise en œuvre d'une architecture similaire à un botnet semble être justifiée.

De plus, ce chapitre définira les différentes caractéristiques d'un botnet intéressantes à mesurer ainsi que des modèles associés (IRC ou P2P). Enfin, ceux-ci seront comparés par rapport aux autres existants [37, 26].

4.2 Cas d'étude

4.2.1 Supervision des réseaux et services

La supervision des réseaux et services est une application dont le passage à l'échelle est primordial et où les botnets semblent être une architecture de support intéressante. Plusieurs domaines composent la supervision des réseaux : gestion de défaut, gestion de configuration, gestion comptable, gestion d'exécution et gestion de sécurité. Une supervision optimale doit pouvoir opérer sur des équipements variés (ordinateurs, pare-feux, routeurs, téléphones, imprimantes, serveurs...) et se faire à travers Internet malgré les nombreux équipements qui pourraient bloquer le trafic légitime de supervision tels que les pare-feux ou encore les routeurs NAT (*Network Addresses Translator*). C'est ce que montre la figure 4.1 où une entreprise peut avoir plusieurs sites voire des machines hors du réseau (mobilité) ou déléguer la gestion de son réseau à un partenaire.

Il est alors clair qu'adapter les botnets pour la supervision des réseaux est une solution envisageable à large échelle puisque les attaquants ont déjà surmonté les problèmes précédents en les utilisant. Par conséquent, les modèles permettront de déterminer si une telle supervision est possible ou non. Plus précisement, l'objectif visé est la configuration de masse avec les propriétés suivantes :

– passage à l'échelle : un grand nombre de machines à configurer,
– efficacité : nouvelle configuration déployée rapidement,
– anonymat : un attaquant ne doit pas pouvoir découvrir les machines supervisées ainsi que celles qui contrôlent le système pour éviter un détournement de l'usage du botnet,

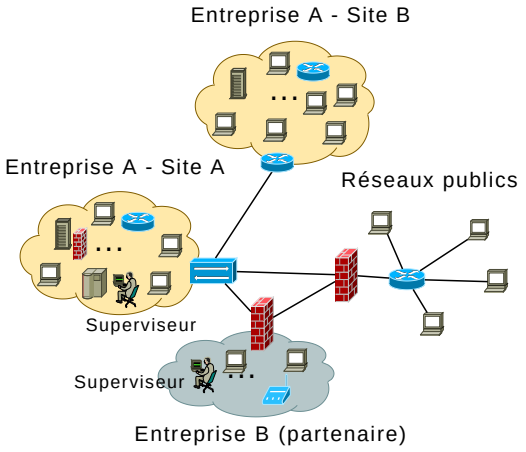


FIGURE 4.1 – Supervision des réseaux à large échelle

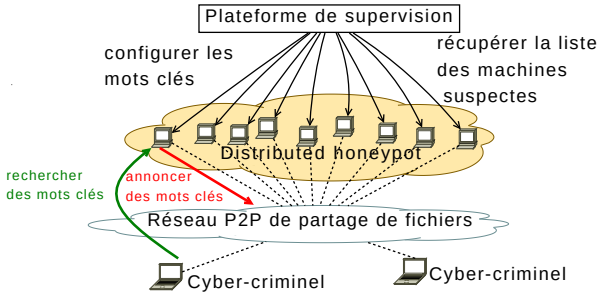


FIGURE 4.2 – Cas de figure du honeypot distribué

– sécurité : une machine supervisée ne doit pas pouvoir lancer de fausses requêtes de supervision sur le réseau,
– garantie d'atteignabilité : l'administrateur doit pouvoir savoir quel est le résultat potentiel de l'envoi d'une requête et notamment le nombre de machines qui l'auront correctement reçu. Cela permet ensuite de prévoir des mesures supplémentaires si ce nombre n'est pas satisfaisant.

4.2.2 Supervision d'un pot de miel distribué

Bien que de manière générale le but soit de proposer une solution globale de supervision à large échelle, l'étude d'un cas précis extrêment contraignant donnera de bonnes indications valables pour la plupart des autres cas. Ainsi, l'application visée doit permettre de traquer les cyber-criminels sur les réseaux pair-à-pair d'échange de fichiers comme proposé dans [2]. Pour cela, des mots-clés spécifiques sont annoncés sur le réseau et il suffit alors de récupérer les personnes qui recherchent ces mots-clés. De manière à améliorer la détection de ces personnes, il faudrait qu'un grand nombre de pairs annoncent ces mots-clés. L'architecture de type honeypot distribué de la figure 4.2, où tout utilisateur du réseau qui le souhaite peut participer, est donc envisageable. Dans le cas présent, la supervision se focalise sur la configuration des honeypots en leur fournissant les différents mots-clés, ce qui correspond bien à faire de la configuration en masse.

Les propriétés énoncées précédemment sont ainsi plus précisément définies :
– passage à l'échelle : tout utilisateur doit pouvoir participer, équivalent à des centaines de milliers de machines,
– efficacité : les mots-clés sont spécifiques mais pas forcément en lien avec le contenu et peuvent donc changer assez rapidement,
– anonymat : un attaquant ne doit pas pouvoir attaquer une partie du système pour se protéger lors d'éventuels téléchargements,
– sécurité : un attaquant ne doit pas pouvoir annoncer une liste de faux mots-clés,
– garantie d'atteignabilité : l'administrateur doit être en mesure d'évaluer le pourcentage de machines qui a bien la liste de nouveaux mots-clés à chaque mise à jour.

4.2.3 Techniques existantes

Dans une solution classique de supervision, le superviseur communique directement avec chaque équipement à superviser. Comme l'architecture centralisée limite le passage à l'échelle, plusieurs solutions décentralisées sont apparues. On peut par exemple citer la supervision par délégation [11] ou encore l'utilisation de niveaux hiérarchiques et de superviseurs en cascade [1]. En fait, chaque superviseur intermédiaire doit transformer les requêtes de supervision de manière à ce que celles-ci soient adaptées aux différents équipements dont ils sont responsables. Les réponses doivent aussi être transformées ou agrégées comme dans [7]. Dans ces cas, les requêtes pour les différents équipements ne sont pas identiques. Dans le contexte de la configuration de masse, où la même requête est transmise à tous les équipements, il n'y a plus besoin d'avoir des superviseurs intermédiaires sensibles à des pannes ou à des attaques supplémentaires. Une extension de la supervision hiérarchique basée sur les réseaux pair-à-pair est proposée dans [6]. [4] propose de résoudre des recherches dans les réseaux pair-à-pair avec des caractères génériques en se basant sur des techniques d'agrégation et de réplication. *Echo Pattern* [8] est un modèle proche des réseaux pair-à-pair qui promet de bonnes performances. Il existe aussi des solutions de supervision utilisant les réseaux actifs [9]. Dans ce cas une requête est un programme léger transmis entre les différents hôtes du réseau. Cependant, leur déploiement est très limité car il nécessite des équipements dédiés. Certaines solutions sont dédiées aux réseaux Ad-Hoc comme [5] où le réseau est découpé en groupes avec pour chacun d'eux un responsable élu. Pour un botnet, aucune machine spécifique n'a besoin d'être utilisée hormis la plateforme de management car tout le monde peut participer et les ressources nécessaires sont faibles contrairement à la plupart des autres solutions.

Cependant, l'adaptation de logiciels malveillants à des fins bienveillantes n'est pas nouvelle à l'instar de [10] qui exposait l'utilisation d'un vers patrouillant sur les machines à superviser avant de retourner à la plateforme de management. Malheureusement, les études demeurent très limitées. Le vers Code-Green [60] est aussi très connu car il désinfectait les machines infectées par Code-Red. Le problème de ce type de vers est la légalité car sa propagation reste souvent incontrôlée. Dans une étude précédente [3], une première ébauche plus générale de l'adaptation de logiciels malveillants pour la supervision des réseaux fut introduite.

4.3 Terminologie

Cette section introduit les différentes notations et métriques qui seront ensuite reprises et adaptées aux différents modèles.

4.3.1 Notations

La figure 4.3 illustre un exemple de botnet quelconque où aucune architecture précise n'est pour le moment spécifiée, c'est-à-dire que les différentes machines représentées peuvent aussi bien être des machines d'utilisateurs que des serveurs. Dans le cadre de la modélisation et du fait de la représentation par des graphes, ces machines sont aussi nommées nœuds. Les deux paramètres les plus importants du réseau sont :

– m : le facteur de branchement maximal qui représente le nombre maximal de voisins d'un nœud,
– N : le nombre total de nœuds dans le réseau.

Pour avoir une modélisation plus réaliste, deux contraintes sont ajoutées sur les nœuds :

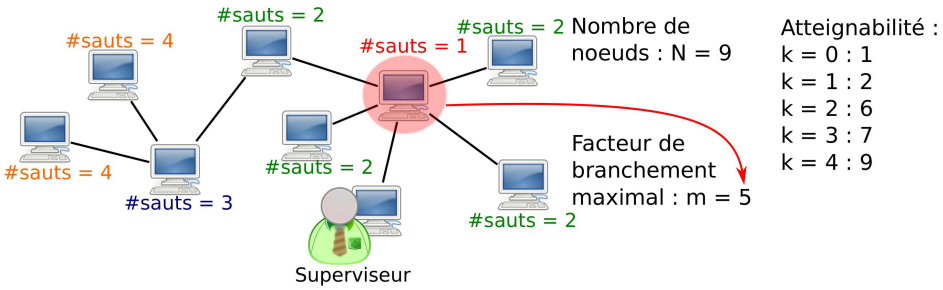


FIGURE 4.3 – Notations générales utilisées pour la modélisation des botnets

– β : la probabilité d'être compromis illustrée sur la figure 4.4(a). Un attaquant qui a compromis un nœud est seulement capable d'écouter l'ensemble des communications et donc de découvrir les voisins.
– $\alpha(m)$ est le facteur de disponibilité mis en évidence sur la figure 4.4(b). Un nœud disponible est capable de faire suivre les requêtes de supervision. Ce facteur représente donc le fait qu'un nœud n'a plus assez de ressources pour réussir à le faire correctement. Ce facteur sera alors modélisé par une fonction décroissante de m car plus un nœud doit maintenir de connexions plus il risque d'être surchargé.

4.3.2 Métriques

Cette section définit les différentes métriques d'évaluation des performances du botnet.
Dans l'objectif du passage à l'échelle, la métrique principale est l'atteignabilité.

Définition 14 : *L'atteignabilité est la proportion moyenne de bots atteints à une distance maximale d du botmaster, ce qui est un bon indicateur de passage à l'échelle. Elle se note* $att(d)$.

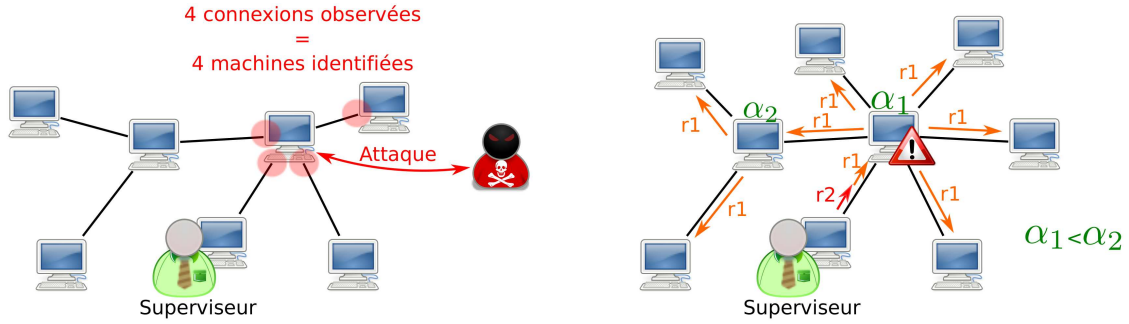
La figure 4.3 illustre cette métrique.
L'atteignabilité précédemment définie est dépendante d'une distance donnée et il est évident qu'une mesure plus générale doit permettre d'estimer le nombre de bots atteignables quelque soit leur distance par rapport au botmaster.

Définition 15 : *L'atteignabilité moyenne, avg_att, est l'atteignabilité moyenne de toutes les distances possibles.*

Il est aussi parfois profitable de connaître le nombre exact de bots atteints et pas seulement la proportion :

Définition 16 : *L'atteignabilité absolue est le nombre de bots atteints à une distance maximale d du botmaster. Elle se note* $\overline{att}(d)$.

Par conséquent, certains modèles calculeront l'atteignabilité absolue et l'atteignabilité normale ou pas mais leur équivalence est évidente puisque $\overline{att}(d) = N \times att(d)$.
De même l'atteignabilité moyenne absolue est définit : Il est aussi parfois profitable de connaître le nombre exact de bots atteints et pas seulement la proportion :



(a) Attaque permettant à l'attaquant de surveiller les connexions d'une machine

(b) Surcharge, la requête r1 entraîne une surcharge sur le nœud qui lui empêche de transmettre la requête suivante (r2)

FIGURE 4.4 – Contraintes sur les nœuds du réseau

Définition 17 : *L'atteignabilité absolue moyenne* $\overline{(avg_att)}$ *est l'atteignabilité absolue moyenne sur toutes les distances possibles.*

Hormis ces métriques visant à calculer le nombre de bots, une autre métrique annexe est le délai nécessaire pour transmettre un ordre aux bots.

Définition 18 : $temps(d)$ *est le temps nécessaire pour que la requête envoyée par le botmaster arrive jusqu'aux bots atteignables à une distance maximale* d.

L'intérêt d'une telle estimation est évident lors de la quantification du temps nécessaire pour qu'un ordre soit exécuté sur un nombre donnés de bots. En effet, en calculant l'atteignabilité à une distance d et en calculant le temps nécessaire pour leur transmettre une requête, il est facile de déterminer le délai pour atteindre un certain nombre de bots.

Par ailleurs, dans le cas où plusieurs requêtes indépendantes doivent être transmises, cette métrique est très utile pour apprécier leur ordre d'émission selon leur priorité car certaines opérations nécessitent d'être effectuées avant un certain temps telles que des mises à jour de sécurité importantes.

4.4 Modèles Internet Relay Chat

4.4.1 Topologie Internet Relay Chat

Pour rappel, un réseau IRC est composé de plusieurs serveurs interconnectés de manière à former un arbre de diffusion. Les différents bots sont alors connectés sur ces derniers. Lorsque le superviseur souhaite envoyer une requête aux bots, il la transmet au serveur sur lequel il est connecté qui la fait alors suivre aux autres via l'arbre de diffusion. Chacun de ces serveurs est aussi responsable de l'envoyer vers les bots connectés. La racine de l'arbre de propagation est donc le serveur sur lequel le superviseur est connecté. Étant donné que le cœur du réseau se situe au niveau des serveurs et que leurs caractéristiques et celles des bots sont forcément différentes (disponibilité, connectivité...), les modèles IRC se focaliseront en premier lieu sur l'arbre de diffusion pour, dans un second temps, prendre en compte les bots. Tout au long de cette section, les termes *nœuds* et *serveurs* sont donc équivalents.

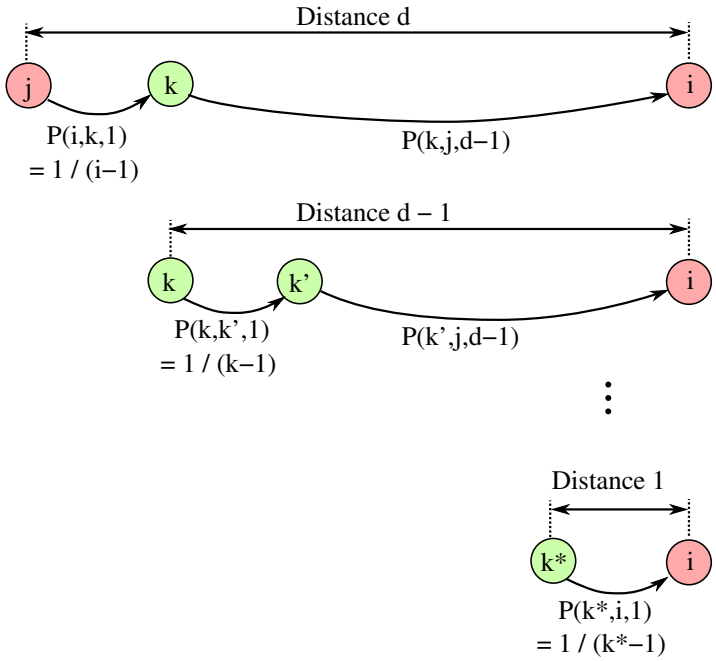


FIGURE 4.5 – Calcul des probabilités de distance exacte dans un arbre aléatoire récursif

4.4.2 Premier modèle

Ce premier modèle considère une construction aléatoire de celui-ci. En réalité, un administrateur d'un réseau IRC aura plutôt tendance à placer des nœuds centraux fiables, c'est-à-dire des serveurs qui auront peu de chances d'être déconnectés car une déconnexion centrale coupera clairement le réseau en deux contrairement à une déconnexion périphérique. Cependant, dans le cadre de cette modélisation, la fiabilité des nœuds est constante et similaire. Ainsi, modéliser le graphe des serveurs IRC par un arbre aléatoire est justifié. Ce modèle et son évaluation furent publiés dans [4].

Distance entre deux nœuds

Les travaux exposés dans [8] soulignent la possibilité de considérer un arbre aléatoire construit pas à pas de manière récursive où chaque nœud est numéroté. À chaque fois qu'un nœud est rajouté dans l'arbre celui-ci a des chances équiprobables d'être connecté à un précédent. Dans le cas présent, l'ensemble ordonné des nœuds est $L = \{n_1, n_2, \ldots, n_N\}$ où $\forall i,\ 1 \leq i \leq N-1$, le nœud n_i a été ajouté avant le nœud n_{i+1}.

Ainsi, lorsque le nœuds n_j est ajouté à l'arbre, celui-ci peut être relié à tout n_i avec $1 \leq i < j$ de manière équiprobable. Par conséquent, la probabilité que le nœud n_j soit connecté au nœud n_i se définit de la manière suivante :

$$p_{i,j} = \begin{cases} \dfrac{1}{j-1} & \text{pour} \quad 1 \leq i < j \\ 0 & \text{sinon} \end{cases} \tag{4.1}$$

La probabilité que le nœud n_i soit à une distance d du nœud n_j avec $1 \leq i < j \leq N$ se dénote $P(i, j, d)$ et est calculable récursivement. En effet, lorsque la distance séparant deux nœuds est égale à un, cela revient à utiliser directement la formule 4.1 qui indique la probabilité pour ce nœud j de se connecter

à un nœud k avec $k < i$. Il suffit alors de la multiplier par la probabilité que ce dernier soit connecté au nœud i à une distance $d-1$ et ainsi de suite. Ce principe est illustré sur la figure 4.5 où pour aller de i à j en d sauts, il est nécessaire de traverser $d-1$ nœuds intermédiaires. Grâce à cette représentation, $P(i,j,d)$ est définit comme suit :

$$P(i,j,d) = \begin{cases} p_{i,j} \quad \text{pour} \quad d = 1 \\ \sum_{k=1}^{j-1} p(k,j) \times P(k,i,d-1) = \frac{1}{j-1} \sum_{k=1}^{j-1} P(k,i,d-1) \quad \text{sinon} \end{cases} \tag{4.2}$$

Les indices de la somme varient entre 1 et $j-1$ car la construction récursive de l'arbre au moment de l'ajout du nœud n_j implique la création d'un chemin entre ce nœud et le nœud n_i. A l'inverse lors de l'ajout d'un nœud n_k avec $k > j$, s'il existait un autre chemin possible passant par k, ce ne serait plus un arbre d'où un indice maximal égal à $j-1$.

Distance entre un nœud et plusieurs autres

Dans le cadre de cette étude, c'est-à-dire l'envoi massif d'une commande via le réseau, il est plus intéressant de déterminer la probabilité $P(o,d,S)$ pour un nœud o d'avoir un ensemble S de nœuds à une distance maximale d. Soit $1 \leq i \leq N-1$ le nombre de nœuds que l'on considère directement connectés au nœud d'origine o, ceux-ci forment l'ensemble :

$$CHOIX = \{c_1, \ldots, c_i\} \subseteq S$$

Son complémentaire spécifie les nœuds qui devront être joignables au deuxième ou au plus au d^{e} saut :

$$RESTE = N - CHOIX = \{r_1, \ldots, r_p\}$$

Cet ensemble doit être divisé entre les différents nœuds de $CHOIX$ sur lesquels ils seront raccordés directement ou indirectement. En clair, les nœuds de $CHOIX$ représentent les racines des sous-arbres au deuxième saut et il est nécessaire d'assigner chaque nœud restant à un de ces sous-arbres. Ainsi, l'ensemble $SUITE$ symbolise l'affectation des nœuds à partir du second niveau :

$$SUITE = \{< s_{11}, s_{12} >, \ldots, < s_{p1}, s_{p2} >\}$$

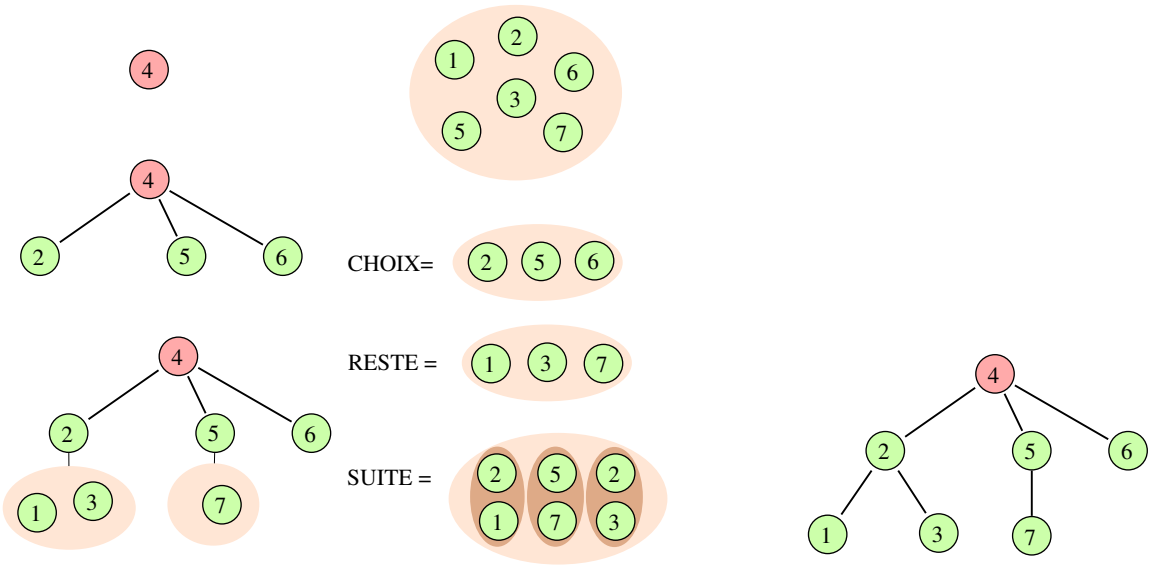
avec : $\forall j, s_{j1} \in CHOIX, s_{j2} \in RESTE$ et $\forall k \neq j, s_{k2} \neq s_{j2}$. La première condition impose que chaque nœud racine des sous arbres appartienne bien à un nœud de la combinaison choisie ($CHOIX$). La seconde contrainte signifie que les sous arbres ne peuvent être engendrés qu'à partir des nœuds restants et ne peuvent bien entendu pas partager un nœud commun (sinon des cycles sont introduits).

Par exemple, si $< c_x, r_y >$ signifie que le nœud r_y fait partie du sous arbre enraciné sur le nœud c_x.

Un exemple est donné sur la figure 4.6(a) avec 7 nœuds et $o = 4$. La première étape choisit de connecter directement à o les nœuds $CHOIX = \{2,5,6\}$. Chacun des nœuds restants ($RESTE = \{1,3,7\}$) est alors assigné à un sous arbre engendré par un de ces nœuds choisis. Ici, 1 et 3 sont greffés sur le sous arbre de 2 et 7 sur le sous arbre de 5. Ensuite, l'arbre est complété récursivement en appliquant le même processus sur les sous-arbres. Un résultat possible est tracé sur la figure 4.6(b).

Bien évidemment, ces ensembles doivent être construits au hasard et le calcul de probabilité nécessite de prendre en compte tous les cas possibles. Ainsi, l'ensemble $CHOIX(S)$ regroupe l'ensemble des combinaisons possibles de 1 ou au maximum le nombre de nœuds restants. Par conséquent, $CHOIX$ sera instancié avec une de ces combinaisons. Plus formellement, $CHOIX(S)$ est construit à partir de l'ensemble S des nœuds possibles du sous arbre considéré :

$$CHOIX(S) = \cup(\{c_1, \ldots, c_i\}, \forall k, l, k \neq l, c_k \neq c_l, c_k \in S, c_l \in S), 1 \leq i \leq |S|$$



(a) Première itération du choix des différents ensembles

(b) Résultat final

FIGURE 4.6 – Calcul des probabilités de distances minimales dans un arbre aléatoire

Ainsi pour toute combinaison c de $CHOIX(S)$, l'ensemble ordonné $RESTE(S,c)$ est définit comme précédemment, c'est-à-dire :

$$RESTE(S,c) = S - c$$

De plus, $SUITE(c,r)$ définit l'ensemble des affectations des nœuds de r aux sous-arbres dont les nœuds racines appartiennent à c. Cet ensemble a donc une taille identique à r :

$$SUITE(c,r) = \cup\{< s_{11}, s_{12} >, \ldots, < s_{p1}, s_{p2} >\}, \forall j, s_{j1} \in c, s_{j2} \in r, \forall k \neq j, s_{k2} \neq s_{j2}, p = |r|$$

Finalement, la probabilité qu'un nœud o soit à une distance au plus égale à d d'un ensemble N s'écrit sous la forme suivante :

$$P(o,d,N) = \begin{cases} \sum_{c \in CHOIX(N)} \sum_{s \in SUITE(c,RESTE(S,c))} [\prod_{n_i \in c} P'(o,n_i) \times P(n_i, d-1, \Phi(s,n_i))] \; si \; d > 0 \\ 0 \; sinon \end{cases} \quad (4.3)$$

où $\Phi(s,n_i) = \cup\{s_{j2}\}$ tel que $< s_{j1}, s_{j2} > \in c$ et $s_{j1} = n_i$, c'est-à-dire l'ensemble des nœuds compris dans le sous arbre ayant pour racine n_i.

$P'(i,j)$ est la probabilité que le nœud i soit connecté au nœud j directement. En fait, cette probabilité est similaire à P_{ij} à la différence près qu'elle dépend du nœud dont l'indice est le plus élevé, c'est à dire celui ajouté en second :

$$P'(i,j) = \frac{1}{max(i,j) - 1}$$

Dans l'équation (4.3), il s'agit donc bien de sommer les différentes probabilités de configurations de sous arbres possibles. Pour chacun d'eux, la probabilité que les nœuds qui lui sont assignés soit à une distance diminuée de 1 est alors calculée récursivement. Il est donc nécessaire de multiplier ces probabilités pour obtenir celle de la configuration dans son ensemble. De plus, le second cas avec une distance négative symbolise le cas où l'ensemble des nœuds originaux, N, n'auraient pas été atteint.

Atteignabilité

Il est important de remarquer que l'atteignabilité n'est pas encore formellement défini. Pour rappel l'atteignabilité est la proportion moyenne de nœuds atteints à une distance d. Par construction, ce nombre peut varier entre 1 et N (nombre total de nœuds) et le calcul exact de l'atteignabilité exige de former tous les ensembles de nœuds possibles de taille variant entre 1 et N. Soit S_i^o, l'ensemble des combinaisons possibles de nœuds de taille i, excepté le nœud o, l'atteignabilité à partir d'un nœud o se définit donc comme :

$$att(d,o) = \frac{\sum_{i=1}^{N} \sum_{s \in S_i^o} P(o,d,s)}{N}$$

.

Cependant, selon le nœud o choisi, l'atteignabilité sera différente car les premiers nœuds seront plus centraux que les derniers par construction. Ainsi, l'atteignabilité générale se calcule en faisant la moyenne sur tous les nœuds d'origine possibles :

$$att(d) = \frac{\sum_{o \in [1,\ldots,N]} att(d,o)}{N}$$

.

Problèmes et limitations de ce modèle

Le problème majeur de ce modèle est l'obligation de calculer de nombreux ensembles comme $CHOIX(S), RESTE, SUITE$ ou S_i^o à cause de la numérotation des nœuds qui impose d'examiner tous les cas possibles. De plus, chacun d'eux est souvent construit grâce à des combinaisons ce qui implique la construction de nombreux ensembles similaires. Par exemple, sur la figure 4.6(b), une permutation quelconque des nœuds donne une structure similaire et des performances similaires tant que le nœud d'origine reste positionné au même endroit quelque soit son numéro. Par conséquent, les calculs se retrouvent très longs et ce modèle n'est donc pas adapté à des tests exhaustifs.

De plus, les différentes contraintes comme la disponibilité d'un nœud ou encore le risque d'attaque ne sont pas encore prises en compte dans ce modèle. Ainsi, l'évaluation dans le chapitre 7 de ce modèle sera succincte.

Enfin, ce modèle n'est qu'une approximation puisque la construction de l'arbre entraîne une certaine dépendance dans l'ordre d'insertion des nœuds et donc leur placement. Par exemple il est impossible d'avoir une branche constituée successivement du nœud 2 puis du nœud 3 et enfin du nœud 1 puisque le nœud 2 est inséré avant le nœud 3 et aurait donc forcément été lié au premier. Ainsi il est en plus nécessaire d'introduire une fonction de vérification de cohérence, par rapport à la construction, pendant le calcul récursif de $P(o,d,N)$ (équation (4.3)).

4.4.3 Second modèle

Face aux problèmes précédents, il fut inévitable de concevoir un nouveau modèle qui les contournaient. En y regardant de plus près, les difficultés rencontrées découlent essentiellement de la construction aléatoire et récursive de l'arbre qui imposait une numérotation des nœuds contraignant fortement la

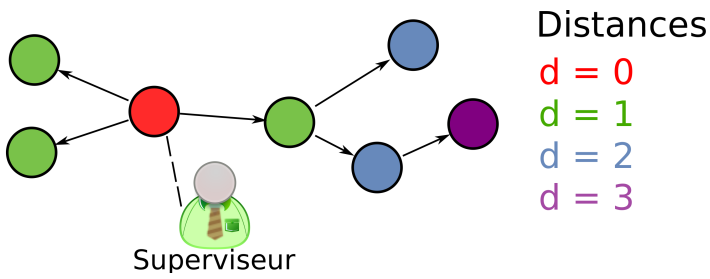


FIGURE 4.7 – Second modèle pour les botnets IRC avec des nœuds non numérotés, propagation d'une requête

structure de l'arbre. Au final, le modèle précédent revient à produire tous les arbres possibles. Par conséquent, ce second modèle n'utilise plus de numérotation de manière à éviter ces problèmes. Ce modèle ainsi que les expérimentations ont été publiés dans [3, 1]

Pour rappel, chaque serveur a un nombre maximum m de voisins : c'est le facteur de branchement maximal. Le nombre de connexions qu'un serveur maintient avec d'autres est donc compris entre 1 et m. Cette limitation recentre le modèle sur les réseaux IRC où un serveur est connecté à quelques autres seulement. La figure 4.7 montre la topologie d'un réseau de serveurs IRC ainsi que le calcul des différentes distances par rapport au nœud du superviseur. Celui-ci peut d'ailleurs être connecté via un lien du même type que les bots ou agir directement sur le serveur. Étant donné la nature spéciale de cette connexion, le nœud serveur sera considéré comme le nœud d'origine.

[5] expose un moyen simple pour estimer le nombre de nœuds à une distance quelconque basée sur les fonctions génératrices qui furent notamment employées dans [48].

En premier, un rappel des notions sur les fonctions génératrices et certaines propriétés est donné ci-dessous et le lecteur intéressé pourra consulter [5] pour de plus amples détails notamment les différentes preuves et leur utilité dans les graphes.

Définition 19 : *Étant donné une fonction de probabilité* $p_k = P(X = k)$*, sa fonction génératrice se définit comme :*

$$G_0(x) = \sum_{k=0}^{\infty} p_k x^k$$

Propriété 1 : *L'espérance d'une fonction de probabilité est égale à la dérivée de sa fonction génératrice instanciée avec* $x = 1$ *:*

$$E(X) = G_0'(1)$$

Propriété 2 : *Si la distribution d'une propriété k d'un objet est générée par la fonction génératrice* $G_0(x)$ *alors la somme de m réalisations indépendantes de l'objet est générée par la fonction génératrice* $[G_0(x)]^m$.

Nombre de voisins d'un nœud

La probabilité p_k est la probabilité pour un nœud d'avoir k serveurs comme voisins (connectivité) : un nœud a donc une probabilité constante d'être connecté à n'importe quel autre. Cette modélisation est équivalente à la précédente à l'exception près que, précédemment, l'ordre des nœuds imposait aux

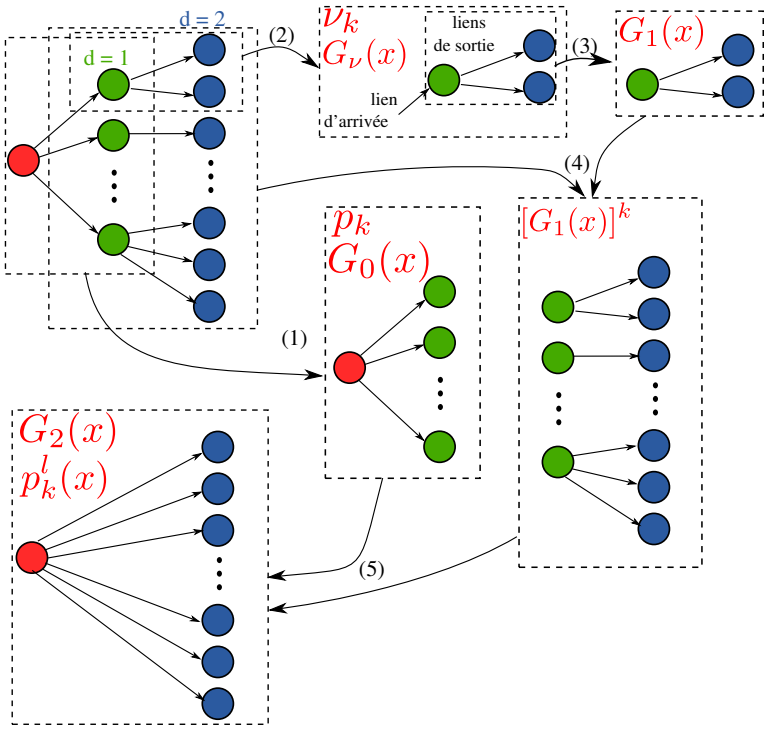


FIGURE 4.8 – Calcul du nombre de voisins au second saut pour le second modèle de botnets IRC

premiers une connectivité plus élevée ce qui obligeait ensuite le calcul de l'atteignabilité en considérant tout nœud comme origine avant d'en faire la moyenne. Avec le nouveau modèle, il n'y a plus besoin d'utiliser cet artifice car chaque nœud est directement considéré dans l'ensemble du réseau.

Concernant p_k, p_0 signifie que le serveur est déconnecté (hors-service) soit $p_0 = 1 - \alpha(m)$. Ensuite, il est impossible d'avoir $k > m$ d'où $p_k = 0$ pour $k > m$. Enfin, sans aucun connaissance à priori, dans les autres cas "normaux", la distribution est aléatoire et uniforme donc $p_k = \alpha(m) \times \frac{1}{m} = \dfrac{\alpha(m)}{m}$ pour $1 \leq k \leq m$. Cette distribution ne suppose donc aucune topologie spécifique meilleure qu'une autre.

La fonction génératrice de la fonction p_k est alors calculable :

$$G_0(x) = \sum_{k=0}^{\infty} p_k x^k = \sum_{k=0}^{m} p_k x^k = p_0 + \sum_{k=1}^{m} p_k x^k = p_0 + \frac{\alpha(m)}{m} \times \sum_{k=1}^{m} x^k \tag{4.4}$$

Par définition, la moyenne de p_k, c'est-à-dire le nombre moyen de voisins, est directement déduit : $\mathbb{E}(k) = G_0'(1)$.

Voisins au second saut

Le modèle doit aussi permettre de calculer le nombre de voisins à n'importe quelle distance. A titre d'exemple, les différentes étapes du calcul pour le second saut sont détaillées ici et schématisées sur la figure 4.8. La première étape consiste à trouver la distribution de probabilités des voisins directs ce qui correspond ainsi à p_k comme le montre la première étape sur la figure.

Ensuite il faut évaluer la probabilité $\nu(k)$ d'atteindre un nœud (serveur) ayant une connectivité égale à k liens en se servant de l'un d'entre eux. C'est la seconde étape représentée sur la figure 4.8. En fait, il y a k liens et donc k façons d'arriver à ce serveur qui a lui-même une probabilité p_k d'avoir k voisins

d'où :

$$\nu(k) = \frac{kp_k}{\sum_{i=0}^{m} ip_i}$$

Le dénominateur divise le cas précis où il y k voisins par l'ensemble des cas possibles.

La fonction génératrice en résultant est :

$$G_\nu(x) = \sum_{k=0}^{\infty} \nu(k)x^k = \sum_{k=0}^{m} \nu(k)x^k = \frac{\sum_{k=0}^{m} kp_k}{\sum_{k=0}^{m} kp_k} = x\frac{G_0'(x)}{G_0'(1)}$$

Considérant un nœud choisi au hasard et un lien d'"arrivée" sur ce dernier, la distribution du nombre de ses voisins atteignables par les liens de "sortie" est l'objet de la troisième étape. La fonction génératrice correspondante de cette distribution est équivalente à $G_\nu(x)$ moins une puissance de x pour ne pas prendre en compte le nœud d'origine :

$$G_1(x) = \frac{G_0'(x)}{G_0'(1)}$$

Étant donné un ensemble de l nœuds au premier saut, p_k^l est la probabilité pour que ces derniers soient connectés à k autres nœuds au total (voisins au second saut par rapport au nœud d'origine). La distribution du nombre de voisins au second saut est alors calculée à partir de la probabilité d'avoir l voisins directs ayant k voisins au total comme schématisé par la quatrième étape sur la figure 4.8 :

$$p_k^l = \sum_{i=0}^{m} p_i \times p_k^i$$

La fonction génératrice associée est donc :

$$G_2(x) = \sum_{k=0}^{\infty}\sum_{i=0}^{m} p_i \times p_k^i x^k = \sum_{i=0}^{m} p_i \sum_{k=0}^{\infty} p_k^i x^k$$

Hors, $\sum_{k=0}^{\infty} p_k^i x^k$ est la fonction génératrice de p_k^i qui est facilement établie grâce à la propriété (2), c'est-à-dire $[G_1(x)]^k$ (étape (4) sur la figure 4.8)d'où :

$$G_2(x) = \sum_{i=0}^{m} p_i [G_1(x)]^i = G_0(G_1(x))$$

La distribution du nombre de nœuds au second saut est ainsi clairement établie (étape (5) sur la figure 4.8)

En répétant ce processus récursif, la fonction génératrice de la distribution du nombre de nœuds à une distance j se définit alors comme suit :

$$G^j(x) = \begin{cases} G_0(x) & \text{pour} \quad j = 1 \\ G^{j-1}(G_1(x)) & \text{pour} \quad j \geq 2 \end{cases} \tag{4.5}$$

Atteignabilité

Grâce à l'équation 4.5 et à la propriété (1), le nombre moyen de nœuds, z_j à une distance j du nœud d'origine est calculable d'une manière similaire à celle présentée dans [48]. Effectivement en appliquant

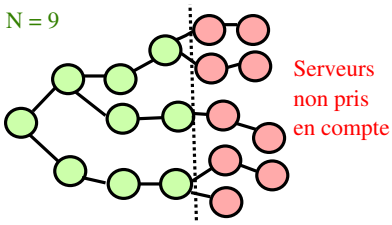


FIGURE 4.9 – Limitation du nombre maximal de serveurs atteignables

la formule (4.5) et grâce à un raisonnement par récurrence, le résultat suivant est obtenu :

$$z_j = \begin{cases} G_0'(1) & , \ j = 1 \\ G_0''(1) & , \ j = 2 \\ \left[\frac{z_2}{z_1}\right]^{j-1} z_1 & , \ sinon \end{cases} \tag{4.6}$$

La difficulté restante consiste donc à calculer z_1 et z_2 dans le modèle décrit ici. A partir de l'équation (4.4), les dérivées sont facilement obtenues en remarquant que p_0 est constant par rapport à la variable aléatoire k :

$$G_0'(x) = \frac{\alpha(m)}{m} \sum_{k=1}^{m} k x^{k-1} \qquad G_0''(x) = \frac{\alpha(m)}{m} \sum_{k=1}^{m} k(k-1) x^{k-2}$$

La formule (4.6) souligne très clairement l'avantage de ce nouveau modèle par rapport au précédent. En effet, la formule n'est plus récursive ce qui va très nettement réduire les temps de calcul et les différentes topologies ne sont pas générées car le modèle se base uniquement sur p_k pour mesurer le nombre de nœuds à une distance d quelconque.

Pour calculer l'atteignabilité, il reste encore à prendre en compte le risque β pour un nœud d'être attaqué. Dans le cas d'un réseau IRC, les serveurs occupent un rôle central et un seul serveur compromis est souvent catastrophique. Notamment, la configuration d'un serveur comprend généralement la liste de tous les autres pour parer aux problèmes de connectivité pouvant survenir ou faciliter les opérations de maintenance. Ainsi les réseau sera jugé opérationnel si aucun des nœuds n'est compromis. La proportion d'entre eux atteints est modérée par la probabilité d'avoir le réseau non découvert qui est $(1-\beta)^N$ c'est-à-dire qu'aucun des N nœuds n'est compromis. L'atteignabilité peut alors être complètement définie :

$$att(d) = \frac{(1-\beta)^N \times min(\sum_{j=1}^{d} z_j, N)}{N} \tag{4.7}$$

Il est important de remarquer que, contrairement au modèle précédent qui introduisait un ensemble de nœuds du réseau et limitait donc automatiquement la taille du réseau, il est nécessaire ici de mettre un seuil maximal. En effet, le modèle se base uniquement sur la probabilité que chaque nœud puisse être connecté à d'autres peu importe si il en existe d'autres ou non et il est donc primordial de ne pas prendre en compte plus de serveurs qu'il en existe tel qu'illustré dans la figure 4.9.

Atteignabilité des bots

L'atteignabilité des bots/équipements est équivalente à celle des serveurs. En effet, sans aucune connaissance à priori, une répartition aléatoire des bots sur les serveurs est justifiée. Étant donné un nombre total de bots B, le nombre d'entre eux atteints à une distance k est donc $bots(k) = att(k) \times B$. La proportion de bots est donc logiquement identique à l'atteignabilité : $att_{bots}(k) = \frac{bots(k)}{B} = att(k)$. Il est alors clair qu'étudier l'atteignabilité des serveurs est représentatif.

Atteignabilité moyenne

L'atteignabilité moyenne est l'atteignabilité moyenne de toutes les distances possibles sachant que la plus courte des distances est 1 et la plus longue est N lorsque l'arbre n'est en fait qu'une chaîne :

$$avg_att(k) = \frac{\sum_{k=1}^{N} att(k)}{N} \tag{4.8}$$

4.4.4 Délais

Le temps nécessaire pour envoyer une requête de supervision est un élément déterminant dans le choix d'une solution de supervision car les contraintes temporelles sont souvent fortes. Pour pouvoir évaluer le temps de transmission d'une requête il faut considérer deux paramètres : le temps t_h pour envoyer un message vers un équipement final et le temps t_s pour envoyer un message vers un autre serveur. Dans le cas d'une solution centralisée, la plateforme de supervision va envoyer successivement les messages vers l'ensemble des machines :

$$temps_{central} = C \times t_h \tag{4.9}$$

où C est le nombre de machines. Dans le cas d'un botnet IRC, les serveurs peuvent transmettre les messages aux serveurs avant de les transmettre aux machines. Considérons $C_{serveur}$, le nombre moyen d'équipements connectés à un même serveur, le temps total est donc composé du temps pour atteindre le dernier serveur à une distance d_{max} plus le temps pour ce serveur de transmettre les requêtes à l'ensemble des machines connectées :

$$temps_{botnet} = d_{max} \times t_s + C_{serveur} \times t_h \tag{4.10}$$

Ainsi, en déterminant l'atteignabilité à la distance d_{max}, le temps pour transmettre une requête à un nombre de machines déterminées peut être estimé.

4.5 Modèles P2P

Comme introduit dans le chapitre 1, les botnets de nouvelle génération ont tendance à fonder leurs architectures sur des réseaux P2P du fait de leur robustesse et leur efficacité intrinsèque. Les modèles qui vont être proposés devront donc pouvoir vérifier ces propriétés dans le cas précis de l'envoi d'une requête en broadcast, ce qui correspond à la fonctionnalité principale d'un botnet. De manière générale, un réseau P2P est totalement décentralisé sans aucun serveur une fois déployé. Les modèles vont donc explorer deux types de topologie P2P, c'est-à-dire deux types d'interconnexions entre les nœuds. Enfin, dans cette section, les termes nœuds, pairs et bots sont équivalents. Les publications relatives à ces travaux sont [2, 1].

4.5.1 Totalement maillé, Slapper

Le vers Slapper [67] est apparu pour la première fois en 2002. Il ne se limite pas au seul rôle d'un ver puisqu'il est accompagné d'un programme annexe sophistiqué permettant la construction d'un botnet. En effet, chaque machine infectée était contrôlable à distance par l'attaquant pour envoyer des courriels indésirables par exemple.

Fonctionnement général

Le botnet formé était en fait un réseau pair-à-pair de type totalement maillé, avec chaque machine connaissant toutes les autres. L'attaquant peut alors envoyer un message vers l'une d'entre elle spécifiquement par l'intermédiaire des autres. Pour cela, un nombre maximal de sauts est défini. Lorsqu'une machine reçoit le message, si le nombre de sauts est zéro alors le message est directement envoyé à la destination puisque le réseau est entièrement maillé. Sinon plusieurs nœuds choisis au hasard sont sélectionnés pour transmettre le message et dans ce cas, le nombre de sauts est décrémenté. Ce protocole permet à l'attaquant de rester caché puisque plusieurs nœuds intermédiaires sont utilisés. Il se préserve ainsi de toute détection par une sonde car aucune machine n'est capable d'identifier si celle qui lui a envoyé le message est l'émetteur initial, c'est-à-dire l'attaquant. Enfin pour permettre une réponse de la part de la destination, chaque machine qui route un message garde en mémoire l'émetteur de ce message ainsi qu'un identifiant contenu dans le message. Cela permet alors de router la réponse en sens inverse. Bien qu'il soit basé sur le protocole UDP, une autre fonctionnalité de Slapper est qu'il propose un mécanisme d'acquittement permettant d'améliorer les performances du protocole.

Broadcast

Dans le contexte de la configuration de masse et des botnets, Slapper est doté d'un mécanisme de broadcast appelé *broadcast segmentation*. Même si le contrôleur ou l'attaquant était capable d'envoyer directement un message à tous les hôtes du réseau formé par Slapper, cela pose deux problèmes :

- passage à l'échelle : initialisation d'une connexion et échange avec chaque machine,
- sécurité : le superviseur, c'est-à-dire le nœud d'origine, est forcément l'émetteur du message.

Pour palier ces problèmes, le superviseur décide d'envoyer le message à deux machines choisies au hasard. Celles-ci répètent la même procédure et ainsi de suite. De manière à arrêter la propagation infinie du message, un nombre de sauts maximal est fixé. Cependant comme les nœuds sont choisis au hasard, il est clairement possible que certains nœuds ne reçoivent jamais le message. La figure 4.10(a) illustre ce mécanisme où les nœuds sont visuellement placés de la même manière que Chord afin de faciliter la comparaison dans la section suivante. De plus la numérotation des nœuds n'est en aucun cas corrélé à la construction du réseau comme c'était le cas pour le premier modèle IRC. Le seul intérêt de celle-ci est de distinguer les différents nœuds et correspond en réalité à une adresse IP.

Le contrôleur se situe sur le nœud 0 et il envoie un message à deux pairs aléatoires : 4 et 8. Ces derniers répètent le même processus. Le non déterminisme de Slapper est présenté par l'intermédiaire de deux cas de figures possibles pour ce deuxième saut car les nœuds sont choisis au hasard. Ainsi, il est impossible d'être sûr d'atteindre tous les nœuds quelque soit le nombre d'itérations. Par exemple, tous les nœuds sont atteints à partir du troisième saut présenté sur la figure 4.10(a) mais cela ne signifie pas que ce sera toujours le cas.

Comme chaque nœud en choisit deux autres, le facteur de branchement de Slapper est fixé à $m = 2$. Dans une optique plus générale, celui-ci sera paramétrable dans le modèle dévoilé ici.

Supposons que t nœuds aient déjà reçus le message sur un total de N nœuds. Ils vont alors transmettre c messages identiques à leur tour à d'autres pairs. On peut alors déterminer la probabilité $p(N,t,c,j)$ d'envoyer ces messages à j pairs qui ne les ont pas encore reçus :

$$p(N,t,c,j) = \frac{C_{N-t}^{j} \times \Gamma_{t+j}^{c-j}}{\Gamma_{N}^{c}} \tag{4.11}$$

avec $\Gamma_n^k = C_{n+k-1}^k$ la combinaison avec répétition de k éléments parmi n.

Dans cette équation, le premier terme au numérateur est le nombre de possibilités de choisir les j nœuds nouvellement contactés parmi l'ensemble c'est-à-dire $N-t$. Parmi l'ensemble des c messages, il

y en a donc j qui sont utilisés pour être envoyer à ces nœuds précisément. Le second terme représente le nombre de destinations possibles des $c-j$ messages restants. Ces derniers peuvent alors être transmis à n'importe quel pair parmi ceux déjà contactés auparavant ainsi que ceux nouvellement contactés soit $t+j$. Le dénominateur représente quant à lui le nombre total de possibilités de choisir la destination des c messages parmi l'ensemble des nœuds. Cette probabilité n'est calculable que pour un nombre cohérent de pairs c'est-à-dire pour $j \leq N-t$.

Le fonctionnement du broadcast implique qu'au d^{e} saut, le nombre de messages dupliqués reçus soit m^d. Cependant, la disponibilité d'un nœud et donc sa capacité à participer au réseau est contrainte par le facteur $\alpha(m)$. Ainsi, chacun des nœuds a une probabilité $\alpha(m)$ de fonctionner normalement et le nombre de messages expédiés réellement au i^{e} saut est donc :

$$msg_d = (m \times \alpha(m))^d$$

Ce nombre limite donc aussi le nombre de nouveaux nœuds qui vont recevoir l'information. Ce dernier est également limité par le nombre de nœuds qui n'ont pas encore reçu de message. On peut donc définir le nombre maximal de nouveaux pairs pouvant recevoir le message :

$$pairs_d = min(msg_d, N - \overline{att}(N,m,d-1))$$

où $\overline{att}(N,m,d-1)$ représente le nombre de nœuds ayant reçu un message au saut précédent, $d-1$, c'est-à-dire l'atteignabilité absolue et équivalent au nombre t dans les explications précédentes. Pour le cas $d=0$, seul le nœud d'origine est considéré comme déjà atteint d'où $att(N,m,0)=1$.

Grâce à la formule (4.11), le nombre moyen de nouveaux nœuds atteints à une distance d est évalué en tenant compte des différents cas possibles c'est-à-dire de 0 à max_{msg} :

$$r(N,m,d) = \sum_{k=0}^{pairs_d} p(N, att(N,m,d-1), msg_d, k) \times k$$

L'atteignabilité absolue s'obtient de manière récursive en gardant à l'esprit que l'atteignabilité à une distance d est le nombre de nouveaux pairs recevant le message et ceux l'ayant déjà reçus :

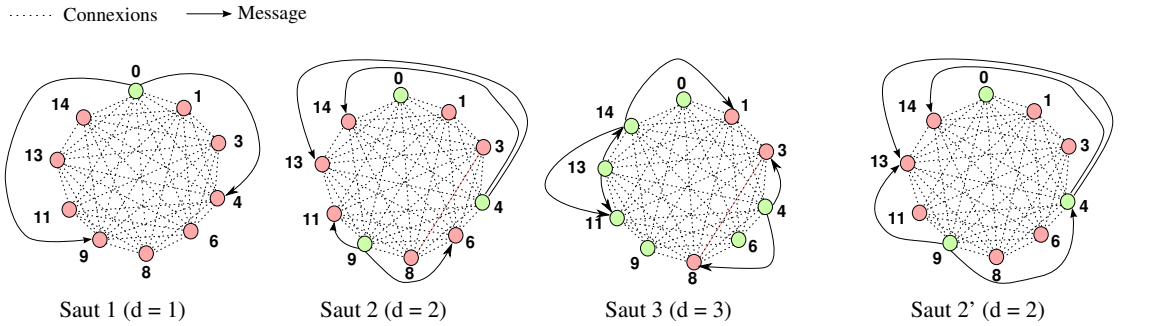
$$\overline{att}(N,m,d) = \begin{cases} 1 & \text{si } d=0 \\ \overline{att}(N,m,d-1) + r(N,m,d) & \text{sinon} \end{cases} \tag{4.12}$$

Enfin, de manière similaire au second modèle IRC, l'atteignabilité doit être tempérée par le fait qu'une partie des nœuds soit compromise. Effectivement, dès qu'un nœud est compromis, l'attaquant est capable de découvrir tout le réseau du fait de l'architecture totalement maillée. Cela est estimé par le paramètre β grâce auquel la formule finale d'atteignabilité est :

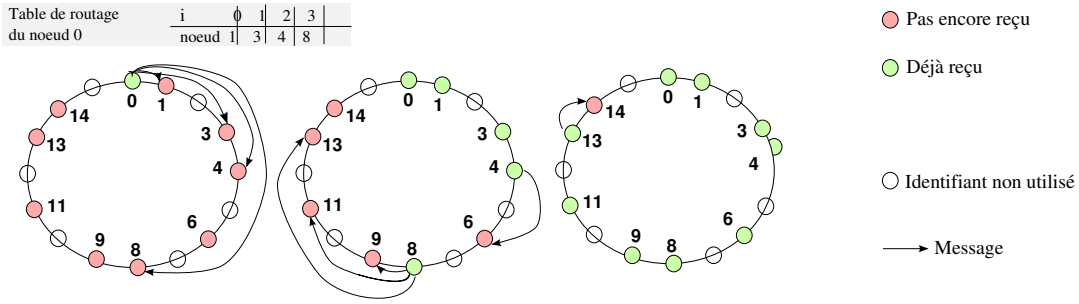
$$\overline{att}(N,m,d) = \begin{cases} 1 & \text{si } d=0 \\ \overline{att}(N,m,d-1) + (1-\beta)^N r(N,m,d) & \text{sinon} \end{cases} \tag{4.13}$$

4.5.2 Table de hachage distribuée, Chord

Un réseau totalement maillé tel que Slapper le construit présente des limites en termes de passage à l'échelle car chacun doit connaître tous les autres et identifier l'ensemble des machines le composant est donc aisé en espionnant une seule d'entre elles. Une évolution en cours des logiciels malveillants est l'utilisation des réseaux pair-à-pair structurés tel que ceux reposant sur une table de hachage distribuée.



(a) Réseau totalement maillé (Slapper), la numérotation et le placement spécifique des nœuds n'est là que pour faciliter la comparaison avec Chord



(b) Table de hachage distribuée (Chord)

FIGURE 4.10 – Mécanisme de broadcast dans les réseaux P2P à partir du nœud 0

Ce type de réseau P2P est prédominant aujourd'hui pour des raisons de robustesse et d'efficacité. Cependant, cette popularité a entraîné la conception de beaucoup de réseau de ce genre ayant chacun leurs caractéristiques particulières (degré de redondance, mécanisme d'indexation des données...). Par conséquent, le modèle présenté dans cette section reposera sur Chord [3] qui est un des pionniers dans ce domaine et décrit les fondements de ce type d'architecture. Chord est donc assez générique pour approximer l'ensemble des réseaux P2P de ce genre et a montré de bonnes performances dans de nombreuses études [3, 4].

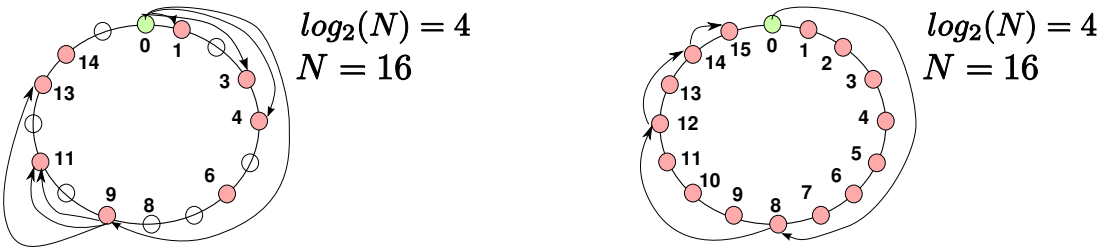
Fonctionnement général

Sur un réseau Chord, chaque nœud ou pair participant est associé à un identifiant tel que $0 \leq id < N_{MAX}$ où $N_{MAX} = 2^k, k \in \mathbb{I}^{+*}$. N_{MAX} est le nombre maximal de nœuds participants ainsi que la taille de l'espace des identifiants. Contrairement à un nœud Slapper qui connaît l'ensemble des autres nœuds, un nœud Chord n'a qu'une vue partielle. La taille de la table de routage d'un nœud est de $log_2(N_{MAX})$. La *ième* entrée du nœud d'identifiant id_1 contient le nœud dont l'identifiant id_2 est le premier successeur de $id_1 + 2^{i-1}$, c'est-à-dire :

$$id_1 + 2^{i-1} \leq id_2 \wedge (\nexists id'_2,\ id'_2 < id_2 \wedge id_1 + 2^{i-1} < id'_2) \tag{4.14}$$

La figure 4.11(a), où les nœuds sont représentés selon le sens des aiguilles d'une montre, met en lumière deux tables de routage, celles des nœuds 0 et 9.

En réalité, les identifiants résultent de l'application d'une fonction de hachage sur les adresses IP. Lorsqu'un nœud id_1 cherche à contacter le nœud id_2, il commence par le rechercher dans sa table de routage. Si ce n'est pas le cas, il repère le nœud dans sa table de routage dont l'identifiant est inférieur



(a) Tables de routage

(b) Localisation du nœud 15 par le nœud 0

FIGURE 4.11 – Fonctionnement général de Chord

à id_2 et tel qu'il soit le plus grand possible. Ce nœud est alors contacté pour localiser id_2 qui répète le même processus et ainsi de suite. Sur l'exemple de la figure 4.11(a), le nœud 0 localisera le nœud 13 par l'intermédiaire de 9. Un tel mécanisme de routage permet, dans le pire des cas, de localiser un autre pair en $log_2(N_{MAX})$ sauts. Ce cas est illustré sur la figure 4.11(b).

Il est important de voir que, dans un esprit de simplification, le nœud 0 est considéré ici comme le nœud d'origine, ce qui n'est pas forcément le cas. Cependant, il est évident que le système fonctionne exactement de la même façon quelque soit le nœud et cela correspond à une simple transposition des identifiants. En pratique, un calcul modulo est effectué pour rechercher le i^e successeur du nœud *id* : $id_1 + 2^{i-1}\ mod(N_{MAX})$.

Des détails plus précis sont disponibles dans [3]. D'un point de vue plus général, ce type de réseau fut conçu pour le partage de fichiers eux-mêmes identifiés par un numéro calculé grâce à la fonction de hachage. Ainsi, un nœud et la ressource qu'il partage n'auront pas forcément le même identifiant. Cependant, le premier successeur de l'identifiant de la ressource l'indexe et permet alors de faire la correspondance. Enfin, comme précédemment introduit, il existe une multitude de réseaux P2P structurés mais Chord a l'avantage d'être assez générique sans ajouter de mécanismes supplémentaires plus avancés. Beaucoup d'entre eux sont notamment dédiés aux phases de déploiement et de maintenance du réseau alors qu'ici, seul le réseau est supposé déployé. De plus, la plupart des autres types de réseaux et les plus performants d'entre eux sont basés sur des tables de hachage et le broadcast dans une telle structure est équivalent quelque soit le réseau sous-jacent.

Broadcast

Pour rester le plus général possible, l'étude présentée ici suppose que les identifiants des nœuds soient aléatoirement et uniformément répartis dans cet espace et que la distance moyenne entre deux identifiants consécutifs soit d d'où $N_{MAX} = N \times d$.

Par conséquent, les premières entrées de la table de routage sont identiques car le successeur est toujours le même. En moyenne, la première entrée i qui est différente est celle telle que $2^{(i-1)} > d$. Le nombre total d'entrées différentes est donc :

$$\#routes = log_2(N_{MAX}) - log_2(d) = log_2\left(\frac{N_{MAX}}{d}\right) = log_2(N) \tag{4.15}$$

Il est important de noter que ce nombre correspond au facteur de branchement maximal m.

Les auteurs de [1] suggérèrent un mécanisme de broadcast efficace grâce auquel un nœud de Chord ne reçoit qu'une seule fois le message. C'est cette méthode qui va servir à diffuser une requête au sein du botnet. Dans ce cas, un message contient une limite d'exploration. Le superviseur amorce le processus en envoyant le message à chaque pair différent de sa table de routage en spécifiant une limite d'exploration

pour chacun d'eux qui est en fait l'entrée suivante de cette table (ou le nœud superviseur pour la dernière entrée). Chaque nœud répète ce processus sauf qu'un nœud p n'enverra jamais un message au nœud dont l'identifiant est supérieur à la limite d'exploration. De même, le dernier nœud qui vérifie cette condition se verra attribuer une limite d'exploration identique à celle du nœud p.

Ce mécanisme est illustré sur la figure 4.10(b). Le contrôleur est le nœud 0, il envoie le message aux différentes entrées de sa table de routage : 1, 3, 4 et 8 avec respectivement comme limite d'exploration 3, 4, 8 et 0. Il est important de souligner que cette limite est exclusive et c'est pourquoi, à la deuxième itération (deuxième saut), le nœud 4 transmet le message au nœud 6 seulement car sa limite est 8. La table de routage d'un nœud contient au plus $log(N_{MAX})$ entrées. Chaque nœud transmettra donc le message broadcast à $log(N_{MAX})$ autres nœuds tout au plus. Le facteur de branchement est directement dépendant de N_{MAX}, c'est-à-dire $m = log(N_{MAX})$ avec normalement N_{MAX} qui est une puissance de 2.

A partir de ces différentes hypothèses et conclusions, l'algorithme 1 calcule, pour un identifiant donné, le nombre de sauts nécessaires pour l'atteindre à partir du nœud 0. Cet algorithme ne peut être appliqué que pour les identifiants valides c'est-à-dire que $\frac{id}{d}$ doit être une valeur entière et positive. Si cet identifiant est une puissance de 2 alors le nœud peut être atteint directement (ligne 5), ce qui correspond à la puissance de 2 dans la formule (4.14).

Cette formule signifie en fait qu'on sélectionne un successeur. Ainsi s'il n'y a aucun identifiant valide entre l'identifiant du nœud à contacter et l'identifiant théorique le plus grand, c'est-à-dire une puissance de 2 (ligne 10), alors le nœud visé est également joignable en un seul saut (ligne 11).

Dans tous les autres cas, il existe un nœud intermédiaire. Il est donc nécessaire de calculer le nombre de sauts pour ce nœud intermédiaire et d'appliquer ce processus successivement jusqu'à pouvoir atteindre directement le nœud concerné. Pour cela, la boucle itère une nouvelle fois en "avançant" dans les identifiants grâce aux lignes 14 et 10.

Une fois de plus, le nœud 0 est toujours considéré comme le nœud d'origine (le superviseur) mais étudier les autres cas est inutile car ils équivalent à transposer l'ensemble des identifiants des nœuds.

Algorithme 1 Nombre de sauts

1: $id = d \times k$ l'identifiant du nœud
2: $entier(x)$ retourne la partie entière de x
3: $l = log(id)$
4: **si** $l \in \mathbb{N}$ **alors**
5: retourner 1
6: **sinon**
7: $total \leftarrow 0$
8: $compteur \leftarrow 1$
9: **tantque** $l \neq entier(l)$ **faire**
10: $total \leftarrow total + 2^{entier(l)}$
11: **si** $id - total < d$ **alors**
12: retourner $compteur$
13: **sinon**
14: $l \leftarrow log(id - total)$
15: **finsi**
16: **fin tantque**
17: **finsi**

Cet algorithme est exécuté pour chaque identifiant possible et permet de définir simplement l'atteignabilité :

Définition 20 : *L'atteignabilité au ieme saut avec N nœuds au total est le nombre de nœuds à une distance i du nœud d'origine.*

De manière à pouvoir comparer les différents modèles, l'intégration des différents types de pannes se révèle obligatoire. Dans le pire des cas, il y a une panne qui empêche la communication normale avec un nœud au premier saut. Dans ce cas, un autre nœud avec un identifiant plus petit est choisi pour le remplacer. Celui-ci se retrouve alors plus près du nœud injoignable précédemment et est donc aussi capable de le contacter. On réitère ce processus en considérant que si un nœud est à une distance h alors il y a au plus h pannes. Pour chacune d'entre elles, un saut supplémentaire sera nécessaire dans le pire cas. La probabilité $p(f < h)$ d'avoir $f < h$ pannes est définie par une loi binomiale. Il suffit alors d'évaluer cette probabilité puis de la multiplier par le nombre de nœuds atteints à la distance h de manière à rajouter ce résultat au nombre de nœuds atteints à une distance $h+f$. L'algorithme 2 fait donc ces différents calculs par ordre croissant des distances h.

Algorithme 2 Effets des pannes

n le nombre de nœuds à la distance h
$total \leftarrow 0$
MAX_HOP est la distance maximale à laquelle se situe un nœud par rapport au superviseur
$\overline{att}[x]$ est le nombre moyen de nœuds atteints à la distance x (atteignabilité absolue)
pour $h \leftarrow 1$ toMAX_HOP **faire**
  pour $i \leftarrow 1$ to h **faire**
    $p \leftarrow n \times *C_h^i * \alpha(m)^{k-i} * (1-\alpha(m))^i$
    $\overline{att}[h+i] \leftarrow \overline{att}[h+i] + p$
    $total = total + p$
  fin pour
  $\overline{att}[h] \leftarrow \overline{att}[h] - total$
fin pour

Contrairement à Slapper et à un réseau IRC, un nœud compromis par un attaquant n'implique pas que l'ensemble du réseau soit inutilisable. En effet, seule la capacité d'espionnage des communications est prise en compte, c'est-à-dire qu'un attaquant n'est pas supposé doté des capacités suffisantes pour contrôler le nœud et donc injecter des requêtes (problèmes pouvant être réglés en ajoutant une couche de cryptographie). Dans le pire des cas, un attaquant peut découvrir tout le réseau s'il a compromis $n_compromis = \frac{N}{log(N)}$ nœuds car chacun des nœuds connaît $log(N)$ autres nœuds. Bien entendu, le pire cas est envisagé car en réalité, il y a de fortes chances d'obtenir de l'information redondante. En supposant que la probabilité qu'un nœud soit compromis est β, la loi binomiale est utilisée pour calculer la probabilité que le réseau soit entièrement vulnérable c'est-à-dire qu'il y ait *n_compromis* nœuds compromis. Cette loi est approximée par celle de Poisson avec $\lambda = N \times \beta$. La probabilité que le réseau soit entièrement découvert est donc :

$$e^{-\lambda} \sum_{i=n_compromis}^{N} \frac{\lambda^i}{i!}$$

Les différentes valeurs d'atteignabilité sont alors multipliées par la probabilité inverse qui permet de prendre en compte la robustesse du système si un attaquant essaie de détourner l'usage du réseau à son propre compte.

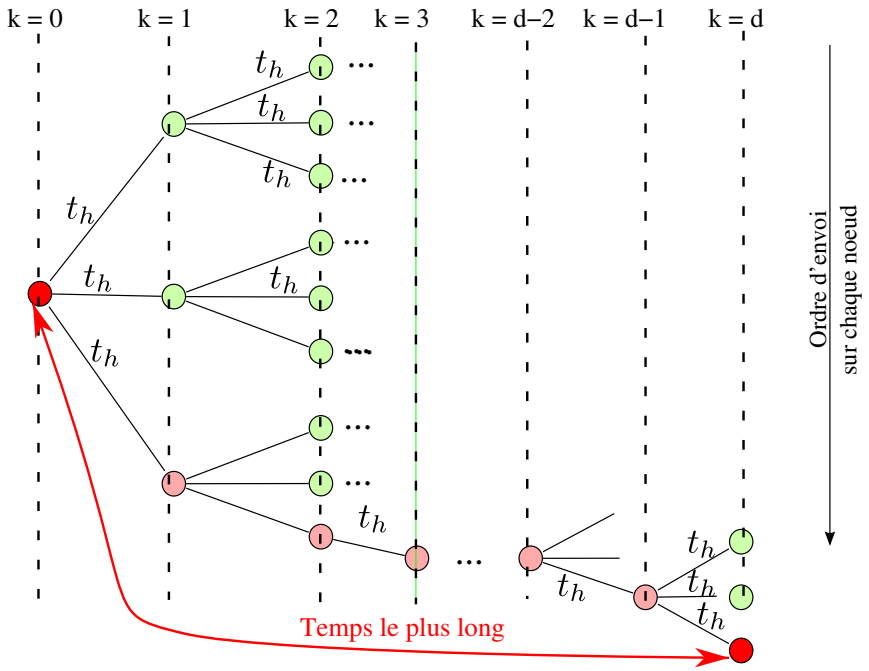


FIGURE 4.12 – Délai de diffusion d'une requête broadcast dans les réseaux P2P ($m = 3$)

4.5.3 Délais

Étant donné le temps t_h pour envoyer un message d'un nœud à l'autre et une distance de d sauts, le temps nécessaire pour transmettre le message aux nœuds du premier saut est $m \times t_h$, c'est à dire le temps individuel par bot multiplié par leur nombre (le facteur de branchement). Ce processus peut être réitéré pour chaque saut et la formule suivante est alors obtenue :

$$m \times k \times t_h \tag{4.16}$$

La figure 4.12 illustre ce calcul pour une distance d. Il est à noter que l'arbre représenté ne peut servir qu'au calcul des délais. En effet, dans le cas de Slapper, un même nœud peut correspondre à plusieurs sur la figure et dans le cas de Chord, un pair n'envoie pas systématiquement m messages du fait de la limite d'exploration (m est le facteur de branchement maximal). Ainsi, le cas considéré ici est le pire cas pour Chord. Dans le cas du calcul des délais, cette figure montre clairement que le nœud recevant la requête en dernier est celui issu du chemin composé des nœuds recevant à chaque fois le message en dernier (pour chaque k). En conséquence, les délais sont bien égaux à la formule (4.16)

4.6 Bilan

Comme il existe plusieurs types d'architectures de botnets, plusieurs modèles ont été créés :
- des modèles IRC (ou pseudo-centralisés),
- des modèles P2P.

Ces derniers possèdent différents paramètres pour modéliser la connectivité des nœuds, leur disponibilité ou encore leur résistance face aux attaques. Ainsi, le chapitre 7 aura pour objectif de mesurer les

performances des botnets dans de nombreux cas et selon plusieurs configurations de manière à réaliser une étude complète et complémentaire des autres modèles existants. En effet, ceux présentés dans ce chapitre cherchent à déterminer le nombre de bots joignables lors de l'envoi d'une requête de broadcast contrairement à la plupart des modèles existants. Bien que cela puisse sembler restrictif à première vue, une telle requête est l'élément essentiel de la phase d'utilisation d'un botnet pour coordonner une action simultanée de plusieurs milliers voire centaines de milliers de machines en même temps.

Les travaux les plus analogues sont [37, 26]. Le premier définit différentes métriques complémentaires à celles exposées ici telles que le diamètre du botnet (distance entre les bots les plus éloignés), la bande passante utilisable, la redondance dans le réseau ou encore le partitionnement du botnet (plus grand nombre de bots interconnectés). [26] est quant à lui beaucoup plus proche des modèles détaillés dans ce chapitre en calculant notamment les chemins les plus courts entre des bots ou encore l'atteignabilité définie similairement à celle exposée dans les sections précédentes. Cependant, seuls les réseaux P2P sont étudiés et la publication de ces travaux suivit de peu ceux présentés ici. Enfin, les deux articles [37, 26] se limitent à des simulations pour générer des graphes et en étudier les propriétés tandis que les modèles précédemment détaillés ne nécessitent pas de simuler plusieurs topologies et de faire la moyenne des résultats. En effet, ces modèles généraux permettent directement d'établir vers quelle valeur la moyenne tenderait dans le cas de la réalisation d'une infinité de simulations.

5

Nouvelles représentations et méthodes pour la rétro-ingénierie

Sommaire

5.1 Introduction . . . 107
5.2 Mini protocole . . . 109
5.3 Distances entre deux messages . . . 110
5.3.1 Longueur d'un message . . . 110
5.3.2 Distribution des caractères . . . 111
5.3.3 Positions des caractères . . . 113
5.3.4 Positions pondérées des caractères . . . 114
5.3.5 Distribution ordonnée des caractères . . . 115
5.3.6 Alignement de séquences . . . 117
5.4 Partitionnement des données . . . 118
5.4.1 Définition du problème . . . 118
5.4.2 Méthodes utilisées . . . 119
5.4.3 Méthode globale . . . 120
5.5 Architecture générale . . . 120
5.5.1 Fonctionnement . . . 121
5.6 Bilan . . . 123

5.1 Introduction

Le chapitre 2 a dressé un état de l'art des différentes techniques de rétro-ingénierie des protocoles dont une étape essentielle est la compréhension des messages échangés. Alors que certaines techniques essayent de retrouver l'intégralité de la syntaxe d'un message, certaines se limitent à déterminer seulement le type de celui-ci. Comme expliqué dans le chapitre 2, ce type peut varier selon sa granularité mais il est généralement accepté qu'un type désigne une classe d'actions particulières : télécharger, envoyer des données, initialiser, acquitter... En se basant sur ces types, la construction de la machine à états du protocole est réalisable.

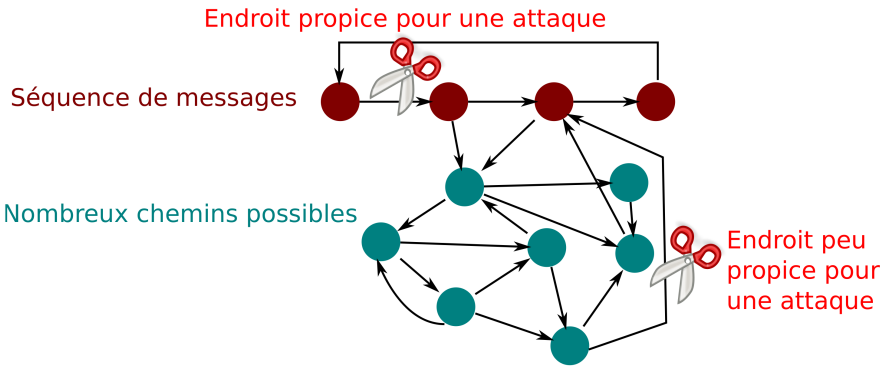


FIGURE 5.1 – Attaque ciblées à différents moments de la communication

Comme la rétro-ingénierie vise à déterminer automatiquement ces éléments (types, machines à états) d'un protocole inconnu, aucune connaissance n'est disponible à priori forçant l'ensemble des méthodes à être totalement non supervisées. Par ailleurs, les logiciels malveillants emploient des protocoles spécifiques ou des adaptations de protocoles connus et leur étude est un réel atout pour mieux les appréhender. Par exemple, la machine à états aide à identifier les étapes essentielles de la communication d'un botnet et permet donc de mieux adapter ses défenses. De plus, les techniques de fuzzing qui envoient des messages plus ou moins aléatoires pour détecter des failles peuvent être mieux ciblées en envoyant le message à un moment critique pour le bot comme l'illustre la figure 5.1 où il y a deux types d'envois de messages (une séquence et un envoi plus aléatoire). Il semble alors plus opportun d'attaquer la communication entre deux états tels qu'il n'existe aucun chemin alternatif possible. Par ailleurs, de nombreuses techniques de défense contre les botnets du chapitre 1 reposent sur la participation active à celui-ci en injectant des requêtes et nécessitent donc la compréhension du protocole.

Les méthodes de rétro-ingénierie actuelles s'appuient sur les messages eux-mêmes ou sur la manière dont ils sont analysés et interprétés par la machine. Cependant, ce chapitre s'efforcera d'introduire une nouvelle méthode de découverte des types de messages basée exclusivement sur les traces réseaux car exécuter un malware comporte naturellement des risques et un tel logiciel est capable de biaiser les outils d'observation installés sur une machine.

La difficulté d'une telle approche est la relative pauvreté sémantique des messages car, sans la grammaire du protocole, leur interprétation est impossible. Par le passé, peu d'outils de ce type ont été proposés et seront comparés, dans la conclusion, à la nouvelle méthode présentée dans ce chapitre.

Cette dernière repose d'un côté sur l'élaboration de représentations simplifiées des messages et sur l'application de techniques récentes de classification d'un autre côté. Les contraintes strictes à respecter sont les suivantes :

- protocole totalement inconnu,
- rétro-ingénierie de l'application impossible, c'est-à-dire que l'instrumentation d'une machine dédiée est proscrit,
- le nombre de types de messages est inconnu,
- aucun jeu d'apprentissage disponible.

De plus, l'utilisation de chiffrement dans les messages peut énormément altérer le bon fonctionnement des outils de rétro-ingénierie. A l'instar de Storm [31] effectuant un chiffrement octet par octet grâce à un simple OU exclusif (XOR), la contrainte souple suivante est ajoutée : la découverte des types doit pouvoir résister à un chiffrement par XOR.

Enfin, la publication [1] est relatif à ce chapitre.

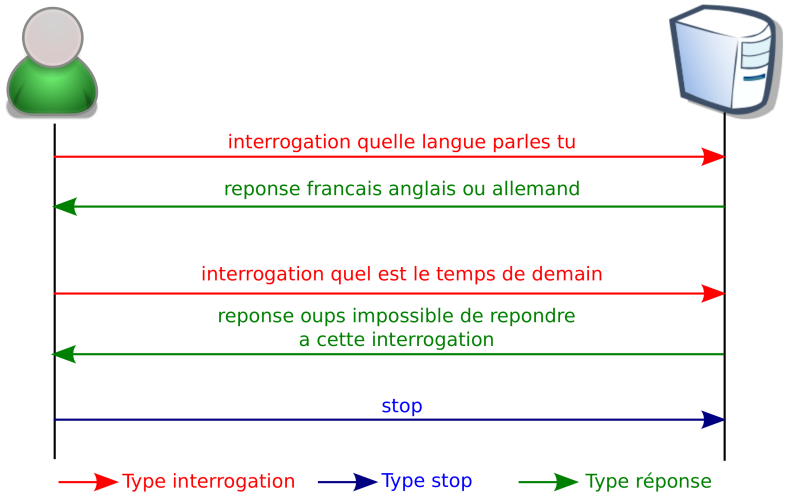


FIGURE 5.2 – Illustration du mini protocole utilisé

5.2 Mini protocole

Un protocole simple permettra d'illustrer clairement les différentes métriques. Ce protocole est textuel et spécifie trois types de messages : un pour poser une question (type *interrogation*), un pour répondre à une question (type *réponse*) et un pour terminer la communication (type *stop*). Une communication débute à la première requête envoyée et se termine avec *stop*. De plus, chaque participant a un rôle spécifique et inéchangeable pour une session :

– le client posant des questions,
– le serveur répondant aux questions.

Un message commence par le type en toute lettre suivi d'un espace et de la question ou la réponse en clair. Pour des raisons de simplicité, seuls les caractères ASCII (sans accents) sont utilisés et aucune ponctuation n'est possible. Un message *stop* peut quant à lui avoir un contenu optionnel tel que la raison de la terminaison de la session par exemple. La grammaire est donc décrite de la manière suivante :

```
message = (type SP phrase) / stop
type = "interrogation" / "reponse"
stop = "stop" [SP phrase]
phrase = 1*(alpha / chiffre / SP)
alpha = A-Z
chiffre = 0-9
SP = " "
```

La figure 5.2 illustre un exemple de messages échangés où un utilisateur pose deux questions au serveur avant de se déconnecter. Ces messages illustreront les différents exemples de ce chapitre et seront indicés et « regroupés » par type de la manière suivante :

– m_0 : *interrogation quelle langue parles tu*,
– m_1 : *interrogation quel est le temps de demain*,
– m_2 : *reponse francais anglais ou allemand*,
– m_3 : *reponse oups impossible de repondre a cette interrogation*,
– m_4 : *stop*.

message	0	1	2	3	4
0	0	6	1	20	33
1	6	0	7	14	39
2	1	7	0	21	32
3	20	14	21	0	53
4	33	39	32	53	0

TABLE 5.1 – Distances entre les messages basées sur la longueur — Pour chaque message en ligne, les cases rouges indiquent une incohérence de distance par rapport à deux autres messages

5.3 Distances entre deux messages

Dans l'objectif d'un classement rapide de nombreux messages, cette section vise à déterminer des représentations abrégées d'un message m pour calculer simplement et rapidement la distance $d(m_1, m_2)$ entre deux d'entre eux. Pour pouvoir évaluer l'efficacité d'une distance, cette section cherchera à réduire au maximum le nombre d'incohérences se définissant de la manière suivante :

Définition 21 : *Il existe une incohérence de distance entre $d(m,n)$ et $d(m,k)$ si et seulement si les messages m et n sont de même type distinct de celui de k et tel que $d(m,n) > d(m,k)$.*

5.3.1 Longueur d'un message

La représentation concise extrême d'un message m se résume à sa longueur $l(m)$. Elle possède les propriétés suivantes :

– elle résiste à un chiffrage par un *XOR* : $l(m) = l(m \oplus k)\ \forall\ key\ k$,
– son calcul est très rapide.

La distance entre deux messages m_1 et m_2 est alors représentée par la différence absolue entre les deux longueurs :

$$dist(m_1, m_2) = |l(m_1) - l(m_2)|$$

Les distances entre les messages de la figure 5.2 figurent dans le tableau 5.1. Pour un message, les cases rouges indiquent une incohérence de distance par rapport à deux autres messages. Par exemple, le message m_0 de type *interrogation* a une distance plus élevée avec le second de même type qu'avec le troisième de type *réponse*. Ces différences sont remarquables aussi bien par ligne que par colonne puisqu'une matrice de distance est symétrique mais seules les incohérences en ligne sont reportées dans ce type de tableau pour garder une bonne lisibilité. Le type *stop* est clairement identifié par sa faible longueur car, comme mentionné par la grammaire, le mot clé *stop* n'est pas forcément suivi par un autre contenu. Cependant, si la raison de la déconnexion est ajointe au message, en déduire le type sera beaucoup plus ardu. De même, la longueur des questions ou des réponses n'est en aucun cas imposée et accentue le faible intérêt de l'utilisation de la longueur comme caractéristique significative d'autant plus que, dans les protocoles réels, beaucoup de champs sont de taille variable, impliquant des messages de longueurs diverses. Par exemple, la distance entre les messages 0 et 2 est très faible malgré leurs types distincts. Plus généralement, cette métrique distingue uniquement les grandes classes de messages souvent existantes dans la plupart des protocoles telles que :

– les messages contenant de l'information dont la taille est assez longue et très variable,

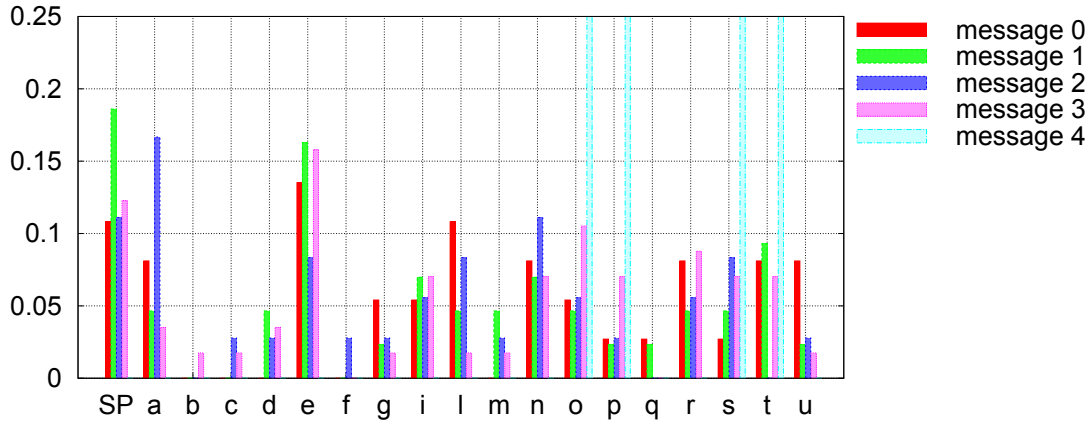


FIGURE 5.3 – Distribution des caractères entre tous les messages — SP est le caractère d'espacement

– les messages d'acquittements ou d'erreurs souvent assez courts.

5.3.2 Distribution des caractères

Distribution

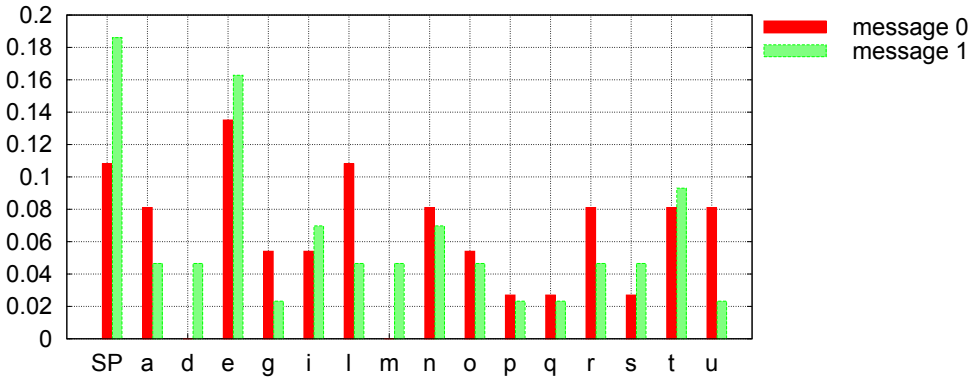
La distribution de caractères $car(m)$ d'un message m représente la fréquence d'apparition de chacun d'eux. Ainsi $car(m)_a = 0.2$ exprime que 20% du message m est composé du caractère « a ». Intuitivement, les messages de même type doivent comporter des informations du même genre et donc être composés avec des données similaires. Dans l'exemple, deux messages de type *interrogation* commencent inéluctablement par le mot clé correspondant suivi d'une question se construisant généralement à partir de pronoms interrogatifs tels que : *qui*, *que*, *quel*... De même, si la rétro-ingénierie repose sur des sessions d'un seul utilisateur, ce dernier aura tendance à formuler des requêtes semblables. Cette hypothèse est d'autant plus forte que les données introduites dans les messages par une machine ont une formulation encore plus systématique (noms d'hôte, adresses...).

Cette métrique est la distribution des caractères ou des valeurs d'octet et réduit ainsi la représentation d'un message à un vecteur de 256 composantes. De plus, elle peut être facilement compressée en estimant que les valeurs non représentées sont celles équivalant à 0 :

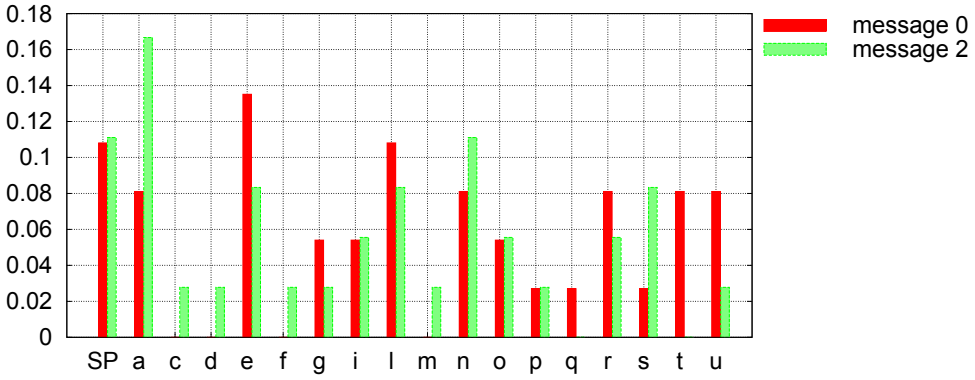
$$car^2(m)_i = car(m)_i,\ car(m)_i > 0$$

Ce format condensé permet de facilement retrouver toutes les valeurs originales au besoin.

Les histogrammes des fréquences des différents messages de l'exemple sont exposés sur la figure 5.3 mettant très visiblement en évidence la singularité du message 4 (*stop*) par rapport aux autres. Différencier les deux autres types de messages demeure néanmoins difficile. La figure 5.4 reprend les histogrammes de deux messages seulement à chaque fois. Les messages m_0 et m_1 sont de mêmes types et présentent bien une forte similarité. Cependant, les messages m_0 et m_2 se ressemblent également et se distinguent principalement par des caractères différents (fréquences égales à zéro). Il est donc nécessaire de définir une métrique arrivant à différencier de telles distributions.



(a) Message m_0 (interrogation) et message m_1 (interrogation)



(b) Message m_0 (interrogation) et message m_2 (réponse)

FIGURE 5.4 – Comparaison des distributions des caractères de deux messages particuliers — SP est le caractère d'espacement

Divergence de Kullback-Leibler

La distance entre deux distributions s'évalue de différentes sortes telles que la distance euclidienne :

$$dist(m_1, m_2) = \sqrt{\sum_{i}^{255} (car(m_1)_i - car(m_2)_i)^2}$$

Cependant, l'utilisation de la divergence Kullback-Leibler est particulièrement efficace. Elle mesure la différence de quantité d'information entre deux distributions. Son utilisation dans le cas présent accentue les différences en fonction du ratio entre les distributions. Ainsi la distance entre la probabilité 0.2 et 0.4 comparée avec celle entre 0.7 et 0.9 sera plus importante malgré une même différence absolue. La divergence de Kullback-Leibler entre deux distributions d^1 et d^2 se définit mathématiquement comme suit :

$$KL(d^1, d^2) = \sum_i d_i^1 log\left(\frac{d_i^1}{d_i^2}\right) \tag{5.1}$$

La base du logarithme n'est pas spécifiquement fixée. Dans ce cadre, la valeur habituelle de deux est employée. Ainsi, cette distance est une quantité en bits.

Alors que les algorithmes de classification reposent en général sur une notion de distance, cette divergence n'en n'est pas une puisqu'elle n'est pas symétrique, c'est-à-dire $KL(d^1, d^2) \neq KL(d^2, d^1)$. Sa

message	0	1	2	3	4
0	0.000	1.812	3.721	2.349	15.443
1	1.812	0.000	3.177	1.171	14.926
2	3.721	3.177	0.000	2.490	19.425
3	2.349	1.171	2.490	0.000	12.215
4	15.443	14.926	19.425	12.215	0.000

TABLE 5.2 – Divergence de Kullback-Leibler symétrique entre les messages (lissage des distributions des caractères avec $\varepsilon = 10^{-6}$) — Pour chaque message en ligne, les cases rouges indiquent une incohérence de distance par rapport à deux autres messages

définition symétrique est alors généralement appliquée :

$$KL^2(d^1, d^2) = \frac{KL(d^1, d^2) + KL(d^2, d^1)}{2} \tag{5.2}$$

En conséquence, la distance reposant sur la distribution de caractères est spécifiée par :

$$dist(m_1, m_2) = \sum_{i=0}^{255} car(m_1)_i log\left(\frac{car(m_1)_i}{car(m_2)_i}\right) + \sum_{i=0}^{255} car(m_2)_i log\left(\frac{car(m_2)_i}{car(m_1)_i}\right) \tag{5.3}$$

Le dernier problème demeurant est le domaine de définition de cette distance. Effectivement, elle risque de calculer une division par zéro quand il existe $car(m_1)_i$ ou $car(m_2)_j$ valant zéro. Une solution envisageable consiste à lisser les données pour transformer les fréquences égales à zéro en de très petites valeurs ε. Dès lors, réduire légèrement les autres fréquences s'avère essentiel pour garder une somme des fréquences égale à un :

$$car^3(m)_i = \begin{cases} car(m)_i - \lambda\varepsilon & , \; si \; car(m)_i > 0 \\ \varepsilon & sinon \end{cases} \tag{5.4}$$

Dans cette équation λ est le nombre de fréquences égales à zéro de la distribution d'origine. Le lissage des distributions n'est pas seulement spécifique à la divergence de Kullback-Leibler et réduit les effets trop négatifs des valeurs nulles même entre deux messages du même type. Les distances correspondantes de l'exemple sont données dans le tableau 5.2. Par rapport aux différences de longueur entre les messages (le tableau 5.1), moins d'incohérences de distance apparaissent mais d'autres problèmes persistent. En effet, bien qu'elle reflète la présence ou non de certains mots-clés, le contenu même du message ajoute beaucoup de bruit à celle-ci. Par exemple, le message m_3, en incluant le mot *interrogation* dans sa réponse, devient similaire à ce type de requête.

5.3.3 Positions des caractères

Une autre caractéristique des messages est la distribution de la position de leurs caractères. En effet, certains mots-clés se retrouvent souvent positionnés à la même place tels que ceux délimitant les premiers champs. Ainsi, cette représentation calcule la position moyenne de chaque caractère ASCII c pour un message m :

$$pos(m)_c = \frac{\sum_{i=1}^{i=k} p_m(c_i)}{k}$$

message	0	1	2	3	4
0	0.000	3.714	7.510	4.989	12.380
1	3.714	0.000	3.023	3.427	13.991
2	7.510	3.023	0.000	3.288	20.640
3	4.989	3.427	3.288	0.000	14.860
4	12.380	13.991	20.640	14.860	0.000

TABLE 5.3 – Distances entre les distributions des positions des caractères (lissage des distributions avec $\varepsilon = 10^{-6}$) — Pour chaque message en ligne, les cases rouges indiquent une incohérence de distance par rapport à deux autres messages

où i itère sur les différents caractères valant c et dont le nombre total est k. $p_m(c_i)$ retourne la position du i^e caractère de valeur c.

L'usage de la distance de Kullback-Leibler symétrique nécessite une distribution de probabilité, c'est-à-dire des valeurs entre zéro et un dont la somme est égale à un. En conséquence, les positions moyennes sont divisées par la somme de celles-ci, c'est-à-dire :

$$pos2(m)_c = \frac{pos(m)_c}{\sum_{i=0}^{255} pos(m)_i} \tag{5.5}$$

Une fois de plus, cette caractérisation des messages implique le stockage de 256 valeurs correspondant aux codes ASCII des caractères. Similairement à la distribution des caractères, elle est facilement compressible en ne gardant que les valeurs non nulles.

Cependant, le tableau 5.3 atteste du faible bénéfice de cette métrique comparée à la précédente puisque, malgré la disparition de certaines incohérences, d'autres sont apparues. En réalité, le problème sous-jacent reste le même car, malgré l'existence de mots-clés, le contenu associé est souvent variable entraînant un décalage des positions. Pour exemple, un protocole plus complexe aurait un message défini comme suit :

```
type champs1 valeur_champs1 champs2 valeur champs2...
```

Dans ce cas, la position de `champs2` contenant probablement un mot-clé est entièrement tributaire de la valeur du premier et notamment de sa longueur.

5.3.4 Positions pondérées des caractères

Face aux difficultés précédentes, cette sous-section dévoile une nouvelle représentation d'un message : la distribution des positions pondérées. L'hypothèse clé réside dans la corrélation entre le type d'un message et le contenu des premiers octets de celui-ci. En effet, la majorité des protocoles définissent un mot-clé représentatif du type du message dans les premiers octets pour en faciliter sa compréhension tout en simplifiant et en accélérant la tâche de l'analyseur syntaxique. En quelques mots, les premiers champs indiquent comment le message doit être interprété mais aussi la manière dont il doit être analysé et découpé.

Par conséquent, lors du calcul des positions des caractères, un poids plus fort sera attribué aux premières d'entre elles. En conséquence, cette nouvelle distribution se formalise comme ci-après :

$$pp^{\alpha}(m)_c = \frac{pp_2^{\alpha}(m)_c}{\sum_{i=0}^{255} pp_2^{\alpha}(m)_i} \tag{5.6}$$

où $pp_2^{\alpha}(m)_c = \frac{\sum_{i=1}^{i=k} p_m(c_i)^{-\alpha}}{k}$. Cette définition est équivalente à la précédente (formule (5.5)) à ceci près qu'un facteur α supérieur à un pondère chacune des positions. Dans le cas présent, $\alpha = 1$ correspond donc à prendre l'inverse. De ce fait, les premiers caractères auront un poids plus important que les suivants. Cette représentation des messages contient une fois de plus autant de valeurs que de caractères possibles et est compressible toujours en éliminant les valeurs nulles.

Cependant, le calcul de l'inverse des positions peut impliquer des valeurs petites voire très petites pour certains caractères (notamment si $\alpha >> 1$) et le lissage de cette distribution pour pouvoir appliquer la distance symétrique de Kullback-Leibler requiert de prendre des précautions pour fixer la valeur ε. Éviter de prendre des risques est faisable en remarquant qu'une probabilité valant 0 engendrera le terme suivant dans la formule 5.2 pour chaque autre valeur x_i :

$$0log\left(\frac{0}{x_i}\right) + x_i log\left(\frac{x_i}{0}\right)$$

Bien que cette somme soit clairement indéterminée, elle correspond à la somme de deux termes, un entre 0 et $-\infty$ et un autre qui tend vers ∞. Il paraît légitime de remplacer les valeurs nulles par un ε petit résultant en une somme d'un terme très petit et d'un terme très grand. Malheureusement, la courbe du logarithme ne croît pas de façon identique pour des valeurs proches de zéro et des valeurs très grandes causant une très grande ou très petite valeur dans la formule précédente. Définir ε est donc une tâche complexe pour deux raisons :

- ne pas trop déséquilibrer le calcul de la distance de Kullback-Leibler avec un ε trop petit ;
- ne pas avoir trop d'impact en ayant un epsilon trop grand et trop proche de valeurs réelles de la distribution.

Cependant, une autre solution est d'envisager que dans le cas précis d'une probabilité égale à zéro, la distance de Kullback-Leibler s'approxime aussi par zéro, c'est-à-dire que les deux termes de la somme précédente s'annulent. ε n'a donc plus besoin d'être paramétré et ce procédé diminue la complexité générale en éliminant l'étape du lissage des distributions.

La divergence de Kullback-Leibler se redéfinit alors comme :

$$KL(d^1, d^2) = \sum_{i,\ d_i^1 \neq 0,\ \ d_i^2 \neq 0} d_i^1 log\left(\frac{d_i^1}{d_i^2}\right) \tag{5.7}$$

Cette nouvelle formulation n'est applicable que pour la définition symétrique de la distance de Kullback-Leibler de la formule (5.2).

Le résultat de l'usage de cette nouvelle définition et de la distribution des positions pondérées des caractères est rapporté dans le tableau 5.4 où aucune incohérence ne persiste. Cette nouvelle distribution est donc un atout majeur dans la différenciation des différents types de messages. Toutefois, ce tableau n'est qu'un exemple particulier permettant d'obtenir ce cas parfait mais pas constamment atteignable. Ainsi, la bonne interprétation générale de cette métrique et des autres doit être soumise à des algorithmes de partitionnement des données pour vraiment pouvoir évaluer leurs intérêts. Toutefois, ce petit exemple démontre la supériorité de cette dernière métrique sur les autres qui sera encore confirmée grâce à l'évaluation complète présentée dans le chapitre 8.

5.3.5 Distribution ordonnée des caractères

Hormis la distance entre la longueur de deux messages, les métriques précédentes ne sont pas adaptées à des messages chiffrés avec un XOR différent. En effet, si chaque message est chiffré (XOR) avec

message	0	1	2	3	4
0	0.000	0.664	3.000	4.444	2.113
1	0.664	0.000	1.439	2.528	2.103
2	3.000	1.439	0.000	1.371	3.304
3	4.444	2.528	1.371	0.000	5.238
4	2.113	2.103	3.304	5.238	0.000

TABLE 5.4 – Distances entre les distributions des positions pondérées des caractères — Pour chaque message en ligne, les cases rouges indiquent une incohérence de distance par rapport à deux autres messages

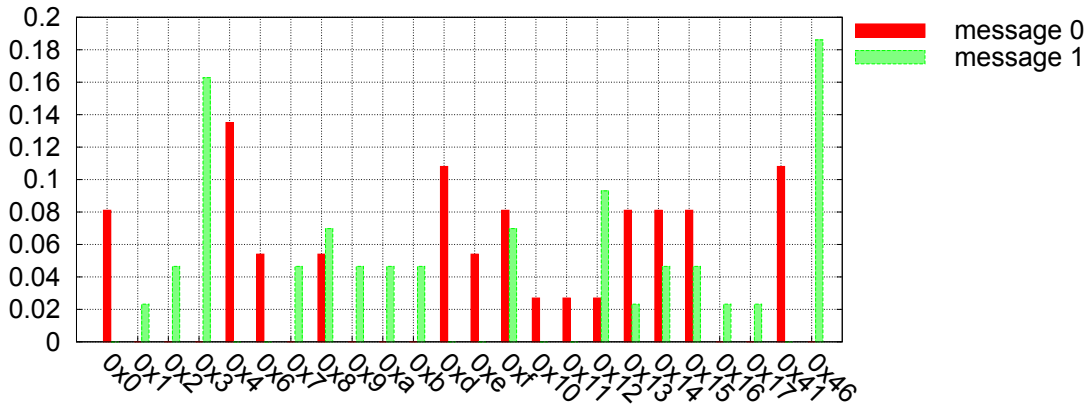


FIGURE 5.5 – Distribution des caractères entre les messages chiffrés $m_0 \oplus 'a'$ et $m_1 \oplus 'f'$ — la valeur hexadécimale des caractères est utilisée

la même clé, les distributions précédemment définies sont envisageables car cela ne correspond qu'à une transition des caractères. Néanmoins, si la clé change pour chaque message, un caractère est en fait représenté par plusieurs différents, ce qui empêche de comparer les distributions.

A titre d'exemple, la figure 5.5 montre la distribution des caractères des deux premiers messages de même type chiffrés avec deux clés différentes d'où une dissemblance évidente. D'ailleurs, la distance entre ces deux messages a été multipliée par plus de 10 en passant des versions non chiffrées à celles chiffrées.

[71] définit la distribution ordonnée inverse des caractères telle que :

$$\forall i,\ j,\ i < j,\ ord(m)_i > ord(m)_j \wedge ord(m)_i \in car(m) \wedge ord(m)_j \in car(m)$$

Plus simplement, cette nouvelle distribution correspond à $car(m)$ (distribution des caractères) ordonnée de la plus grande à la plus petite valeur. Ainsi, le stockage d'un message sous une forme ou l'autre est équivalent en terme d'espace. Elle est résistante face à un chiffrement par XOR impliquant un chiffrement unique pour chaque caractère. Ainsi, la fréquence d'un tel caractère est équivalente à sa version non chiffrée et donc à sa position dans l'ordonnancement :

$$ord(m \oplus k) = ord(m) \forall\ key\ k$$

La figure 5.6 découle de cet ordonnancement sur trois messages. Les messages de même type (m_0 et m_1) sont bien isolés du message m_3 de même que les distances de Kullback-Leibler associées.

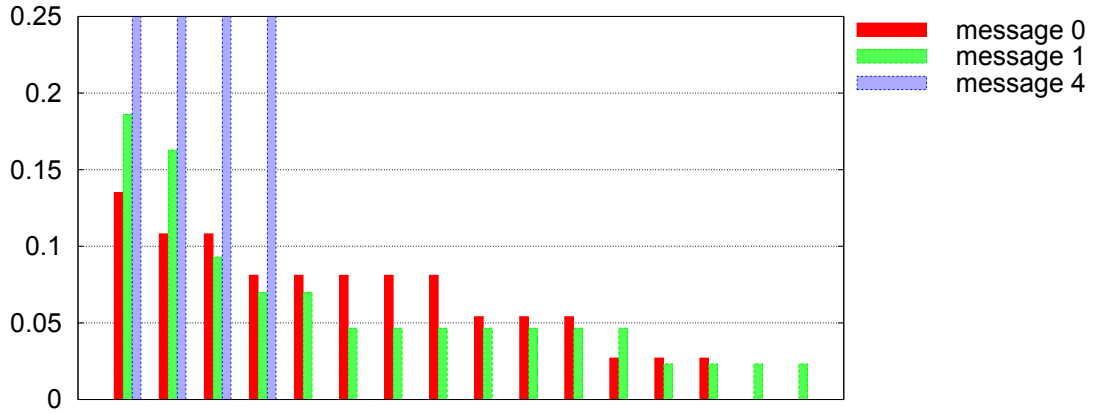


FIGURE 5.6 – Distribution ordonnée des caractères entre les messages chiffrés $m_0 \oplus 'a'$, $m_1 \oplus 'f'$ et $m_4 \oplus 'k'$

5.3.6 Alignement de séquences

Le chapitre 2 a mis en exergue les nombreux travaux de rétro-ingénierie tirant partie de l'alignement de séquences local grâce à l'algorithme de Smith-Waterman [28]. Bien que cet algorithme ne soit pas redétaillé ici, les métriques précédentes sont complémentaires à ces travaux pour deux raisons principales :

– ce sont des alternatives pouvant se révéler efficace ;
– la complexité de leur calcul est plus faible.

Le second point mérite d'être détaillé. En effet l'algorithme de Smith-Waterman est connu pour avoir une complexité en $O(nm)$ où n et m sont les tailles respectives de deux messages. Pour diminuer cette complexité, la simplification via l'emploi d'heuristique est une direction possible [25]. Une autre solution consiste simplement à envisager d'autres moyens de comparer deux messages tels que ceux détaillés auparavant. Il est clair que la différence de longueur entre deux messages est très rapide puisqu'elle ne nécessite qu'une opération. Cependant, les meilleures représentations sont celles basées sur des distributions. Dans ce cas, la distance de Kullback-Leibler est appliquée. En examinant les formules (5.1) et (5.7), la formule (5.2) peut se simplifier aisément en une seule somme :

$$KL^2(d^1,d^2) = \frac{1}{2}\sum_i \left[d_i^1 log\left(\frac{d_i^1}{d_i^2}\right) + d_i^2 log\left(\frac{d_i^2}{d_i^1}\right)\right] \tag{5.8}$$

Plus exactement, l'indice de la somme varie entre 0 et 255 au maximum dans le cas des distributions. La complexité de ce calcul ne dépend donc pas de la taille des messages souvent bien plus grande que le nombre de caractères employés. Ainsi, cette complexité est en $O(1)$ par rapport à la taille des messages.

Cependant, une étape de caractérisation des messages est obligatoire pour former les distributions. Sans optimisation, elle nécessite trois itérations :

– une première sur l'ensemble du message pour récupérer la caractéristique de chaque caractère (valeur, position) ;
– une seconde permettant de calculer les moyennes sur les différents caractères ;
– une troisième optionelle qui permet, selon la représentation utilisée, d'obtenir réellement une distribution, c'est-à-dire dont la somme des valeurs est égale à un.

Alors que les deux premiers points vont nécessiter d'examiner le message intégralement, la dernière est une opération globale réalisable en une passe sur celle-ci (256 valeurs au maximum) car la somme de toutes les valeurs peut être calculée à la seconde étape. Ainsi, la complexité de l'analyse de chaque message se limite à $O(2n) = O(n)$ où n est la taille du message. De plus, cet examen n'est réalisé qu'une unique fois par message. Seul le calcul de la distance en complexité constante est pratiqué pour chaque couple de messages contrairement à l'algorithme d'alignement nécessitant d'être entièrement réexécuté pour chacun d'eux.

5.4 Partitionnement des données

5.4.1 Définition du problème

Le partitionnement de données ou clustering vise à construire des clusters à partir d'un ensemble de points $X = \{x_i, i = 1 \dots n\}$. De plus, le nombre de clusters ou classes n'est pas connu à priori. Étant donné que chaque point appartient à une classe donnée $C = \{c_i, i = 1 \dots m\}$, $T_i = \{x_{i1}, \dots, x_{ip}\}$ est l'ensemble des points de X appartenant à la classe i. Il est important de noter que chaque x est indexé par un unique indice (ik) et non deux, avec $ik \in 1 \dots n$. Par conséquent il faut préciser que $\forall\ T_i$, $\forall\ x \in T_i\ x \in X$, c'est-à-dire $\forall x_{kp}, 1 < kp < n$. Ces notations seront utilisées dans l'ensemble de cette section.

Un algorithme de clustering s'assimile à une fonction $\psi : X \to K$ où $K = k_i, i = 1, \dots, t$ c'est-à-dire une fonction qui associe une classe k_i à chaque point x_i. L'ensemble K est différent de l'ensemble C et leurs cardinalités peuvent diverger selon l'efficacité de l'algorithme utilisé. Obtenir des ensembles C et K similaires est l'objectif à atteindre pour une technique de clustering performante.

Une bonne similitude est synonyme d'un nombre de classes proches voire identiques avec des clusters consistants, c'est-à-dire ne contenant qu'un seul type de messages en majorité. Un algorithme de partitionnement a donc deux objectifs dont le premier se définit simplement :

$$min(|card(C) - card(K)| \tag{5.9}$$

La notation $card(A)$ désignant le cardinal de A est ici employée pour ne pas la confondre avec la notation $|x|$ représentant la valeur absolue de x.

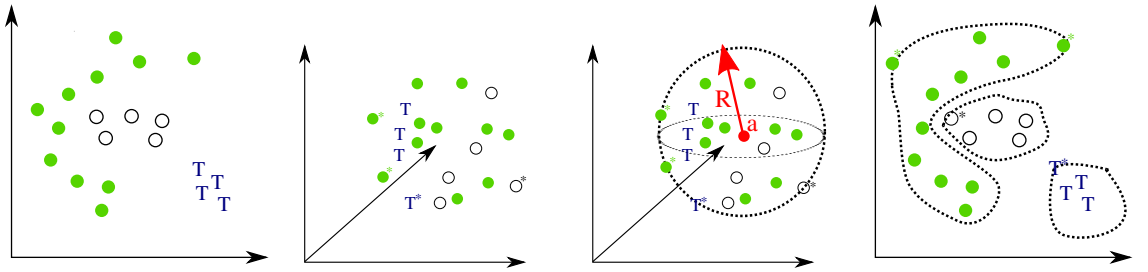
L'objectif de consistance se spécifie grâce à l'ensemble $R_i = \{x_{i1}, \dots, x_{ip}\}$ contenant les points affectés à la classe i par Ψ. Chaque classe trouvée se voit étiqueter par le type réel majoritaire :

$$\forall k \in K, type(k) = \{c_k \in C, \forall c_l \in C, card(\{x, \psi(x) = k, x \in T_l\}) \leq card\{x, \psi(x) = k, x \in T_k\}\}$$

En définissant chaque cluster comme un unique point de donnée, une grande précision est possible (pas de types mélangés) mais implique une faible consistance car plusieurs points de même types sont éparpillés dans différents clusters. De ce fait, l'objectif est d'obtenir les plus gros clusters pour chaque type réel avec le moins de messages de type invalide au sein de chacun d'eux :

$$max\left(\sum_{c \in C} \max_{k \in K, type(k) = c} (card(\{x, \psi(x) = k, x \in T_c\}))\right) \tag{5.10}$$

Bien évidemment, remplir ces deux objectifs simultanément n'est pas toujours réalisable mais ils définissent exactement et formellement le but visé par les algorithmes de clustering. Dans le cadre de la rétro-ingénierie, les points de données coïncident avec les messages.



(a) Points initiaux (b) Projection dans l'espace de re-description (c) Recherche de l'hypersphère englobante (d) Projection de l'hypersphère dans l'espace d'origine

FIGURE 5.7 – Les étapes de SVC (Les étoiles symbolisent les vecteurs supports)

5.4.2 Méthodes utilisées

Plusieurs méthodes sont applicables et deux d'entre elles ont été sélectionnées : le clustering hiérarchique ascendant[27] et SVC (Support Vector Clustering). Ces méthodes sont brièvement décrites ici et détaillées dans l'annexe A.

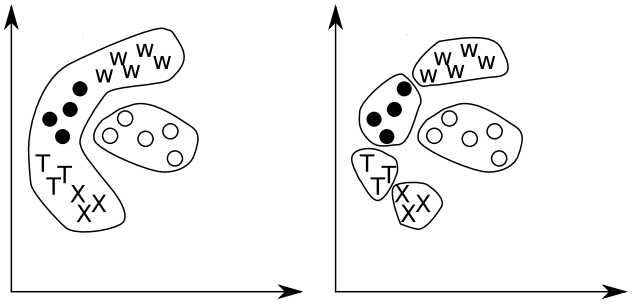
Le clustering hiérarchique ascendant [27] est une méthode de partitionnement très connue. Son principe réside dans la notion de distance entre deux points telles que celles décrites dans la section précédente. En effet, la première étape construit un cluster pour chaque message individuel. Ensuite, les plus proches sont regroupés entre eux à chaque étape jusqu'à regrouper l'ensemble des points dans un même cluster.

Bien que les fondements des machines à vecteurs de support ou séparateurs à vaste marge (SVM ou Support Vector Machines en anglais) datent des années 60-70, ce n'est qu'au cours des années 90 qu'elles furent véritablement explorées d'abord dans le domaine de la classification pure avant d'être étendues à de nombreux domaines. Cette popularité découle de leurs performances surpassant les autres algorithmes de classification dans de nombreux cas [7]. Plusieurs méthodes mettant en œuvre les SVM existent et le choix dépend de l'application visée. Généralement basée sur de l'apprentissage et donc inadapté au problème énoncé au début cette section, une autre méthode demeurant peu exploitée mais donnant de bons résultats sans apprentissage a été proposée par Ben-Hur *et al.* initialement dans [19] avant d'être affinée dans [18] sous le nom de *Support Vector Clustering* (SVC) dont les grandes étapes sont synthétisées dans la figure 5.7.

L'intérêt des SVM est la séparation de points de données non séparables dans l'espace d'origine (figure 5.7(a)). Ceux-ci sont donc initialement au sein de clusters de formes arbitraires difficilement séparables. L'astuce des SVM projette ces points dans un espace de plus grande dimension (figure 5.7(b)), appelé espace de re-description, dans lequel les points seront facilement dissociables. Intuitivement, un espace plus grand caractérisera les données plus précisément pour mieux les séparer. Dans le cadre de SVC, les contours des clusters de l'espace d'origine se réduisent à une hypersphère dans l'espace de plus grande dimension. Son objectif est donc de trouver cette sphère (figure 5.7(c)) puis de la repasser dans l'espace d'origine pour découvrir les clusters qu'elle forme (figure 5.7(d)).

Les détails de la méthode sont mentionnés dans l'annexe A et aboutissent sur le problème d'optimisation suivant :

$$W = \sum_i \Phi(x_i)^2 \beta_i - \sum_{i,j} \beta_i \beta_j K(x_i, x_j) \tag{5.11}$$



(a) SVC (b) Partitionnement hiérarchique

FIGURE 5.8 – Méthode globale

avec comme contraintes :

$$0 \leq \beta_i \leq C \; ; \; \sum_i \beta_i = 1 \tag{5.12}$$

K est une fonction noyau permettant de s'affranchir de l'utilisation d'une fonction de projection (figure 5.7(b)). Le noyau Gaussien fut choisi et se définit par :

$$K(x_i, x_j) = e^{-q\|x_i - x_j\|^2} \tag{5.13}$$

5.4.3 Méthode globale

Au cours des premières expérimentations de l'application de SVC pour la découverte des types de messages, de nombreux clusters ne furent pas découverts. Plus précisément, SVC trouvait effectivement des clusters de données assez diverses correspondant géométriquement à des formes arbitraires. D'un autre côté, le partitionnement hiérarchique se révélait encore plus mauvais en terme d'efficacité. L'idée maîtresse introduite dans [1] est d'appliquer le partitionnement hiérarchique sur chaque cluster découvert par SVC. En effet, les expérimentations réalisées ont montré que les formes irrégulières des clusters produits renferment plusieurs petits clusters différents en réalité. Bien évidemment, ce cas dépend des données à classer mais les bénéfices d'une telle approche pour l'identification des messages sont sans équivoque dans le chapitre 8. La figure 5.8 illustre de tels clusters nécessitant l'application d'une classification à double niveau :

- application de SVC,
- partitionnement hiérarchique pour redécouper chacun des clusters découverts par SVC en utilisant la méthode du barycentre puisque SVC s'est déjà chargé de trouver les formes arbitraires.

5.5 Architecture générale

Il est maintenant possible de décrire précisément comment les techniques et les représentations des messages expliquées précédemment vont être mises en œuvre.

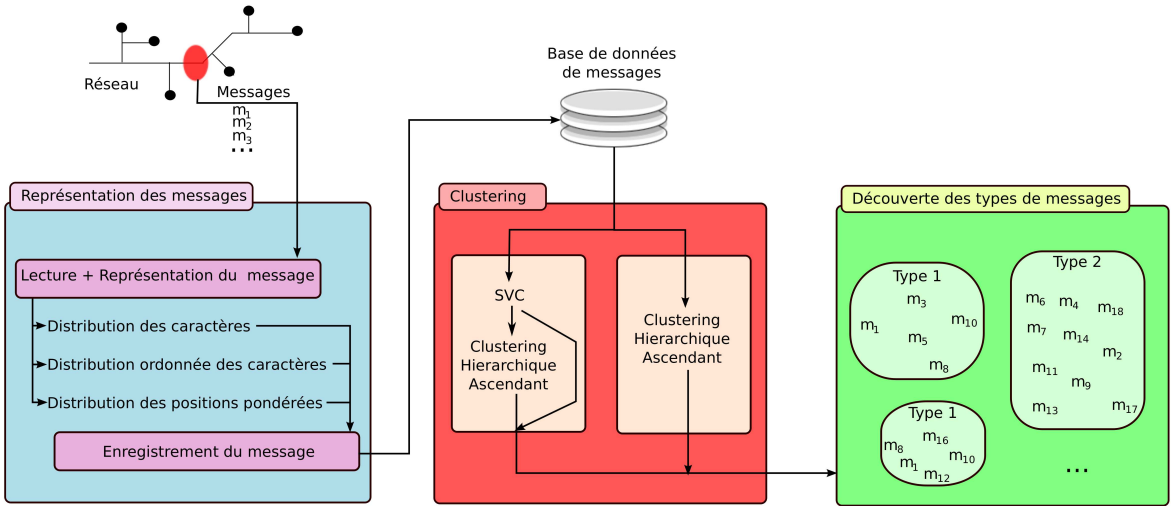


FIGURE 5.9 – Architecture générale

5.5.1 Fonctionnement

La figure 5.9 expose les différentes étapes de la nouvelle approche de classification des types de messages. La première étape consiste à récupérer un ensemble de messages. La méthode étant totalement non supervisée, aucun message n'est étiqueté et ils doivent être directement partitionnés en différents clusters.

Dans un premier temps, ils sont stockés sous une des représentations simplifiées introduite précédemment. Plus précisément, trois d'entre elles seront considérées :

- la distribution des caractères reflétant le contenu du message et servant de points de comparaison avec les autres ;
- la distribution des positions pondérées des caractères se révélant plus profitable par rapport aux autres ;
- la distribution ordonnée des caractères présentant une résistance au chiffrement par un XOR et mesurant ainsi l'efficience de la méthode suggérée avec un protocole faisant usage d'un tel chiffrement.

Il n'est donc plus obligatoire de stocker le message intégralement. La base de données ainsi construite alimente les différentes méthodes de clustering telles que le partitionnement hiérarchique (en utilisant la distance de Kullback-Leibler symétrique), SVC ou la méthode globale combinant les deux. Les différents clusters regroupent alors les messages de même type. Du fait du caractère non supervisé, aucun type réel ne peut être assigné à un cluster sans pour autant invalider la méthode puisqu'elle vise à aider la construction de la machine à états pour laquelle le nom exact d'un type n'est d'aucune utilité.

Paramétrage automatique de SVC

La méthode de reconnaissance des types est non supervisée et ne peut donc être préalablement entraînée alors que beaucoup de paramètres existent et doivent être fixés. Concernant le partitionnement hiérarchique, la définition du paramètre principal τ indiquant la distance maximale autorisée pour rassembler deux clusters fut déjà l'objet de travaux par le passé [11]. SVC nécessite beaucoup plus de paramètres mais les expérimentations réalisées sur plusieurs protocoles dans le chapitre 8 mettent en évidence que

peu d'entre eux ont un impact important et ont besoin d'être redéfinis pour chaque protocole utilisé. En outre, ils sont souvent dépendants de contraintes limitant leurs valeurs possibles telles que la constante C :

$$\beta_init_i < C < 1 \tag{5.14}$$

où β_init_i sont les valeurs initiales de β_i dans la forme duale (5.11) ajustées lors de la résolution du problème d'optimisation. En réalité, la valeur de ces variables n'a pas d'impact particulier sur le résultat de l'optimisation mais joue un rôle dans le nombre d'itérations nécessaires pour la résolution. Effectivement, si ces valeurs sont proches de leurs valeurs finales alors la résolution sera très rapide. SVC impose néanmoins la contrainte suivante :

$$\sum_i \beta_init_i = 1$$

Sans connaissance à priori, une affectation uniforme est préférable :

$$\beta_init_i = \frac{1}{N}$$

où N est le nombre de points ou messages à classer. De plus, les expérimentations montrent le faible impact de la valeur de C tant qu'elle correspond à une valeur proche de β_init_i :

$$\beta_init_i \lessapprox C$$

Cependant, l'utilisation du noyau gaussien demande de fixer justement le paramètre q de la formule (5.13). Une technique, proposée dans cette thèse, a donc été mise au point pour le déterminer automatiquement. Pour rappel, le rôle de SVC (premier niveau de classification) est d'obtenir des clusters assez grands même si ils contiennent des messages de différents types. L'apprentissage automatique du paramètre q repose alors sur deux métriques décrivant des caractéristiques générales sur les groupes de messages définis sans pour autant prendre en compte le type réel de ces derniers (méthode non supervisée).

Tout d'abord la largeur d'un cluster $k \in K$ se définit par la distance maximale entre deux points :

$$largeur(k) = \max_{x_i \in k, y_i \in k} dist(x_i, y_i)$$

La première caractéristique d'une classification K_q (ensemble des clusters construits) dépendant du paramètre q est donc sa largeur égale à celle du cluster le plus large :

$$largeur(q) = \max_{k \in K_q} largeur(k)$$

La seconde caractéristique générale d'un partitionnement est le nombre de clusters découverts : $card(K_q)$

Le paramètre q a un rôle majeur définissant le degré de séparation des données dans l'espace de re-description. Ainsi, une faible valeur de q n'entraînera que des différences limitées entre les points comme le montre l'équation (5.13) engendrant ainsi un fort regroupement des messages dans un seul ou peu de clusters. Plus q augmente, mieux les messages sont séparés et plus de clusters sont alors construits. En partant d'un cluster unique, lorsque SVC découpe les messages en grands groupes, la largeur maximale doit décroître assez significativement sinon cela signifie plus probablement que seuls quelques messages ont été séparés du cluster initial. Toutefois, la soustraction d'un unique message à un cluster peut grandement modifier sa largeur d'où la nécessité d'observer deux changements parallèles :

- une augmentation significative du nombre de clusters,
- une augmentation significative de la largeur.

L'appréciation du paramètre q demande de suivre l'évolution de ces valeurs en parallèle. Étant donné p valeurs de q ($q_1, q_2, \ldots, q_p$), l'ensemble des largeurs associées est noté $< largeur(q_1), \ldots, largeur(q_2) >$ et $< card(q_1 >, \ldots, card(q_2) >$ représente le nombre de clusters découverts. Ainsi, en se basant sur la différence de ces valeurs entre deux q_i et q_j consécutifs, l'évolution du nombre de clusters et de la largeur du plus grand cluster est observable :

$$evol_c(q_i) = \frac{card(q_i) - card(q_{i-1})}{\max_i(card(q_i))} \; si \; i > 1, \; 0 \; sinon \tag{5.15}$$

$$evol_w(q_i) = \frac{largeur(q_{i-1}) - largeur(q_i)}{\max_i(largeur(q_i))} \; si \; i > 1, \; 0 \; sinon \tag{5.16}$$

Les divisions ne servent qu'à ramener les valeurs calculées ente 0 et 1 pour une comparaison plus aisée. La condition quand à elle permet de fixer la première valeur à 0 lorsqu'il n'y a pas de référentiel précédent. De plus, comme le nombre de clusters augmente et la largeur diminue, l'opposé de cette dernière est prise en compte pour faciliter la comparaison parallèle de ces métriques sur un même graphique. Le chapitre 8 révélera l'intérêt pratique et efficace de ces métriques dans un cas réel.

5.6 Bilan

Ce chapitre a expliqué en détail une nouvelle méthode de découverte automatique des types de messages d'un protocole. Elle se singularise par deux aspects :

- la représentation des messages,
- les méthodes de partitionnement.

Le premier point a notamment mis en évidence qu'un message peut facilement être substitué par une distribution comprenant au plus 256 valeurs alors que plusieurs techniques actuelles bien connues analysent intégralement le message [16, 6] grâce à des techniques d'alignement de séquences impliquant une complexité très élevée par rapport à celles induites par les nouvelles métriques proposées ici. Par ailleurs, la distribution ordonnée des caractères permet de les représenter indépendamment d'un chiffrement par XOR.

[8] est aussi une méthode totalement automatique analysant uniquement les messages pour en extraire également la syntaxe. Malgré cet intérêt non négligeable, les auteurs s'appuient sur une hypothèse forte considérant que les champs textuels sont forcément séparés par des caractères séparateurs préétablis (espace, tabulation...). La méthode préalablement proposée dans ce chapitre ne formule en aucun cas une hypothèse de ce type de manière à rester le plus général possible pour pouvoir être appliquée sur des protocoles totalement inconnu à l'instar de certains malwares (vers, botnets...).

Le second point important de la méthode suggérée est l'emploi de SVC, une technique récente qui se base sur le paradigme des machines à vecteurs de support présentant de très bons résultats dans les différents domaines où elles furent employées mais également dans le cadre de la rétro-ingénierie comme le montre le chapitre 8. Enfin, il a été nécessaire de coupler cette méthode avec celle du partitionnement hiérarchique ascendant beaucoup plus connue et ancienne.

Enfin, au vue du nombre important de paramètres, un procédé d'apprentissage automatique de ces derniers complète ce nouveau système de découverte des types de messages.

6

Nouvelles techniques de fingerprinting

Sommaire

6.1 Introduction . . . 125
6.2 Définitions des problèmes . . . 126
6.2.1 Terminologie . . . 126
6.2.2 Fingerprinting supervisé . . . 126
6.2.3 Fingerprinting non supervisé . . . 127
6.3 Machines à vecteur de support multi-classes . . . 127
6.4 Fingerprinting comportemental et temporel . . . 128
6.4.1 Modélisation formelle . . . 129
6.4.2 Générations des signatures . . . 130
6.4.3 Mise en œuvre . . . 132
6.5 Fingerprinting syntaxique . . . 134
6.5.1 Représentation . . . 134
6.5.2 Distances . . . 136
6.5.3 Technique supervisée . . . 140
6.5.4 Technique non supervisée . . . 141
6.6 Conclusion . . . 146

6.1 Introduction

De nombreuses techniques de fingerprinting furent introduites dans le chapitre 3 mais peu d'entre elles cherchent à déterminer le nom et la version d'un équipement (logiciel ou physique). La plupart de celles qui se focalisent sur ce problème caractérisent un équipement par la valeur de quelques champs ou l'ordre dans lequel ils apparaissent dans le message [4, 24, 23].

Ainsi, peu d'information est utilisée alors que des travaux récents soulignent clairement l'avantage d'un nouveau type d'information :

– le comportement d'un équipement [22],
– la structure syntaxique des messages émis par un équipement [8].

Ce chapitre présentera donc deux nouvelles méthodes de fingerprinting :

– le fingerprinting comportemental,

– le fingerprinting syntaxique.

La première se base sur [22] en étudiant les messages émis par une machine ainsi que la manière dont elle répond à ceux reçus tout en tenant compte du temps nécessaire pour émettre un message. Cette approche nécessite une nouvelle formalisation équivalente à un arbre qui fut l'objet d'une publication également [1].

La seconde utilise les arbres syntaxiques introduits dans [8]. Toutes les deux mettent en œuvre l'utilisation de machines à vecteurs de support nécessitant une phase d'apprentissage, impliquant donc un fingerprinting supervisé. Cependant, une autre contribution majeure est la classification automatique non supervisée des arbres syntaxiques grâce à l'adaptation d'un nouvel algorithme. Enfin, la publication [2] traite du fingerprinting syntaxique.

6.2 Définitions des problèmes

6.2.1 Terminologie

L'objectif du fingerprinting est de réussir à identifier un équipement à partir des messages qu'il envoie en les comparant avec une empreinte ou fingerprint. La définition exacte de celle-ci dans cette étude sera précisée selon les différents cas. De manière générale, l'objectif est d'associer la représentation d'un ou plusieurs messages d'une machine à un type d'équipement.

Toutes les méthodes proposées reposent sur une définition arborescente de l'empreinte d'où un ensemble de tests T composés de N arbres : $T = \{t_1, \ldots, t_N\}$. Le fingerprinting tentera d'associer à chacun d'eux un type d'équipement parmis K : $D = \{d_1, \ldots, d_K\}$. Le fingerprinting supervisé est entraîné grâce à jeu d'apprentissage défini par un ensemble de M arbres $L = \{l_1, \ldots, l_M\}$.

La fonction $real(t_i) : T \cup L \rightarrow D$ retourne le type correspondant à l'arbre t_i. Une fois le système déployé, cette fonction n'est, bien entendu, pas disponible. Cependant, sa définition sur l'ensemble T servira à identifier les objectifs des approches proposées.

6.2.2 Fingerprinting supervisé

Le fingerprinting supervisé comporte deux phases. La première recherche la fonction $\Omega_L : L \rightarrow D$ visant à identifier correctement le plus d'arbres du jeu de test :

$$\Omega_L = \arg\max_{\Omega} card(\{l_i \in L, \Omega(l_i) = real(l_i)\}) \tag{6.1}$$

L'objectif ultime est alors d'obtenir : $\forall l_i \in L,\ \Omega_L(l_i) = real(l_i)$.

Dans un second temps, cette même fonction prend en entrée les arbres $t_i \in T$ pour distinguer les machines du jeu de test. Un bon apprentissage doit alors aider à associer chacune d'elle au type d'équipement convenable, c'est-à-dire $\Omega_L(t_i) = real(t_i)$. Cette fonction sera simplement notée Ω par la suite.

6.2.3 Fingerprinting non supervisé

Étant non supervisé, ce genre de technique repose exclusivement sur le jeu de test pour inférer un classifieur :

$$\Psi_T : T \rightarrow P$$

Il est important de remarquer que cette fonction n'associe pas précisément un arbre à un type d'équipement mais à une valeur d'un autre ensemble P, correspondant aux différents types découverts par Ψ, pouvant être différent de D. En effet, sans ensemble d'apprentissage, la nomenclature inconnue des équipements suscite la création d'une nouvelle arbitraire similaire à la réalité. En quelques mots, la similitude se définit d'abord par un nombre de classes ou de types d'équipements trouvés proche du nombre réel :

$$min(|card(C) - card(K)|) \tag{6.2}$$

De plus, les classes trouvées doivent être consistantes, c'est-à-dire ne contenant en majorité qu'un seul type. Les formules ci-dessous sont construites de la même manière que dans la section 5.4 où chaque cluster trouvé se voit étiqueté par le type de l'équipement représentant la majorité des arbres qu'il contient :

$$\forall p \in P, type(k) = d_l \in D, \forall d_m \in D, card(\{t, \Psi(t) = p, real(t) = d_m\}) \leq card\{t, \Psi(t) = p, real(t) = d_l\}$$
$$max\left(\sum_{d \in D} \max_{p \in P, type(p) = d} (card(\{t, \Psi(t) = p, real(t) = d\}))\right) \tag{6.3}$$

avec $R_i = \{t_{i1}, \ldots, t_{ip}\}$ les arbres affectés au cluster i.

L'objectif ultime est de créer des groupes d'arbres du même type totalement consistants :

$$|\Psi[T]| = K$$

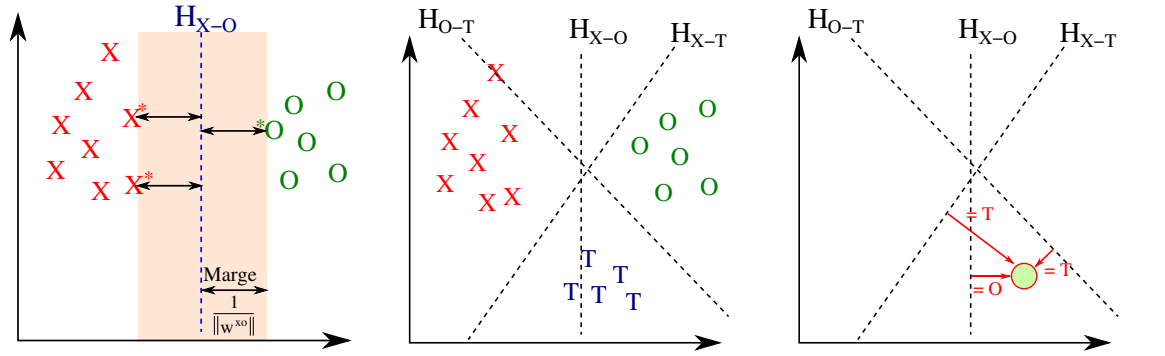
$$\forall < t_i, t_j >,\ real(t_i) = real(t_j) \Leftrightarrow \Psi_T(t_i) = \Psi(t_j)$$
$$\forall < t_i, t_j >,\ real(t_i) \neq real(t_j) \Leftrightarrow \Psi_T(t_i) \neq \Psi(t_j)$$

6.3 Machines à vecteur de support multi-classes

Dans le cadre d'une classification supervisée, les machines à vecteurs de support ou SVM seront une fois de plus employées vu leurs bonnes performances dans de nombreux domaines [7]. Contrairement au chapitre précédent, la technique mise en œuvre est supervisée et est qualifiée, plus exactement, de un-contre-un (one-versus-one) [10] car elle offre un bon compromis entre complexité et performances [16].

Cette méthode de segmentation multi-classes est dérivée de la méthode plus classique prenant en compte deux classes seulement qui recherche un séparateur à vaste marge, c'est-à-dire un hyperplan séparant au mieux les deux types de données. La méthode un-contre-un établit un tel séparateur pour chaque paire de classes. Les grandes étapes de cette technique sont les suivantes :

- apprentissage :
  - projection des points dans un espace de plus grande dimension,
  - découverte des hyperplans séparateurs.
- tests :



(a) Hyperplan séparateur entre 2 classes (les points étoilées sont les vecteurs supports)

(b) Hyperplan par couple de classe

(c) Classification d'un nouveau point par élection

FIGURE 6.1 – Classification multi-classes avec les SVM

– chercher de quel côté de chaque hyperplan le point à tester se trouve,
– déterminer quelle est la classe (type) la plus probable.

En effet, un des points-clés des SVM est la projection de points non linéairement séparables dans un espace de plus grande dimension, dit de re-description, où ils le seront. La figure 6.1(a) illustre l'hyperplan séparateur entre deux classes. SVM vise à ce qu'il soit le plus éloigné de chacun d'elle. La marge dénote donc cet « espace » vide de toute donnée et sa largeur est définie par rapport à l'hyperplan. Tout hyperplan se caractérise par un vecteur orthogonal $\vec{w}$ à celui-ci et un scalaire b représentant son décalage par rapport à l'origine de l'espace en suivant le vecteur $\vec{w}$. Considérant deux classes, l et k, l'hyperplan noté H_{l-k} est spécifié par le vecteur w^{lk} et le scalaire b^{lk}. Le cas un-contre-un définit l'ensemble des hyperplans comme sur la figure 6.1(b). Enfin, tout point à tester est projeté dans le même espace et comparé à chaque hyperplan pour identifier la classe la plus appropriée, puis la plus plébiscitée est sélectionnée. Ainsi dans l'exemple de la figure 6.1(c), la classe T est choisie car :

– H_{O-T} « vote » T,
– H_{X-T} « vote » T,
– H_{X-O} « vote » O.

La description formelle des SVM est consultable dans l'annexe B et il est important de noter qu'une fonction noyau K permet de s'affranchir de la projection des points originaux dans un espace de plus grande dimension.

6.4 Fingerprinting comportemental et temporel

Pour différencier les équipements sur un réseau, le fingerprinting comportemental suppose un protocole donné et un système capable de définir le type des messages correspondants. Cette tâche est réalisable en s'appuyant sur les spécifications du protocole ou grâce au système proposé dans le chapitre 5 lorsqu'elle ne sont pas disponibles.

Ce genre de fingerprinting différencie les types d'équipements selon les types de messages qu'ils envoient ainsi que leurs manières pour y répondre. L'observation des délais d'envoi des réponses est un indice supplémentaire pour affiner l'identification. De ce fait, il est obligatoire de définir une session c'est-à-dire une suite de messages envoyés et reçus englobant l'ensemble d'une communication entre plusieurs entités. Plus spécifiquement, une session peut se délimiter grâce à un champ spécifique de type

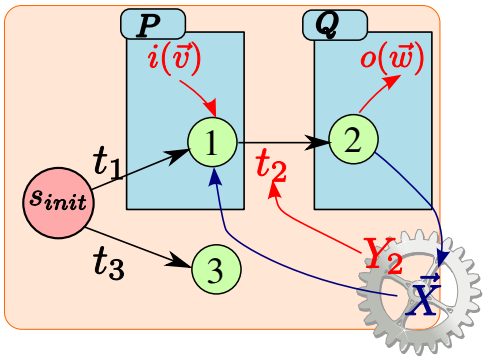


FIGURE 6.2 – Illustration d'un arbre TR-FSM

cookie lorsque la syntaxe est connue sinon elle se ramène à la définition d'un flux TCP ou UDP. Dans le cas d'UDP, elle est jugée terminée après une période relativement longue d'inactivité.

6.4.1 Modélisation formelle

La caractérisation comportementale repose sur l'élaboration d'une nouvelle formalisation appelée « Random Tree Parametrized Extended Finite State Machines » (TR-FSM) ou machine à états finis paramétrée à variables aléatoires temporelles. Ce nouveau modèle est dérivé de [22] en y ajoutant :

- des variables aléatoires temporelles représentant un gain d'information,
- une contrainte topologique imposant une structure arborescente à la machine à états.

La figure 6.2 illustre les différentes notations introduites ci-après. Un TR-FSM est un tuple $M =< S, s_{init}, I, O, \vec{X}, T, \vec{Y} >$:

- S un ensemble fini d'états $|S| = n$,
- s_{init} l'état initial,
- $I = \{i_0(\vec{v_0}), i_1(\vec{v_1}), \ldots, i_{p-1}(\vec{i_{p-1}})\}$ est l'alphabet pour les entrées dont le cardinal est p. Chaque symbole possède un vecteur de paramètres,
- $O = \{o_0(\vec{w_0}), o_1(\vec{w_1}), \ldots, o_{q-1}(\vec{w_{q-1}})\}$ est l'alphabet pour les sorties dont le cardinal est q. Chaque symbole possède un vecteur de paramètres,
- $\vec{X}$ est un vecteur de variables représentant la configuration de la machine,
- T est un ensemble fini de transitions, tel que :

$$\forall t \in T,\ t =< s_1, s_2, i(\vec{v}), o(\vec{w}), P(\vec{X}, i(\vec{v})), Q(\vec{X}, i(\vec{v}), o(\vec{w})) >$$

avec :

- s_1 et s_2 sont respectivement l'état initial et final,
- $i \in I$ le symbole qui a déclenché la transition,
- $o \in O$ le symbole produit par la transition,
- $P(\vec{X}, i(\vec{v}))$ est la condition globale à satisfaire pour réaliser la transition (une entrée et une configuration bien définie),
- $Q(\vec{X}, i(\vec{v}), o(\vec{w}))$ représente l'action découlant de l'exécution de la transition en se basant sur les différents paramètres.

- $\vec{Y}$ est un vecteur aléatoire de dimension $n-1$ où chaque valeur exprime le temps moyen pour effectuer une des transitions.

La topologie de la machine à états doit respecter une structure arborescente dont la racine est l'état initial. Par conséquent, il existe un enchaînement unique de transitions entre cet état et n'importe quel autre :

$$\begin{aligned} \forall s \in S \mid s \neq s_{init}, \exists\,!\ r \ states\ s_{i1}, s_{i2}, \ldots, s_{ir},\ s_{i1} = s_{init} \wedge s_{ir} = s \\ \forall j, 1 \leq j < r, t_{ij} \in T,\ t_{ij} =< s_{ij}, s_{i(j+1)}, i_{ij}(\vec{v_{ij}}), o(\vec{w}), P_{ij}(\vec{X}, i(\vec{v})), Q_{ij}(\vec{X}, i_{ij}(\vec{v}), o_{ij}(\vec{w})) > \end{aligned} \tag{6.4}$$

où la notation ij représente un index. Cette topologie entraîne l'existence d'une seule transition pour arriver dans un quelconque état d'où $|T| = n-1 : T = \{t_1, \ldots, t_{n-1}\}$. De plus, en tant qu'arbre, un TR-FSM a deux attributs importants : sa hauteur ou profondeur ainsi que sa cardinalité équivalente à $|S|$.

6.4.2 Générations des signatures

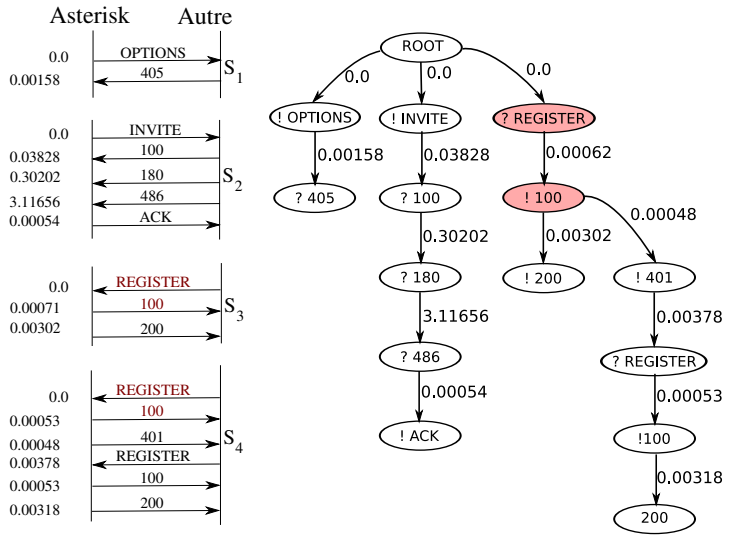
A titre d'exemple, le protocole SIP est employé ici [6] mais ne sera détaillé que dans le chapitre 8. Par la suite, les termes « état » et « transition » sont respectivement synonymes de « nœud » (sommet) et « arc » (arête). Un arbre comportemental s'édifie à partir d'un nombre fini de sessions d'un même équipement agrégés tel que ceux de la figure 6.3(b) La signature se représente donc comme un TR-FSM à partir de plusieurs sessions auxquelles il a participé. Chaque état est associé au type d'un message préfixé par **!** lorsqu'il est émis et préfixé par **?** lorsqu'il est reçu par l'équipement en question. Par exemple, les messages préfixés par **!** sur la figure 6.3(b) sont ceux envoyés par un équipement de type Asterisk alors que les autres sont ceux qu'il reçoit et peuvent donc provenir de différents équipements. Une transition se matérialise par un arc entre deux états et le vecteur $\vec{Y}$ de délai moyen correspond aux attributs des arcs. La signature de la figure 6.3(b) est dérivée des sessions capturées sur la figure 6.3(a). Chaque session représente une séquence de transitions dont les préfixes communs sont fusionnés. Par exemple, les session trois et quatre présentent deux premiers messages identiques d'où une partie de l'arbre en commun symbolisée en rouge sur la figure. En outre, un nœud racine générique est volontairement introduit pour que l'ensemble des sessions ait au moins un nœud en commun permettant de respecter la contrainte structurelle arborescente.

La construction des signatures est réalisée par l'algorithme 3 où, pour des raisons de simplicité, le délai d'une transition est directement contenue dans le nœud d'arrivée de celle-ci car chaque nœud n'est atteignable que par une unique transition. Brièvement, l'algorithme maintient un pointeur sur le nœud courant réinitialisé à la racine générique *ROOT* pour analyser chaque session. Pour chaque message d'une session (ligne 13), l'algorithme détermine si le nœud courant possède un fils du même type que le message (lignes 16-18). Dans ce cas, le délai de la transition correspondante est mise à jour avec le nouveau. Sinon, un nouveau nœud et une transition sont créés. Le nœud courant pointe alors sur ce nouveau nœud ou sur le nœud précédemment trouvé. Comme le délai d'une transition est une moyenne, construire une liste des délais d'une transition est préférable avant d'en calculer la moyenne après l'exécution de cet algorithme.

Étant donné n messages répartis en s sessions avec $n_i = |S_i|$, le nombre d'entre-eux appartenant à la session S_i, l'algorithme 3 boucle sur l'ensemble des messages de toutes les sessions (lignes 11 et 13) soit un total de n itérations. Dans le pire des cas, chaque message devra être comparé avec l'ensemble des fils du nœud courant qui sont, au plus, aussi nombreux que le nombre de sessions examinées précédemment, d'où un nombre d'itérations égal à $it = \sum_{i=1}^{s} i \times n_i$. Le maximum de cette valeur est $it = s(n-(s-1)) + \sum_{i=1}^{s-1} i = ns + 1.5s - 0.5s^2 < ns + 1.5s$ lorsque toutes les sessions ne contiennent qu'un message exceptée la dernière. Comme le nombre de session utilisée pour chaque signature est un paramètre constant, la complexité générale de cette algorithme est en $O(n)$.

Algorithme 3 Génération d'une signature

1: S une liste de sessions
2: S_i^j est j^e message de la i^e session
3: $l.len$ est le nombre d'élément dans la liste l
4: $m.type$ renvoie le type du message m préfixé par ! ou ?
5: $m.time$ retourne le délai du message m
6: $n.children$ est la liste des nœuds fils de n
7: $create_node(t)$ créer un nouveau nœud de type t
8: $n.update(d)$ met à jour le délai de la transition vers n grâce au délai d
9: $n.add_child(n2,d)$ ajoute le nœud n_2 comme fils de n avec une transition de délai d
10: n_{ROOT} est le nœud racine de la signature
11: **pour** $i \leftarrow 1 \dots length(S)$ **faire**
12: $current_node \leftarrow n_{ROOT}$
13: **pour** $j \leftarrow 1 \dots length(S_i)$ **faire**
14: $child = current_node.children$
15: $k \leftarrow 1$
16: **tantque** $k < child.length \wedge child_k.type \neq S_i^j.type$ **faire**
17: $k \leftarrow ind + 1$
18: **fin tantque**
19: **si** $k > child.length$ **alors**
20: $new \leftarrow create_node(S_i^j.type)$
21: $current_node.add_child(new, S_i^j.time)$
22: $current_node \leftarrow new$
23: **sinon**
24: $child_k.update(S_i^j.time)$
25: $current_node \leftarrow child_k$
26: **finsi**
27: **fin pour**
28: **fin pour**



(a) Sessions (valeur à gauche = délai (sec.))

(b) Signature

FIGURE 6.3 – Exemple de génération d'une signature pour un équipement de type Asterisk (SIP)

6.4.3 Mise en œuvre

Fonction noyau

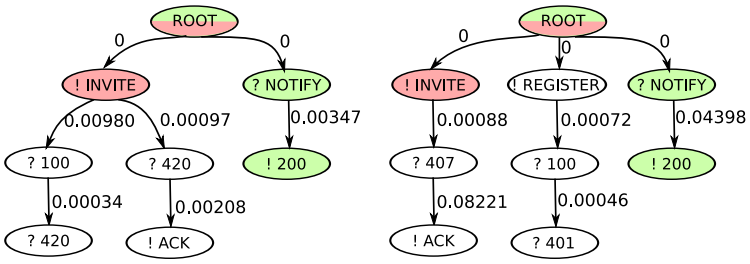
La classification SVM est appliquée dans le cadre du fingerprinting comportemental exigeant alors la formulation d'une fonction noyau K spécifique car la plupart d'entre elles sont limitées aux valeurs numériques et non à des arbres. De nouvelles fonctions spécifiques aux arbres furent mises au point récemment en recherchant les structures similaires [14, 15, 3] . Dans cette étude, ce concept est étendu aux TR-FSM en se basant sur des travaux précédents [8] évaluant la similarité entre deux arbres de manière équivalente à [8]. la similarité doit être égale à un pour deux TR-FSM identiques et zéro lorsqu'ils sont totalement différents.

Pour un arbre t_i, l'ensemble des chemins de la racine à chaque nœud est désigné par $chemins^i = chemin^i_1, \ldots, chemin^i_m$ où chacun des éléments est un chemin unique vers un nœud spécifique. La fonction $noeuds(chemins^i)$ renvoie la topologie de l'arbre sans les attributs des arcs, c'est-à-dire les délais des transitions. Par analogie, $noeuds(chemin^i_j)$ en fait de même pour un chemin donné. L'intersection entre deux arbres se définit alors comme :

$$I_{ij} = noeuds(chemins^i) \cap noeuds(chemin^j) \tag{6.5}$$

La figure 6.4 illustre l'intersection de deux signatures symbolisée par les couleurs (une couleur par chemin commun). Pour tous les chemins partagés, la similarité s'obtient en se basant sur les différences de délais des transitions par l'intermédiaire de la fonction *poids* :

$$inter_sim = \sum_{\substack{p \in I_{ij} \\ noeuds(chemin^i_k)=p \\ noeuds(cehmin^j_l)=p}} poids(chemins^i_k, chemins^j_l) \tag{6.6}$$



(a) Twinkle 1.10 (logiciel)

(b) Cisco 7940 firmware 8.93 (téléphone)

FIGURE 6.4 – Signature de deux équipements différents. Les attributs des arcs sont les délais moyens. Les chemins partagés sont colorés

Sans considération des transitions, $chemin_l^j$ et $chemin_k^i$ sont identiques. La fonction *poids* compare les délais de chaque transition commune s'appuyant sur un noyau Laplacien :

$$poids(p_1, p_2) = \sum_{n \in p_1} e^{-\alpha |f_{delai}(n,p_1) - f_{delai}(n,p_2)|} \tag{6.7}$$

En réalité, seuls les délais induits par l'équipement considéré sont à prendre en compte c'est-à-dire ceux des messages envoyés. Ainsi $f_{delai}(n,p)$ retourne le délai moyen associé au nœud n dans le chemin p si le message représenté par ce nœud est émis et vaut zéro sinon.

Théorème *La fonction suivante est une fonction noyau valide respectant les conditions du théorème de Mercer (chapitre 3 de [20]) :*

$$K(t_i, t_j) = \sum_{\substack{p \in I_{ij} \\ noeuds(chemin_k^i) = p \\ noeuds(chemin_l^j) = p}} \sum_{n \in p} e^{-\alpha |f_{delai}(n,p_1) - f_{delai}(n,p_2)|} \tag{6.8}$$

Preuve : L'équation (6.7) est un noyau valide construit à partir de celui de Laplace K_l. En effet $f_{delai}(n,p)$ est une fonction à valeur réelle déductible à partir de t_i puisque les chemins p sont des sous-arbres de t_i et t_j qui contiennent également les nœuds n. Les termes de la somme définie pour K s'écrivent alors sous la forme $K_l(f(t_i), f(t_j)$ et sont donc un noyau valide par construction. De plus, la somme de noyaux est également un noyau prouvant ainsi que K remplit bien toutes les conditions nécessaires. Le livre [20] donne de plus amples détails sur les façons de combiner des fonctions des noyaux ainsi que les preuves associées.

Classification

Sur la figure 6.4 le logiciel de téléphonie envoie les messages dix fois plus rapidement que le téléphone grâce à des capacités matérielles plus élevées (processeur, mémoire...). Cette observation confirme l'hypothèse que l'ajout d'information temporelle grâce aux TR-FSM procure un avantage certain sur le fingerprinting d'un équipement. Bien entendu, les délais sont largement contraints par l'endroit où les messages sont capturés et il est nécessaire, en pratique, de modérer ces délais selon le temps aller-retour

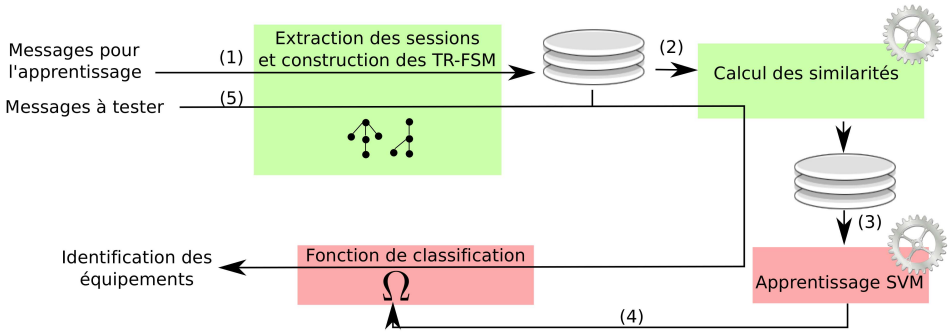


FIGURE 6.5 – Fingerprinting comportemental supervisé

induit par le réseau. Cependant, ce n'est pas nécessaire si le système est entraîné dans le même environnement que celui où il sera utilisé car les latences sont alors directement prises en compte dans la définition des signatures.

De façon générale, le fingerprinting comportemental proposé suit les étapes de la figure 6.5 :

1. les messages du jeu d'apprentissage sont regroupés en session qui sont ensuite agrégées dans des TR-FSM,
2. la similarité entre chaque TR-FSM est calculée grâce à la fonction de noyau K,
3. SVM prend en entrée ces différentes similarités,
4. SVM en déduit la fonction de classification,
5. cette fonction classe alors un par un chaque nouvel TR-FSM construit.

Alors que la signature d'un équipement est un ensemble de TR-FSM généré à partir de sessions dans lesquelles il a participé, l'empreinte ou le fingerprint d'un type d'équipement est l'ensemble des TR-FSM des équipements de ce type construits pendant la phase d'apprentissage.

6.5 Fingerprinting syntaxique

6.5.1 Représentation

Grammaire ABNF

La syntaxe des messages d'un protocole est généralement décrite grâce à la forme de Backus-Naur augmentée (ABNF) [2]. La figure 6.6 présente les éléments principaux d'une telle représentation. Les briques de bases sont les éléments terminaux qui représentent des valeurs réelles, c'est-à-dire des caractères. Deux types existent :

– les terminaux fixes équivalent à une chaîne constante de caractères,
– les plages de caractères spécifiées par les bornes inférieure et supérieure en hexadécimal.

Une règle détermine la façon dont un non terminal est dérivé. Un non-terminal est donc un élément construit à partir d'autres hormis trois cas particuliers :

– une alternative est un choix entre plusieurs éléments,
– une séquence est une suite ordonnée d'un nombre fixe d'éléments,

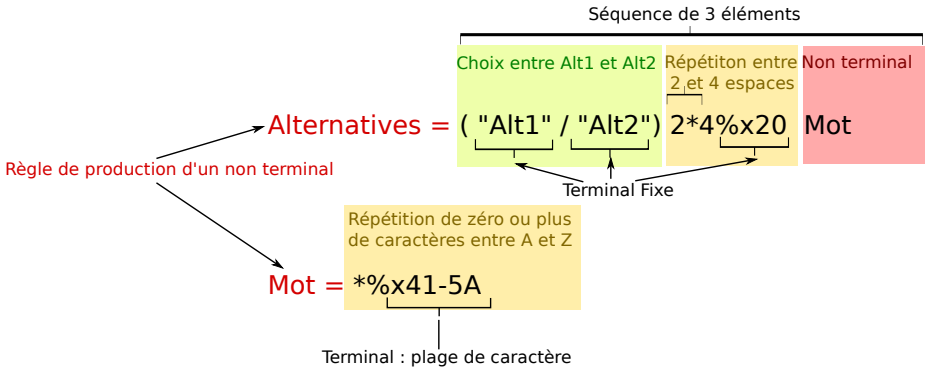


FIGURE 6.6 – Élement de définition d'une grammaire

– une répétition est une séquence d'un nombre variable, mais définie par une plage de valeur, du même élément.

Un non terminal doit forcément se situer dans la partie gauche d'une règle pour pouvoir le dériver tout en pouvant apparaître de la sorte à plusieurs reprises. Un message découle de la dérivation récursive de la première règle de la grammaire.

Il existe également plusieurs autres types d'éléments non détaillés ici car émanant directement de ceux précédemment exposés tels que des éléments optionnels, une répétition d'un nombre exact d'éléments, des définitions décimale ou binaire d'un terminal...

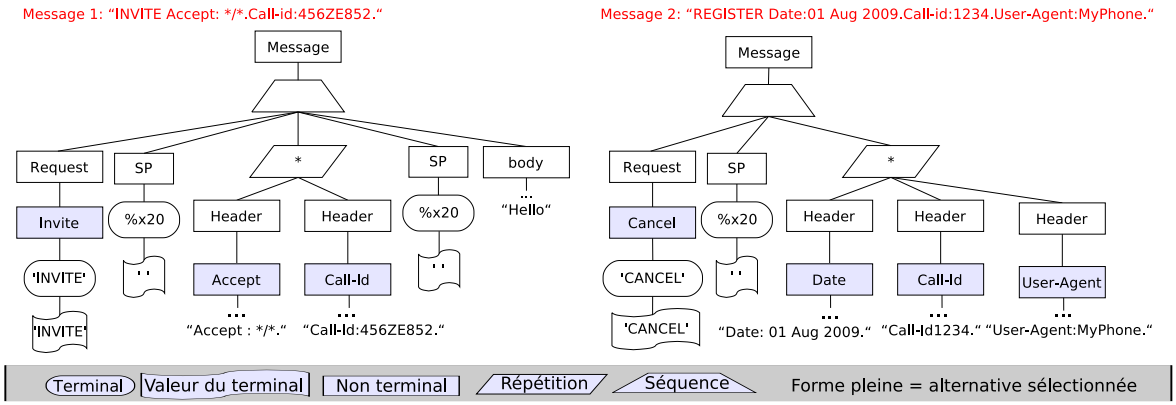
Les exemples de cette section seront illustrés grâce à la mini-grammaire de la figure 6.7(b) contenant très peu de règles basiques.

Arbre syntaxique

L'idée majeure de cette nouvelle approche de fingerprinting repose sur les différences significatives du contenu d'un message ainsi que sa structure hiérarchique inférée par la grammaire. En effet, le contenu est clairement une source d'information mais souvent facilement modifiable alors que l'organisation de cette information dans un arbre syntaxique est plus difficilement altérable tout en gardant un message valide sémantiquement proche. De plus, la forme d'un message est logiquement dépendante de la manière dont le logiciel utilisant le protocole construit le message car les spécifications d'un protocole n'imposent pas forcément tout (ordre des champs laissés libres, champs optionnels,...)

Un message se représente donc par un arbre syntaxique construit sur la base des règles de dérivation utilisées en partant de la règle initiale. Les nœuds sont alors créés de la manière suivante selon le type d'élément qui a été dérivé dans le message :

– un terminal est un nœud de l'arbre annoté par son nom ne contenant qu'un seul nœud représentant la valeur réelle correspondante dans le message,
– un non terminal est un nœud interne annoté par son nom ayant un unique nœud fils correspondant à celui créé par la partie droite de la règle utilisée,
– une séquence est un nœud interne avec un nombre fixe de nœuds fils correspondants à ceux créés par chacun des éléments composant la séquence,
– une répétition est un nœud interne avec un nombre variable mais défini de nœuds fils correspondants à ceux créés par l'élément répété,
– une alternative ne crée aucun nœud car l'élément sélectionné dans l'alternative constitue directement le nœud.



(a) Arbres syntaxiques

Message = Request SP *Header SP 0*1Body	Alpha = %x41-5A / %x61-7A ; A-Z / a-z
Request = Invite / Notify / Cancel	HCOLON = *SP ":" *SP
Invite = "INVITE"	SP = %x20 ; espace
Cancel = "CANCEL"	Accept = "Accept" HCOLON *Alpha "."
Notify = "NOTIFY"	Date = "Date" HCOLON *Alpha "."
Header = Accept / Date / Call-id / User-Agent	Call-Id = "Call-Id" HCOLON *Alpha "."
Body = *Alpha	User-Agent = "user-Agent" HCOLON *Alpha "."

(b) Mini-grammaire

FIGURE 6.7 – Construction des arbres syntaxiques à partir d'une grammaire

La figure 6.7(a) clarifie ce mécanisme de construction sur deux messages à partir de la mini-grammaire de la figure 6.7(b).

6.5.2 Distances

Sous-arbres et isomorphismes

Les méthodes de classification telles que celles proposées ici reposent généralement sur la notion de distance ou similarité entre deux points de données. Une fois les arbres syntaxiques des messages construits, mesurer la distance entre deux d'entre eux demeure essentielle. Les arbres ne sont pas des données numériques et peuvent être qualifiés de données catégoriques puisque chaque nœud a un type qui lui est attribué (catégorie) et les liens entre les nœuds également peuvent s'énoncer de la sorte. Par conséquent, les distances numériques classiques sont impraticables. De nouvelles sont apparues et se basent pour la plupart sur la distance d'édition [9] appréciant la distance entre un arbre t_1 et un arbre t_2 comme le nombre d'opérations (ajout, suppression, substitution de nœuds) pour transformer l'un en l'autre. Malheureusement, les auteurs de [6] notent à juste titre que ce calcul, considéré comme un problème NP-complet [24], impose d'ajouter des contraintes telles que l'ordre des fils d'un nœud pour pouvoir être réalisable en un temps polynomial. Ils proposent alors de nouvelles distances n'ajoutant aucune contrainte spécifique et présentant également une complexité polynomiale.

Elles reposent sur la notion d'isomorphisme entre deux arbres t_1 et t_2. Il est donc nécessaire de rappeler dans un premier temps quelques notions essentielles sur les arbres qui sont un sous ensemble des graphes :

- un graphe est le tuple $< V,E >$ où V correspond aux nœuds ou sommets et E l'ensemble des arêtes,
- deux nœuds u et v sont dits adjacents et notés $u \sim v$ si et seulement si ils sont reliés par une arête,
- un chemin entre deux nœuds u_0 et u_n est noté $u \bowtie v$ et correspond aux nœuds entre u_0 et u_n inclus, c'est-à-dire $\forall\ 0 < i \leq n,\ u_{i-1} \sim u_i$,
- $G =< V,E >$ est arbre si et seulement si $\forall u \in V, v \in V, u \neq v \Leftrightarrow !\exists u \bowtie v$.

Un homomorphisme de graphe est une fonction qui transforme un graphe en un autre, respectant la connectivité initiale des sommets :

Définition 22 : *Soit $G_1 =< V_1,E_1 >$, $G_2 =< V_2,E_2 >$ deux graphes, $f : G_1 \to G_2$ est un homomorphisme de graphe si $f = (f_V, f_E)$ où $f_V : V_1 \to V_2$ et $f_E : E_1 \to E_2$ tel que s'il existe une arête entre les sommets u et v de G_1, c'est-à-dire $u \sim v$, alors $f_V(u) \sim f_V(v)$ où $f_V(u)$ et $f_V(v)$ sont des sommets de G_2.*

Deux graphes sont isomorphes s'ils comportent le même nombre de nœuds et qu'ils sont connectés de la même manière :

Définition 23 : *f est un isomorphisme si f est un homomorphisme injectif et surjectif.*

Torsello *et al.* définissent ainsi la notion d'isomorphisme de sous-arbres [6] :

Définition 24 : *Soit deux arbres $t_1 = (V_1,E_1)$ et $t_2 = (V_2,E_2)$, la bijection $\phi : H_1 \to H_2$ avec $H_1 \in V_1$ et $H_2 \in V_2$ est un isomorphisme de sous-arbre si les sous-arbres de t_1 et t_2 engendrés respectivement uniquement par les nœuds H_1 et H_2 sont isomorphes. Ces sous-arbres se notent $t_1[H_1]$ et $t_2[H_2]$.*

L'objectif sera donc de chercher les sous-arbres isomorphes les plus grands pour en déduire le degré de similarité.

Plus particulièrement, les arbres utilisés ici sont étendus au cas attribué. Un arbre attribué est spécifié par le triplet $< V,E,\alpha >$ où α est une fonction associant à chaque nœud de l'arbre un ensemble de caractéristiques. Un isomorphisme de sous-arbres sur ce type reste défini comme précédemment, c'est-à-dire sans prendre en compte α. Cependant, la similarité sera d'autant plus grande que les nœuds des sous-arbres partagent des caractéristiques communes. Ainsi, si $\phi : H_1 \to H_2$ dénote un isomorphisme de sous-arbres entre $t_1 =< V_1,E_1,\alpha_1 >$ et $t_2 =< V_2,E_2,\alpha_2 >$, la similarité des sous-arbres engendré $t_1[H_1]$ et $t_2[H_2]$ est :

$$W(\phi) = \sum_{u \in H_1} \sigma(u, \phi(u))$$

où σ est une fonction de comparaison symétrique entre les caractéristiques de deux nœuds soit : $\sigma(u,\phi(u)) = \sigma(\alpha(u),\alpha(\phi(u)))$ (Dans l'article original, les auteurs laissent le choix de définir librement et indépendamment la fonction α pour chacun des arbres). ϕ_{12} dénote la bijection permettant d'obtenir la similarité maximale parmis tous les isomorphismes de sous-arbres possibles à partir des arbres t_1 et t_2 .

Distances

La cardinalité de l'arbre $t =< V,E,\alpha >$ notée $|t|$ est égale à son nombre de nœuds soit $|V|$. Trois des distances suggérées dans [6] seront mises en œuvre dans les expérimentations, ce choix ayant été établi à partir d'expériences préliminaires tout en gardant l'objectif de conserver pour chacune des techniques (supervisée et non supervisée) une distance normalisée et non normalisée :

$$d1(t_1,t_2) = |t_1| + |t_2| - 2W(\phi_{12})\ (\textit{non normalisée supervisée ou non}) \tag{6.9}$$

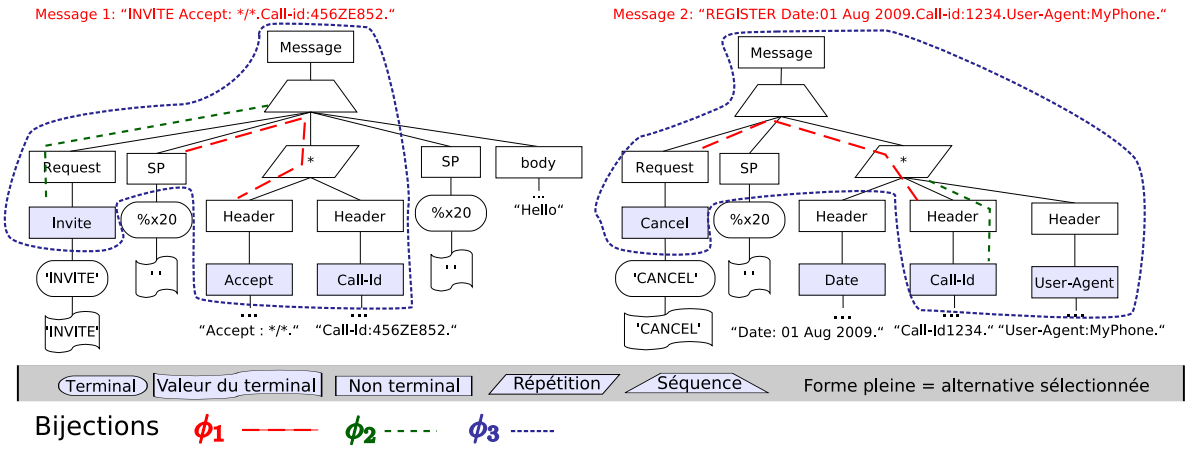


FIGURE 6.8 – Bijections de sous-arbres syntaxiques

$$d2(t_1,t_2) = 1 - \frac{W(\phi_{12})}{max(|t_1|,|t_2|)} \ (normalisée\ non\ supervisée) \tag{6.10}$$

$$d3(t_1,t_2) = 1 - \frac{W(\phi_{12})}{|t_1|+|t_2|-W(\phi_{MAX})} \ (normalisée\ supervisée) \tag{6.11}$$

Isomorphismes et arbres syntaxiques

Les arbres syntaxiques sont des arbres enracinés sur un sommet précis pour lesquels les distances précédentes sont également applicables. La seule contrainte supplémentaire est que le nœud racine d'un sous-arbre coïncide avec le nœud racine du sous-arbre correspondant par la bijection. Comme les valeurs des terminaux et les terminaux eux-mêmes sont souvent liés sémantiquement à un cas particulier (date, identifiant de session, nom d'utilisateur,...) et non à une caractéristique générale d'un type d'équipement, ils seront exclus de tous les sous-arbres. De plus, deux nœuds sont équivalents si ce sont des non terminaux de même nom ou si ce ne sont pas des non terminaux. Ainsi, les séquences et les répétitions sont considérées équivalentes. De cette manière $\sigma(u,v) = 1$ si les nœuds u et v sont équivalents. Cette définition est bien symétrique. La figure 6.8 reprend les exemples de messages précédents et met en exergue trois bijections telles que les sous-arbres générés soient bien isomorphes d'où :

- $W(\phi_1) = 1$ car seules les racines sont équivalentes (une répétition et une séquence),
- $W(\phi_2) = 1$ pour les mêmes raisons,
- $W(\phi_3) = 8$ en « inversant » les sous-arbres `Header` dans le deuxième arbre, témoignant ainsi de l'intérêt de l'isomorphisme qui repère une structure identique même si elle n'est pas positionnée au même endroit dans l'arbre.

Cependant, un grand nombre de structures similaires imposantes peuvent se trouver à plusieurs endroits de l'arbre sans néanmoins refléter une sémantique identique à l'instar des règles contenant `*Alpha` permettant de générer une chaîne de caractères alphabétiques aussi bien pour le corps du message que pour les différentes entêtes. La figure 6.9 illustre la définition de plusieurs entêtes contenant ce type de structure. Le message quatre est plus similaire au cinq qu'au trois alors que la logique aurait voulu le contraire car des messages contenant le même type d'entête (ici `User-Agent`) et donc sémantiquement proche devraient être plus similaires. En fait, la similarité est fortement biaisée par la longueur des chaînes de caractères alors que la prise en compte du type d'entête ne compte que pour un.

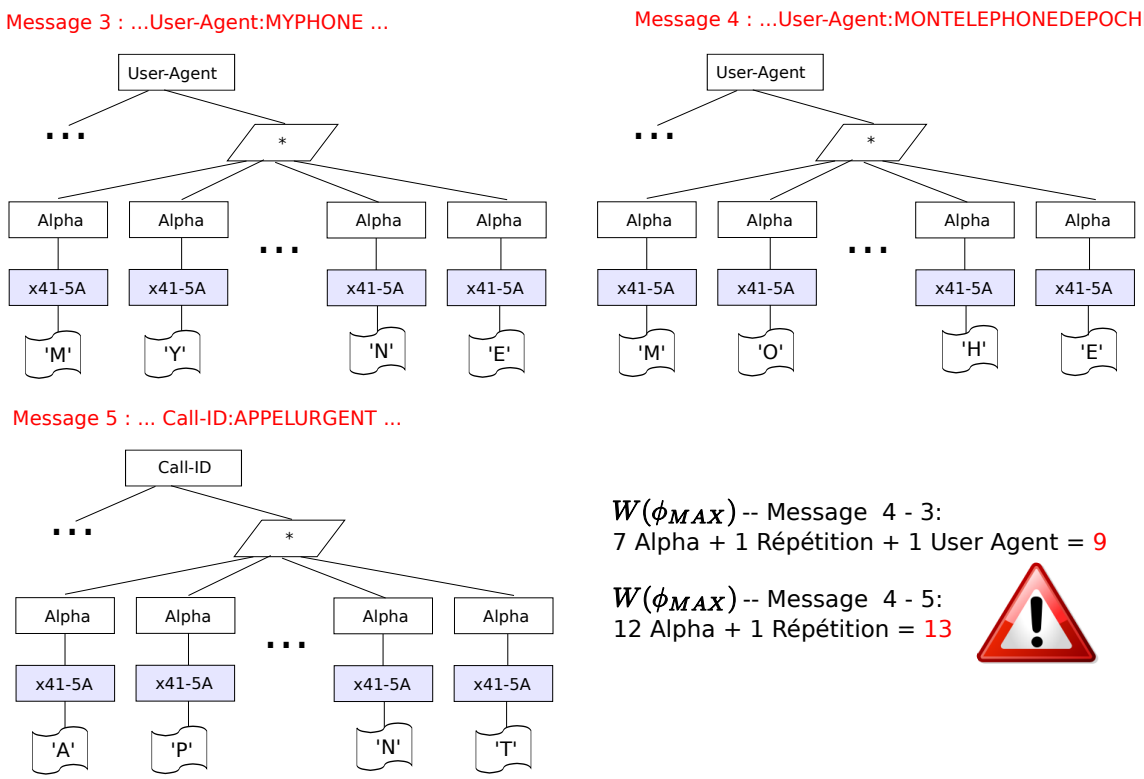


FIGURE 6.9 – Biais introduit par les structures trop imposantes

Pour palier ce problème, un nœud se caractérise par son type mais aussi par le chemin commençant de la racine jusqu'à lui c'est-à-dire l'ensemble de ses ancêtres. La fonction $\alpha(n)$ retourne un couple $< nom, r \bowtie n >$ où *nom* est le nom du nœud n (le nom d'un non terminal ou le caractère « ? » pour une séquence ou une répétition) et r la racine de l'arbre général. Étant donné les fonctions $nom(n)$ retournant le nom d'un nœud et $parent(n)$ retournant le parent de n, la fonction $\sigma(u,v)$ mesurant la similarité de deux nœuds est spécifiée comme suit :

$$\sigma(u,v) = \begin{cases} 1 & si\ u = r \wedge v = r \\ 1 & si\ nom(u) = nom(v) \wedge \sigma(parent(u), parent(v) = 1 \\ 0 & sinon \end{cases} \quad (6.12)$$

En pratique

En théorie, le calcul de la similarité entre deux arbres t_1 et t_2 impose de découvrir tous les isomorphismes de sous-arbres. Cependant, les contraintes précédentes exigent que les nœuds des sous-arbres partagent les même ancêtres. La définition récursive de la formule 6.12 indique bien que tous les ancêtres doivent être similaires. Ainsi, l'isomorphisme de sous-arbres entre t_1 et t_2 engendrant la similarité maximale se restreint à identifier un isomorphisme de sous-arbres enracinés sur la racine originale, les autres étant forcément sous-optimaux.

Dans un premier temps, les chemins entre la racine et tous les nœuds possibles de t_1 et t_2 sont calculés à l'instar de la figure 6.10 où les points représentent une arête et les points d'interrogation un nœud

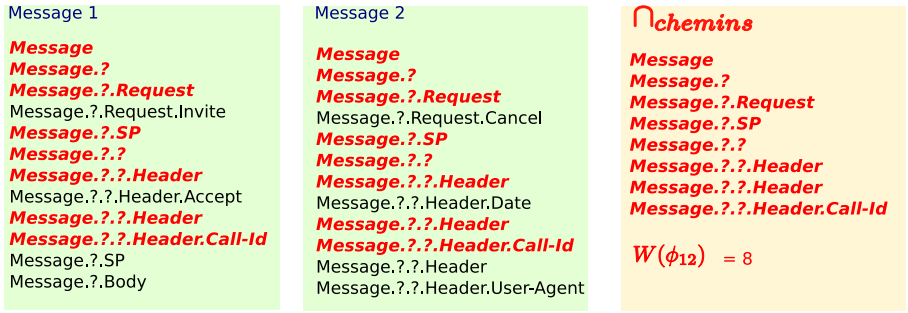


FIGURE 6.10 – Calcul de la similarité en déterminant l'intersection des chemins des différents arbres

séquence ou répétition. Extraire l'ensemble de ces chemins est réalisable en parallèle de la construction de l'arbre syntaxique n'impliquant alors aucune complexité supplémentaire.

La similarité découlant des sous-arbres isomorphiques est équivalente au nombre de nœuds similaires, c'est-à-dire de même nom, qu'ils partagent (formule 6.12). Par conséquent, ces sous-arbres doivent contenir exactement les mêmes chemins. Ainsi, l'intersection $\cap_{chemins}$ est l'ensemble des chemins identiques entre les deux arbres bien que ce ne soit un ensemble au sens mathématique du terme mais plutôt une liste car elle peut contenir des doublons comme présenté dans la figure 6.10. Un chemin du premier arbre peut également correspondre à plusieurs chemins dans le second mais un seul figurera logiquement dans l'intersection. De manière générale, l'algorithme 4 réalise cette intersection en itérant sur chaque chemin de l'arbre t_1 et il sera ajouté dans $\cap_{chemins}(t_1,t_2)$ s'il en existe un identique dans les chemins de t_2 tout en supprimant ce dernier pour ne pas l'inclure plusieurs fois.

Finalement, ces chemins partageant la même racine font ainsi partie du même sous-arbre et hormis pour le nœud racine, chaque chemin est égal à un autre à l'exception du dernier nœud. Ainsi la cardinalité de l'intersection soit $|\cap_{chemins}(t_1,t_2)|$ correspond à la similarité entre les deux arbres originaux t_1 et t_2. Dans le pire des cas, la complexité de cet algorithme est $O(|t_1||t_2|)$.

6.5.3 Technique supervisée

Le fingerprinting syntaxique supervisé emploie la méthode SVM multi-classes de la section 6.3 prenant comme entrées les arbres syntaxiques ou plus précisément les distances associées (voir figure 6.11) :

1. les messages du jeu d'apprentissage sont convertis et stockés l'un après l'autre sous forme d'une liste de chemins,
2. la similarité entre chaque couple d'arbres est calculée équivalente au nombre d'éléments dans la liste des chemins communs (intersection),
3. l'ensemble des distances est calculés entre chaque couple d'arbres pour pouvoir entraîner SVM
4. SVM en déduit la fonction de classification,
5. cette fonction classe alors chaque message du jeu de test un par un.

La distance normalisée utilisée est d_3. Cependant, SVM nécessite une mesure de similarité et non une distance d'où $d'_3 = 1 - d_3$. La distance non normalisée est d_1 dont les valeurs sont ramenées volontairement entre 0 et 1 pour un paramétrage plus facile : $d'_1 = e^{-0.01 d_1}$. Alors que la signature d'un message

Algorithme 4 similarité(t_1,t_2)

1: $res = [\,]$ est le résultat initialisé à une liste vide
2: $chemins(t)$ retourne la liste des chemins de l'arbre t
3: $l.ajouter(e)$ ajoute l'élément e à la liste l
4: $long(l)$ retourne la longueur de la liste l
5: $enlever(l,e)$ enlève l'élément e de la liste l
6: $c_1 = chemins(t_1)$
7: $c_2 = chemins(t_2)$
8: **pour** $c \in c_1$ **faire**
9: $i \leftarrow 1$
10: $bool \leftarrow VRAI$
11: **tantque** $bool \wedge i < long(c_2)$ **faire**
12: **si** $c = c_2[i]$ **alors**
13: $bool = FAUX$
14: $res = res.ajouter(c)$
15: $enlever(c_2,c)$
16: **finsi**
17: $i \leftarrow i+1$
18: **fin tantque**
19: **fin pour**

est son arbre syntaxique, l'empreinte d'un type d'équipements est l'ensemble des signatures correspondantes construites pendant la phase d'apprentissage.

6.5.4 Technique non supervisée

ROCK

Alors qu'il est possible d'évaluer la distance entre deux arbres syntaxiques, ceux-ci restent toutefois des données catégoriques pour lesquelles les techniques usuelles de classification demeurent inadaptées. En effet, la méthode des K-means, des K-medoid ou plus généralement celles de partitionnement hiérarchique ou celles évaluant la densité des points de données dans l'espace sont bien conçues pour les données numériques telles que les distributions définies au chapitre précédent [4, 22].

Dans le cas de la classification d'arbres, l'usage d'algorithmes spécifiques aux données catégoriques est essentiel. ROCK [22] fait partie de cette famille d'algorithmes permettant une segmentation efficace des points de données en introduisant la notion de voisinage, défini par une distance maximale, entre deux points. Étant donnée une fonction de distance d, les arbres t_1 et t_2 sont voisins si et seulement si $d(t_1,t_2) < \tau$ où τ est un seuil paramétrable. Jusqu'à ce point, l'algorithme ressemble de très près à un algorithme de partitionnement hiérarchique sauf que l'étape suivante ne regroupe pas les points voisins entre eux mais seulement ceux qui partagent des voisins en commun. Ce processus est réitéré en prenant en compte non plus la distance entre deux points mais la distance générale entre deux clusters. Cette distance nommée *score* considère le nombre total de voisins entre toutes paires possibles de points de chaque cluster mais aussi une estimation théorique de la valeur maximale de ce nombre. L'objectif de cette normalisation est d'éviter que seuls les gros clusters soient fusionnés impliquant un risque élevé d'obtenir un grand cluster et de multiples petits. Plus exactement, ROCK établit un graphe où chaque

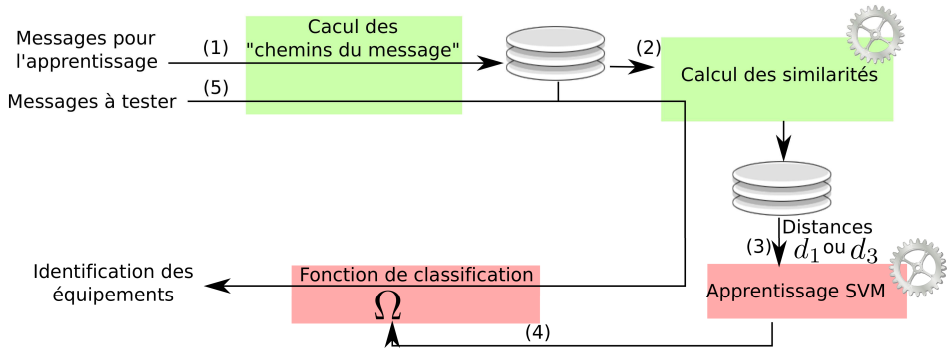


FIGURE 6.11 – Fingerprinting syntaxique supervisé

nœud est un cluster (un point à l'étape initiale) et où une arête pondérée indique le nombre de voisins communs entre deux clusters.

L'algorithme original suppose connu le nombre de clusters et s'arrête donc quand ils sont trouvés. La figure 6.12 est un exemple de clusters de différentes formes. Le chapitre précédent avait souligné l'importance du choix d'un algorithme selon la forme des clusters (figure A.2 en annexe) d'où l'introduction de techniques modernes cherchant à découvrir des formes irrégulières. SVC fait partie de cette famille mais n'est pas applicable dans le cas des arbres syntaxiques car sa dernière étape déterminait si deux points faisaient partie du même cluster en générant un échantillon de points intermédiaires. Alors que cette tâche est facile dans le cadre de valeurs numériques (combinaison convexe), elle l'est beaucoup moins avec des arbres car elle reviendrait à générer des arbres en utilisant des transformations simples à l'instar de la distance d'édition. Le gain de complexité atteint grâce aux distances définies précédemment serait alors tout à fait annihilé d'où la nécessité d'un autre algorithme tel que ROCK. Ainsi dans l'exemple de la figure A.3(a) en annexe, il est capable de dissocier les clusters malgré leurs formes irrégulières et les faibles distances qui les séparent.

En effet, un algorithme classique de partitionnement grâce au saut minimal aurait sans doute fusionner les clusters X et O peu séparés alors que l'utilisation d'un barycentre n'aurait pas forcément bien délimité leur frontière et risquerait de ne pas découvrir la forme irrégulière de X. Cependant, ROCK détecte que les nœuds frontières entre X et O ne partagent aucun voisin commun et ne seront donc pas inclus dans le même cluster. Les points A et B satisfont cette condition les fusionnant malgré tout. En conséquence, ROCK utilise la mesure *score* pour rassembler en priorité les points partageant le plus de voisins tels que C et D sur la figure.

Enfin, il est important de noter l'existence d'autres méthodes comme par exemple CURE qui, pour éviter les inconvénients des méthodes du saut minimal ou du barycentre, représente un cluster par un nombre limité de points. ROCK évalue en fait la densité de voisins de deux nœuds, se rapprochant ainsi des algorithmes basés sur la mesure de densité de points tel que DBScan dans le cas de valeurs numériques. Une vue plus complète de l'ensemble de ces algorithmes ainsi que leurs cas d'utilisation est consultable dans [4].

QROCK

L'inconvénient majeur de ROCK est une complexité assez élevée [5] car à chaque fois qu'un cluster est modifié, son *score* avec tous les autres est recalculé et une liste de score triée doit être maintenue à jour pour pouvoir choisir les bons clusters à fusionner en priorité. De plus, il nécessite de connaître à

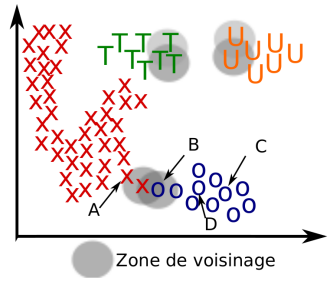
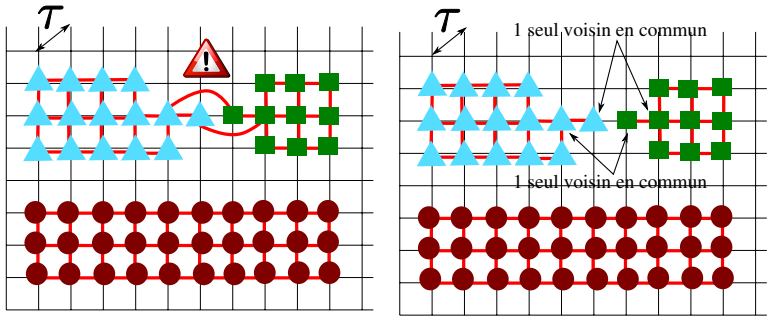


FIGURE 6.12 – Classification avec ROCK



(a) QROCK

(b) Nouvelle version en imposant au minimum deux voisins communs pour établir un lien entre deux nœuds

FIGURE 6.13 – Graphe des points partageant des voisins avec ROCK

priori le nombre de clusters alors que cette donnée n'est généralement pas disponible telle que dans cette étude. QROCK [5] est une version beaucoup plus rapide créant un cluster à partir de chaque composante connexe du graphe construit initialement par ROCK. Plus précisément, chaque sommet de ce graphe est un point du jeu de données et il existe alors une arête entre deux sommets s'ils partagent au moins un voisin (complexité de la construction en $O(n^2)$). Un algorithme de recherche des composantes connexes est clairement plus rapide que ROCK. Cependant, la densité du voisinage n'est plus du tout prise en compte engendrant ainsi la fusion des points A et B sur la figure 6.12. Ce problème est clairement illustré sur la figure 6.13(a) où les points sont disposés sur une grille pour une meilleure lisibilité. Une distance τ égale à la largeur d'une case est obligatoire pour espérer regrouper quelques points impliquant inévitablement la fusion des deux clusters supérieurs (le même phénomène pourrait se « propager » via une « chaîne » de points plus longue). À contrario, en imposant une distance τ égale à la diagonale et deux nœuds en commun minimum, les clusters sont bien séparés.

Compromis

Les problèmes soulevés par la définition originale et QROCK imposent de combiner les avantages de chacun :

– garder la notion de densité apparaissant dans ROCK,
– éviter de calculer des métriques trop complexes pour ordonnancer les regroupements.

La complexité de l'algorithme ROCK émane de son processus de regroupement nécessitant de recalculer le *score* d'un cluster par rapport à tous les autres à chaque fois qu'il est modifié sans oublier de maintenir ordonnée la liste des scores. Le nouveau compromis instauré ne recalculera pas le *score* à chaque étape car il correspond au nombre maximum de voisins en commun entre deux nœuds de chaque cluster C_1 et C_2 soit plus formellement :

$$score(C_i, C_j) = good(C_i, C_j) = max_{p_t \in C_i, p_l \in C_j} \#voisins(p_t, p_l)$$

où $\#voisins(p_t, p_l)$ est le nombre de voisins partagés par p_t et p_l. A première vue cette métrique nécessite aussi d'être recalculée à chaque étape de la classification. Cependant, similairement à QROCK, le nombre de clusters n'est pas fixé à priori et cette nouvelle version regroupe les clusters tant que ce score est au dessus d'un certain seuil γ. Ainsi, cela revient à extraire les composantes connexes du graphe initialement construit dont les arêtes seront pondérées avec un poids supérieur à γ. En conséquence, cette

nouvelle version de ROCK n'évalue pas *good* à chaque fois et nécessite encore moins une normalisation puisqu'elle se base exclusivement sur deux points.

Théorème *Le classement établi par la nouvelle version de ROCK est indépendante de l'ordre de fusion des points.*

La preuve est directe puisqu'il équivaut à un calcul de composantes connexes mais justifie l'intérêt de cette nouvelle version car ROCK devait initialement maintenir une liste triée de scores pour être déterministe. La complexité est équivalente à celle de QROCK car ces algorithmes calculent la même chose à une condition prêt sur les arêtes. Toutefois, le problème de QROCK souligné sur la figure 6.13(a) ne se pose plus comme le démontre la figure 6.13(b). L' algorithme 5 remplit la matrice d'adjacence en calculant pour tout couple de sommets le nombre de voisins en commun. Cet algorithme recherche donc les voisins existants (lignes 5-7) puis en considère un comme le voisin commun pour tenter d'en trouver un autre.

Algorithme 5 Calcul de la matrice d'adjacence

1: $T = \{t_1, \ldots, t_N\}$ l'ensemble des arbres à classer
2: $d(t_i, t_j)$ est la distance entre les arbres t_i et t_j
3: τ est la distance maximale entre deux voisins
4: M_{ij} est le nombre de voisins communs entre t_i et t_j initialisé à 0
5: **pour** $i \leftarrow 1 \ldots N$ **faire**
6: **pour** $j \leftarrow 1 \ldots N$ **faire**
7: **si** $d(t_i, t_j) < \tau$ **alors**
8: **pour** $k \leftarrow 1 \ldots N$ **faire**
9: **si** $k \neq \wedge k \neq j \wedge d(t_i, t_j) < \tau$ **alors**
10: $M_{ij} = M_{ij} + 1$
11: **finsi**
12: **fin pour**
13: **finsi**
14: **fin pour**
15: **fin pour**

Dans un second temps, l'algorithme 6 détermine les composantes connexes en associant à chaque arbre à classer, un booléen initialisé à *FAUX* indiquant qu'il n'est encore inclus dans aucun cluster. L'algorithme cherche donc ce type d'arbre pour initialiser un cluster puis recherche ceux qui partagent avec lui au moins γ voisins qui seront alors ajoutés au nouveau cluster. Une procédure similaire est alors appliquée récursivement sur chacun d'eux pour « parcourir » l'ensemble des arbres formant la composante connexe pondérée.

Mise en œuvre

La technique non supervisée du fingerprinting syntaxique exécute l'algorithme ROCK modifié sur les arbres syntaxiques dont la distance pour évaluer si deux nœuds sont voisins s'appuie sur une des distances introduite dans la section 24. La figure 6.14 récapitule les différentes tâches réalisées dont le premières sont les mêmes que pour le cas supervisé :

1. les messages du jeu d'apprentissage sont convertis et stockés sous forme d'une liste de chemin l'un après l'autre,

Algorithme 6 classification

1: $T = \{t_1, \ldots, t_N\}$ l'ensemble des arbres à classer
2: M_{ij} est le nombre de voisins communs entre t_i et t_j
3: $init(t)$ crée un nouveau cluster contenant exclusivement t_i
4: $c.add(t)$ ajoute l'arbre t au cluster c
5: $Label_i$ indique si t_i est déjà intégré dans un cluster et est initialisé à $FAUX$
6: **pour** $i \leftarrow 1 \ldots N$ **faire**
7: **si** $\neg Label_i$ **alors**
8: $c = init(t_i)$
9: $Label_i = VRAI$
10: **pour** $j \leftarrow 1 \ldots N$ **faire**
11: **si** $i \neq j \wedge M_{ij} > \gamma \wedge Label_j = FAUX$ **alors**
12: clustering(j,c)
13: **finsi**
14: **fin pour**
15: **finsi**
16: **fin pour**
17: **clustering**(k,cluster) :
18: $Label_k = VRAI$
19: c.add(t_k)
20: **pour** $j \leftarrow 1 \ldots N$ **faire**
21: **si** $k \neq j$ and $M_{kj} > \gamma$ and $Label_j = FAUX$ **alors**
22: clustering(j,c)
23: **finsi**
24: **fin pour**

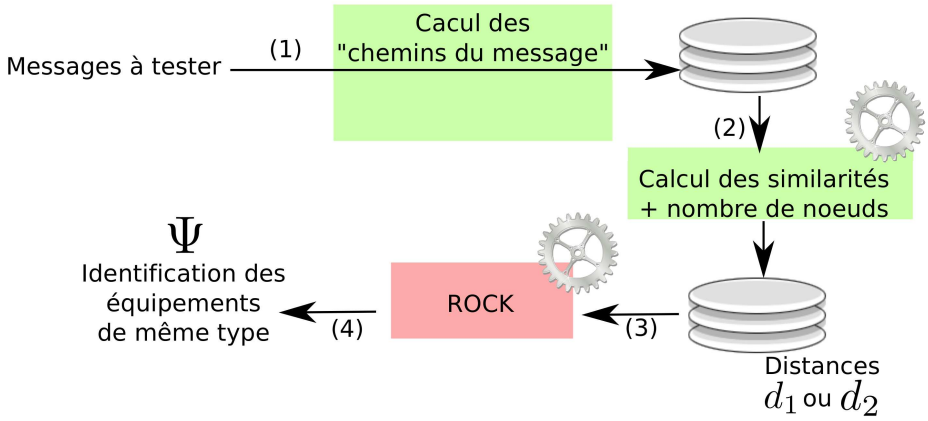


FIGURE 6.14 – Fingerprinting syntaxique non supervisé

2. la similarité entre chaque couple d'arbres est calculée ainsi que leur cardinalité équivalent au nombre d'éléments dans la liste des chemins,
3. l'ensemble des distances est calculé entre chaque couple d'arbres,
4. l'algorithme ROCK est exécuté sur ces distances,
5. la sortie de ROCK identifie les groupes d'équipements supposés du même type.

La distance normalisée utilisée est d_3. Cependant, SVM nécessite une mesure de similarité et non une distance d'où $d'_3 = 1 - d_3$. La distance non normalisée est d_1 dont les valeurs sont ramenées volontairement entre 0 et 1 pour un paramétrage plus facile : $d'_1 = e^{-0.01 d_1}$. L'empreinte d'un équipement est l'ensemble des arbres syntaxiques assignés au même cluster.

6.6 Conclusion

Ce chapitre a détaillé trois nouvelles techniques de fingerprinting reposant sur deux modélisations différentes. La première d'entre elle a introduit la formalisation des TR-FSM c'est-à-dire des arbres reflétant le comportement d'un équipement en termes de messages échangés mais aussi une notion temporelle en prenant en compte les délais pour envoyer un message. L'utilisation de cette nouvelle caractérisation est applicable aux SVM grâce à la définition d'une fonction noyau spécialement adaptée.

La structure syntaxique d'un message est très représentative car elle ne se limite pas à la valeur des champs ou à la structure à plat du message facilement modifiables. Ce gain d'information est donc considérable et permet, en plus d'un fingerprinting supervisé, de réaliser un fingerprinting totalement non supervisé grâce à la définition d'un nouvel algorithme de classification.

Que ce soit le gain d'information inféré par la structure syntaxique d'un message ou par le comportement temporel d'un équipement, ces techniques restent néanmoins passives et se différencient clairement des autres approches proposées. Seul [8] qui a inspiré l'utilisation d'arbres syntaxiques est proche mais se démarque des travaux présentés ici par la construction d'un arbre générique pour chaque type d'équipement nécessitant un temps d'apprentissage très long par rapport à la technique présentée ici (le chapitre 9 donnera plus de détails).

Troisième partie

Evaluation

7

Performances des botnets

Sommaire

7.1 Introduction . . . 149
7.2 Réseaux IRC . . . 150
7.2.1 Premier modèle . . . 151
7.2.2 Paramètres . . . 151
7.2.3 Atteignabilité moyenne . . . 152
7.2.4 Nombre de sauts . . . 153
7.2.5 Nombre de nœuds . . . 155
7.3 Paramètres des modèles pair-à-pair . . . 156
7.4 Slapper . . . 156
7.4.1 Nombre de sauts . . . 156
7.4.2 Nombre de nœuds . . . 157
7.5 Chord . . . 158
7.5.1 Nombre de sauts . . . 159
7.5.2 Distance entre deux identifiants *d* . . . 160
7.5.3 Nombre de nœuds . . . 161
7.5.4 Atteignabilité moyenne . . . 161
7.6 Comparaison . . . 162
7.6.1 Facteur de branchement . . . 162
7.6.2 Impact des attaques . . . 162
7.6.3 Délais . . . 163
7.7 Conclusion . . . 164

7.1 Introduction

Les performances envisageables par les modèles introduits dans le chapitre 4 font l'objet de celui-ci. Pour rappel, l'objectif pratique visé est une supervision à large échelle tel que l'envoi d'un message (une requête) par le contrôleur (superviseur) en broadcast à un grand nombre d'équipements. Les botnets, outils des attaquants, ont semblé être un choix judicieux mais qui nécessite quand même d'être prouvé. Ainsi, ce chapitre étudiera trois architectures de botnets possibles pour déterminer leur viabilité et les comparer selon de nombreuses configurations :

– réseau IRC,
– réseau P2P de type Slapper,
– réseau P2P de type Chord.

D'un point de vue technique, les deux premiers reposent sur des modèles mathématiques évalués grâce à l'outil Maxima [9] permettant de faire des calculs formels et plus précisement de calculer des séries. Le dernier est quant à lui un modèle algorithmique programmé en Python [5].

Avant d'entrer dans le vif du sujet, les différents paramètres et métriques des modèles sont brièvement rappelés ci-après. L'atteignabilité est la métrique de base permettant d'évaluer la performance d'un botnet dans le cadre de la supervision à large échelle puisqu'elle représente une estimation du nombre de bots à une distance maximale k du superviseur recevant les requêtes de ce dernier lorsqu'il les envoie en broadcast. Les modèles reposent sur la définition de plusieurs paramètres pouvant avoir un impact sur cette métrique :

– le nombre de nœuds N au total dans le réseau,
– la distance maximale k de propagation de la requête en nombre de sauts par rapport au superviseur,
– le facteur de branchement maximal m ou connectivité qui correspond au nombre maximal de connexions associées à un nœud. Ce paramètre est équivalent au nombre de nœuds vers lesquels un nœud peut envoyer une requête.

Un nœud représente soit un serveur IRC soit un bot P2P selon le modèle utilisé. De manière générale, les expérimentations évalueront l'atteignabiltié avec N ou k fixé tandis que l'autre variera.

Il est aussi nécessaire de rappeler que les modèles tiennent compte de deux types de problèmes :

– la probabilité de panne calculable en prenant la probabilité inverse du facteur de disponibilité $\alpha(m)$,
– le probabilité β pour un nœud d'être compromis.

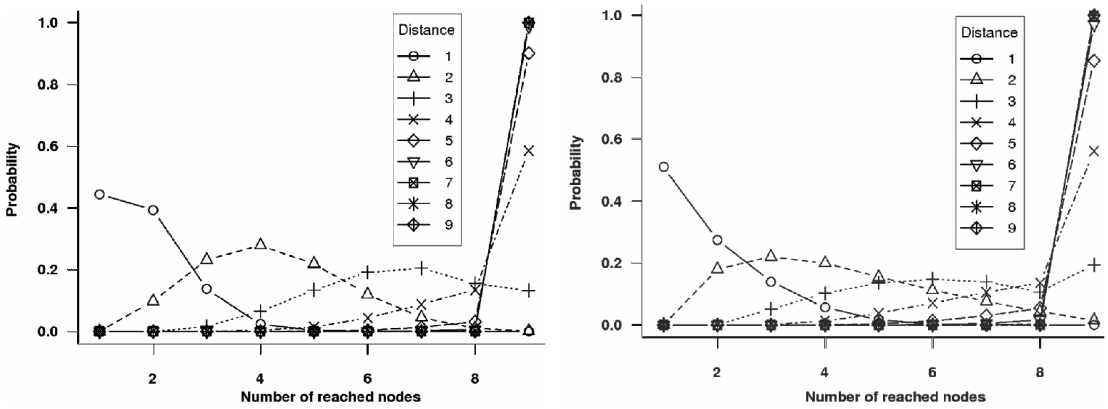
Enfin, l'atteignabilité absolue est une autre métrique calculable et permet de connaître en moyenne le nombre de bots atteints quelque soit la distance. Bien entendu, il est nécessaire de fixer une distance maximale désignant le plus long chemin que peut parcourir une requête :

– dans le cas du modèle IRC, le pire cas est un arbre linéaire impliquant une distance maximale N,
– la diffusion aléatoire dans Slapper ne permet pas de calculer cette métrique,
– le nœud superviseur de Chord peut atteindre tout nœud en $log(N_{MAX})$ sauts où N_{MAX} est la taille de l'espace des identifiants propres à Chord uniquement.

L'intérêt de l'atteignabilité moyenne réside dans son aptitude à mettre en évidence les différences générales de performance entre les différents facteurs de branchement selon le nombre de nœuds du réseau. Dans un second temps, il est néanmoins nécessaire de prendre en compte le nombre de sauts et plus seulement une moyenne pour mettre en avant des propriétés plus détaillées.

7.2 Réseaux IRC

Les études sur les botnets IRC vont essentiellement se consacrer au second modèle introduit dans le chapitre 4 malgré une rapide évaluation du premier dans la première sous-section. De plus, il ne faut pas oublier que les modèles IRC se basent uniquement sur les serveurs constituant le cœur d'une telle architecture.



(a) La requête est envoyée par le nœud 5 (b) Moyenne sur tous les noeuds

FIGURE 7.1 – Premier modèle IRC — Atteignabilité selon différentes distances et 10 serveurs dans le réseau

7.2.1 Premier modèle

Le premier modèle construit présente plusieurs faiblesses telles que sa complexité rendant son utilisation difficile. Ainsi, peu de cas furent testés avec ce modèle et cette sous-section présentera quelques résultats simples seulement. En effet, la figure 7.1(b) représente les distributions de probabilité d'atteindre un certain nombre de serveurs à une distance maximale donnée sachant que le réseau est composé de 10 serveurs IRC. L'axe des abscisses est au maximum égal à neuf car le dixième est celui d'origine d'où est envoyée la requête de supervision. A titre d'exemple, la probabilité de joindre quatre serveurs à 3 sauts au plus est proche de 10%. Pour les distances élevées, la plupart des nœuds sont atteints comme le montre les pics de probabilité proches de 1 pour 9 nœuds. D'ailleurs, il est important de remarquer qu'à partir d'une distance de 6 sauts, les résultats sont similaires ce qui signifie qu'une requête peut être diffusée à l'ensemble du botnet en maximum six sauts dans cette configuration donnée (10 serveurs).

Enfin, cette courbe demeure la moyenne des probabilités sur tous les nœuds d'origine possibles et évaluer les performances à partir d'un nœud donné est aussi envisageable. Malgré sa complexité extrêmement gênante, c'est ainsi l'unique modèle qui peut servir à mesurer l'efficacité du placement du superviseur dans le réseau à l'instar du nœud 5 sur la figure 7.1(a).

7.2.2 Paramètres

Disponibilité d'un nœud

Pour rappel, la disponibilité d'un nœud schématise la capacité d'un nœud à pouvoir faire suivre les messages qui lui sont expédiés. Plus un nœud est connecté à d'autres, plus il existe de sources et de destinations pour les messages transitants. Ainsi, $\alpha(m)$ doit être une fonction décroissante comprise entre 0 et 1 pour m variant entre deux et l'infini car le cas $m = 1$, correspondant à une topologie de deux nœuds seulement, est exclu. Plusieurs définitions de $\alpha(m)$ sont envisageables :

- $\alpha(m) = \alpha_1(m) = 1/m$
- $\alpha(m) = \alpha_2(m) = e^{(i-m)}$

La deuxième fonction décroît moins rapidement mais est paramétrée par *i* devant être fixé de manière à obtenir des valeurs assez élevées lorsque le facteur de branchement *m* est petit. Par exemple, $i = 3$ sera la valeur adoptée dans le cas d'expériences avec trois comme borne inférieure de *m*. Ce sera d'ailleurs le cas considéré par la suite. Un tel α équivaut alors à un environnement instable privilégiant les nœuds à faible connectivité car maintenir actives de nombreuses connexions est difficile.

La probabilité pour un nœud d'être découvert et compromis est $\beta = 0.01$. Ces différents paramètres ont été établis arbitrairement mais placent le botnet dans un environnement hostile car, par exemple, pour seulement 20 serveurs IRC, β induit une probabilité d'être totalement compromis de 20 % comme expliqué dans la section 2. En réalité, ces paramètres nécessitent d'être adaptés aux contraintes spécifiques de l'environnement dans lequel le système est déployé. Il est alors possible d'instancier les modèles grâce à des observations réelles réalisées par les méthodes présentées dans le chapitre 1. En effet, en exécutant simplement le programme bot sur une machine instrumentée, la connectivité (facteur de branchement *m*) est facilement déterminable de même que sa disponibilité en analysant la manière dont la machine réagit face à la charge. Par exemple, observer le nombre de messages, la consommation de bande passante ou la charge du processeur en parallèle de la connectivité est concevable pour mener à bien cette tâche.

Dans l'optique de rester le plus général possible, les résultats présentés ci-après supposent un réseau hostile comme expliqué précédemment.

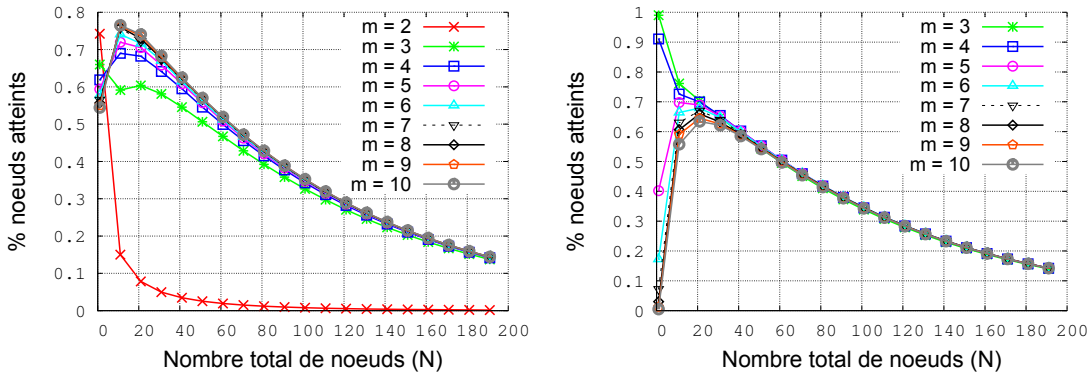
7.2.3 Atteignabilité moyenne

La première expérimentation évalue l'atteignabilité selon le nombre total *N* de serveurs dans le réseaux et le facteur de branchement *m*. La figure 7.2(a) suppose $\alpha(m) = \alpha_1(m)$ et mesure la proportion de serveurs IRC recevant une requête envoyée par le superviseur (atteignabilité moyenne). L'atteignabilité progresse similairement quelque soit le facteur de branchement avec, en premier lieu, une augmentation. En effet, une requête pourra être logiquement transmise à plus de serveurs dès lors qu'ils sont plus nombreux. À partir de $N = 10$, toutes les courbes amorcent une diminution et se confondent malgré l'augmentation constante du nombre de serveurs par le fait qu'un serveur compromis (probabilité β) impacte l'ensemble du réseau IRC. Par conséquent, la dégradation de l'atteignabilité est due à une plus forte probabilité que le réseau soit compromis comme le rappelle le terme $(1-\beta)^N$ de la formule (4.7) introduite dans la section 4.4.3. Une bonne atteignabilité n'est pas forcément synonyme d'un grand nombre de nœuds car elle doit être modérée selon les risques d'attaques. Dans le cas présent, un arbre de diffusion de plus de dix serveurs ne semble donc pas raisonnable. Le facteur de branchement comptabilise les nœuds pères et enfants qu'un nœud peut avoir au maximum. Ainsi $m = 2$ est l'équivalent d'une chaîne d'où une atteignabilité désastreuse et ne sera donc plus étudié par la suite.

Sous l'hypothèse $\alpha(m) = \alpha_2(m)$, la figure 7.2(b) se démarque de la précédente essentiellement pour des valeurs faibles de *N* pour lesquelles l'atteignabilité est une fonction croissante du facteur de branchement maximal, c'est-à-dire $\alpha_2(m) > \alpha_1(m)$. L'objectif d'avoir une fonction $\alpha(m)$ décroissante sans pour autant être trop restrictive est donc bien atteint. Les tests suivants prennent uniquement en compte $\alpha(m) = \alpha_2(m)$. D'après la figure 7.2, l'atteignabilité est contrainte par le facteur $\alpha(m)$ pour des faibles valeurs de N (<20) alors qu'elle est soumise aux effets des attaques pour un nombre de serveurs plus élevé. Cette propriété peut s'énoncer de la façon suivante :

$$avg_att(N,m) = \begin{cases} f_1(\alpha(m),N) & , N \leq 20 \\ f_2(\beta,N) & sinon \end{cases} \tag{7.1}$$

où f_1 et f_2 sont les fonctions retournant les valeurs des courbes de la figure 7.2.



(a) $\alpha(m) = \alpha_1(m)$ (b) $\alpha(m) = \alpha_2(m)$

FIGURE 7.2 – Modèle IRC — Atteignabilité selon différents facteurs de branchement

Par ailleurs, la figure 7.3(b) détaille l'atteignabilté absolue moyenne pour N petit et confirme l'observation générale de la figure 7.2(b) où les facteurs de branchement les plus faibles aident à obtenir une meilleur efficacité. Ainsi, comme le montre la définition de f_1, l'atteignabilité est contrainte par deux paramètres qui, de plus est, sont antagonistes : le facteur de branchement m et la probabilité de panne pour une machine (non disponibilité) : $1 - \alpha(m)$.

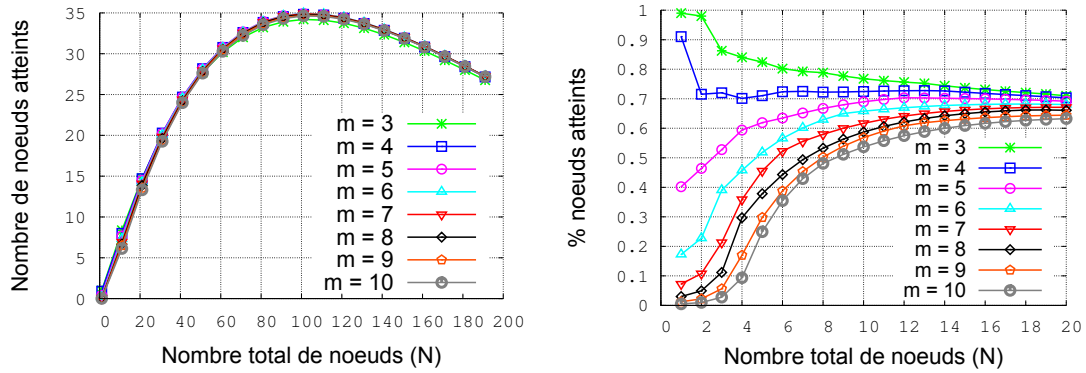
L'atteignabilité absolue sur la figure 7.3(a) décrit le nombre moyen de serveurs atteints quelque soit la distance considérée. Les faibles différences exposées par les précédentes courbes ne se répercutent pas sur le nombre absolu de serveurs joignables et une valeur maximale de 35 serveurs joignables est mise en évidence. Cependant, cette efficacité demande le déploiement de cent serveurs réduisant ainsi l'impact d'un tel résultat. En revanche, déployer plus de cent serveurs n'est en aucun cas bénéfique puisque que l'atteignabilité moyenne, absolue ou non, diminue pour des valeurs supérieures. Le superviseur préférera donc maximiser une des ces métriques selon les contraintes techniques et financières qui lui sont imposées.

Étant donné 35 serveurs joignables et une moyenne de 100 bots connectés sur chacun d'eux, 3500 machines peuvent être supervisées par l'intermédiaire d'une seule requête. De plus, 100 clients par serveur IRC est une valeur volontairement faible par rapport au nombre d'utilisateurs réels sur de tels serveurs. En outre, les bots sont des clients beaucoup moins actifs et ne font généralement qu'épier un canal IRC pour récupérer les ordres.

7.2.4 Nombre de sauts

Alors que les expérimentations précédentes ne cherchaient pas à déterminer l'atteignabilité par rapport à la distance, les suivantes visent à évaluer le nombre de serveurs joignables selon le nombre de sauts parcourus par une requête. Dans la perspective de tests significatifs, le nombre total de serveurs N varie : 20, 50, 100 et 200. Les figures 7.4(a),7.4(b),7.4(c) et 7.4(d) montrent que l'atteignabilité est toujours plafonnée. En réalité, cela se justifie par la formule (4.7) qui modère l'atteignabilité par le terme $(1-\beta)^N$. Ainsi même si le nombre théorique de nœuds atteints est de 1, l'atteignabilité vaudra au maximum $(1-\beta)^N$. Avec $N = 20$, ce terme vaut 0.81 et est bien égal à la valeur plafond de la figure 7.4(a).

Connaître le nombre de sauts nécessaires permet d'estimer le temps nécessaire pour transmettre une requête. Naïvement, un facteur de branchement élevé, synonyme de forte connectivité, devrait aider à



(a) Atteignabilité absolue

(b) Atteignabilité moyenne pour $1 \leq N \leq 20$

FIGURE 7.3 – Modèle IRC — Nœuds atteints avec $\alpha(m) = \alpha_2(m)$

atteindre rapidement un grand nombre de bots. Cependant lorsque N est petit à l'instar de $N = 20$, l'effet d'un fort facteur de branchement reste marginal par rapport aux risques de surcharge et de déconnexion amplifiés par ce même facteur. De plus, β a un impact équivalent quelque soit le facteur de branchement. Par conséquent, pour N petit, les facteurs de branchement les plus petits sont ceux présentant les meilleurs résultats comme sur la figure 7.4(a). Bien entendu, un facteur de branchement trop faible peut se révéler problématique lorsque N est élevé car il implique un nombre de sauts beaucoup plus grand pour atteindre l'atteignabilité maximale comme le met en exergue la figure 7.4. L'atteignabilité est donc clairement dépendante des quatre paramètres principaux des modèles :

- N et β pour la valeur maximale envisageable,
- m et $\alpha(m)$ qui ont des effets antagonistes et contraignent l'évolution de l'atteignabilité selon le nombre de sauts avant d'obtenir la valeur plafond.

Cependant $m = 7$ n'est pas particulièrement plus mauvais que $m = 10$ dans les cas présentés ici et $m = 5$ semble d'ailleurs être le meilleur compromis d'autant plus que la section précédente a prouvé l'inanité d'avoir $N > 100$. Par ailleurs, 5, 6 ou 7 sauts sont suffisants pour obtenir la meilleure atteignabilité grâce à ce facteur de branchement.

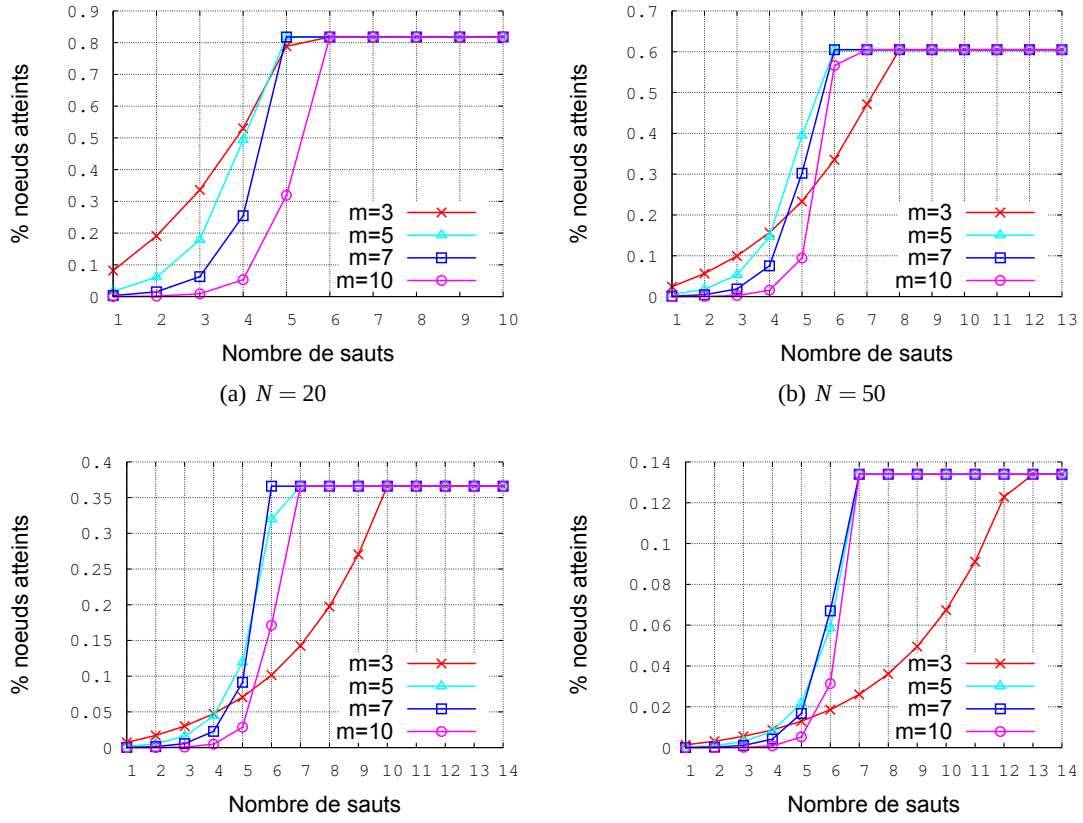
Ce type d'expérimentation estime également le nombre de serveurs atteignables à une distance maximale et donc le temps pour transmettre une requête de supervision grâce à la formule (4.10) rappelée ci-dessous :

$$temps_{botnet} = d_{max} \times t_s + C_{serveur} \times t_h$$

Par exemple, $0.6 \times 50 = 30$ serveurs sont atteints en 6 sauts sur la figure 7.4(b). Une fois de plus, 100 bots sont connectés à chaque serveur en moyenne entraînant une population de 3000 machines. Le délai pour leur expédier la requête est le suivant : $6 \times t_s + 100 \times t_h$. Dans le cas d'une solution centralisée, la même tâche est réalisée dans le laps de temps égal à $3000 \times t_h$ (formule (4.9)). Une architecture de type botnet IRC pour la supervision sera donc préférable à une solution centralisée si et seulement si :

$$\begin{aligned} & 6 \times t_s + 100 \times t_h < 3000 \times t_h \\ \Leftrightarrow\ & t_s < \frac{2900 \times t_h}{6} \simeq 483 \times t_h \end{aligned} \tag{7.2}$$

En outre, le temps t_h pour envoyer un message à un bot et le temps t_s pour l'envoyer à un serveur sont similaires, c'est-à-dire $t_s \simeq t_h$. Par conséquent, cette nouvelle solution de supervision réduit considérablement les délais malgré l'introduction de nœuds intermédiaires.



(a) $N = 20$ (b) $N = 50$

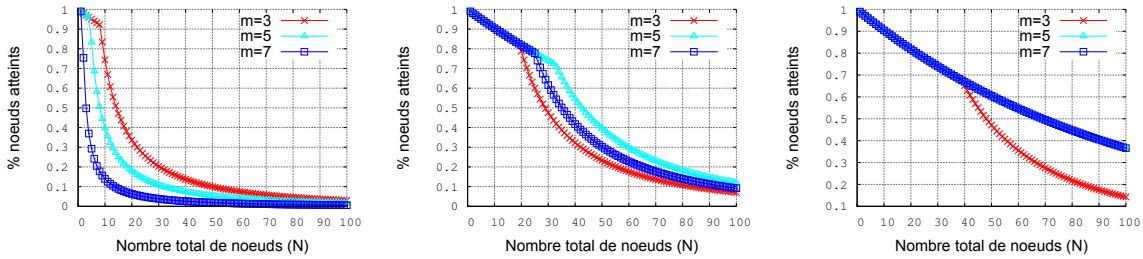
(c) $N = 100$ (d) $N = 200$

FIGURE 7.4 – Modèle IRC — Dépendance entre atteignabilité et facteur de branchement

7.2.5 Nombre de nœuds

Toujours en fixant $\alpha(m) = \alpha_2(m)$, N fluctue beaucoup plus finement dans cette expérimentation que précédemment impliquant ainsi de fixer le nombre de sauts sur la figure 7.5. Désormais, $N = 100$ est considéré comme maximal du fait des précédentes observations. Les courbes sont une nouvelle fois divisibles en deux phases. La première se caractérise par des courbes légèrement décroissantes et exactement identiques. Une fois de plus, elles correspondent à atteindre tous les nœuds tout en prenant en compte la probabilité de compromission β. Dans un second temps, la décroissance s'accélère plus ou moins rapidement selon le facteur de branchement maximal m. Effectivement, comme le nombre de sauts est fixe, l'atteignabilité dépend du facteur de branchement mais aussi de son impact antagoniste via $\alpha(m)$ (disponibilité du nœud).

Enfin, les résultats présentés ici appuient le choix d'un facteur de branchement égal à cinq qui est le meilleur compromis pour obtenir de bonnes performances en général sans être néanmoins le meilleur choix pour chaque cas pris individuellement.



(a) $k = 3$ (b) $k = 5$ (c) $k = 7$

FIGURE 7.5 – Modèle IRC — Atteignabilité selon N et le nombre de sauts j

7.3 Paramètres des modèles pair-à-pair

Pour pouvoir comparer équitablement le modèle IRC avec les modèles pair-à-pair, les paramètres $\alpha(m)$ et β doivent être correctement ajustés. Comme nos précédentes conclusions reposent sur une moyenne de 100 clients par serveur IRC, la probabilité d'être attaqué reflétait également la probabilité d'être attaqué pour chacune des machines connectées. Ainsi, la nouvelle valeur de β est fixée à $\frac{\beta_{IRC}}{100} = \frac{0.01}{100} = 0.0001$. La disponibilité est déterminée de la même manière : $\alpha(m) = e^{\frac{i-m}{100}}$. Dans le cas des études sur les modèles P2P, i sera fixé à différentes valeurs. En effet dans le cas de Slapper, le facteur de branchement minimal étudié est de 2 d'où $i = 2$. Contrairement à un réseau IRC où une telle connectivité impose une structure arborescente en forme de chaîne (le nœud d'arrivée et le nœud de sortie), ce paramètre représente ici un choix aléatoire de 2 nœuds quelconque dans le réseau et est donc possible. Par conséquent, une comparaison des deux modèles nécessitera de prendre en compte un facteur de branchement incrémenté de un pour le modèle IRC dans l'objectif que chaque serveur puisse envoyer au maximum 2 messsages et pas seulement un.

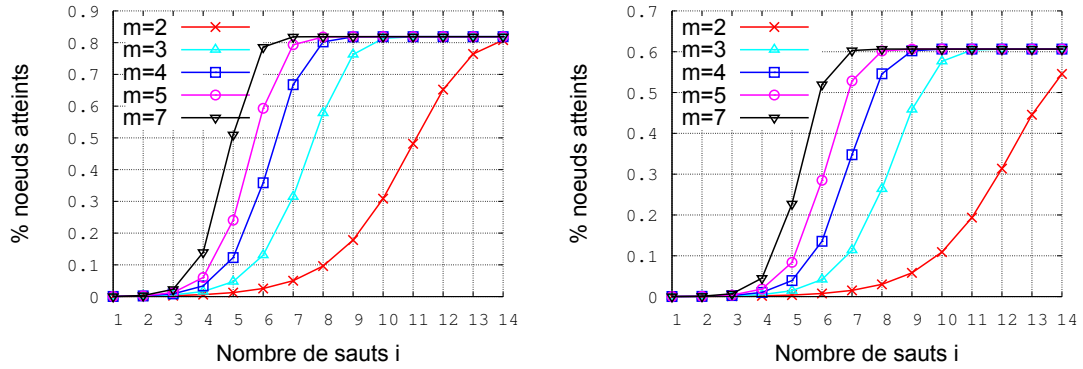
Enfin, fixer m à une valeur quelconque est impossible dans le cas de Chord puisqu'il est égal au nombre d'entrées différentes dans la table de routage de chaque nœud lui-même fixé par $log_2(N)$.

7.4 Slapper

7.4.1 Nombre de sauts

Plusieurs facteurs de branchement sont testés sur la figure 7.6. Comme pour les botnets IRC, l'atteignabilité est plafonnée à une valeur maximale qui, pour les mêmes raisons, est directement déductible à partir de β et ainsi égale au maximum à $(1-\beta)^N$.

Contrairement à IRC, avoir un facteur de branchement élevé est bénéfique d'après les courbes tracées puisque les meilleures performances sont obtenues ainsi. Une autre différence majeure est le nombre de sauts nécessaire pour obtenir l'atteignabilité maximale. En effet, celui-ci est à peu près constant pour un facteur de branchement donné lorsque celui-ci vaut au minimum quatre. Par exemple avec $m = 5$, huit sauts sont suffisants pour atteindre la valeur plafond. Ainsi, limiter efficacement le nombre de nœuds devant faire suivre une requête de supervision est aisée. Pour rappel, dans un réseau de type Slapper, la requête est retransmise par tous les nœuds la recevant jusqu'à ce que celle-ci soit passée par un certain nombre de nœuds intermédiaires. Habituellement, cette valeur est fixée arbitrairement par le contrôleur.



(a) $N = 2000$ (b) $N = 5000$

FIGURE 7.6 – Slapper — Atteignabilité

Avec cette étude, elle peut être estimée automatiquement, évitant ainsi les retransmissions inutiles surchargeant le réseau.

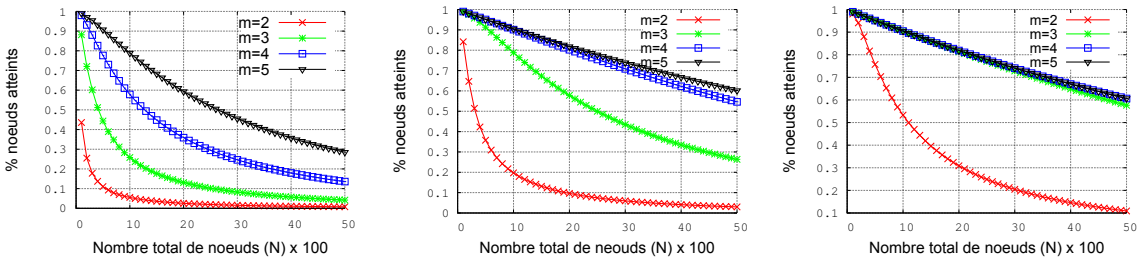
Dans la même optique, une distance maximale de quatre ou cinq sauts est totalement futile car, dans le meilleur des cas, seul 15% des bots recevraient effectivement la requête. Enfin, bien que la distance nécessaire pour atteindre la valeur maximale soit fixe, le nombre total de bots a un effet sur la dynamique de l'atteignabilité. En effet, quand N augmente entre la figure 7.6(a) et 7.6(b), les courbes s'amplifient moins rapidement au début pour des distances faibles et plus fortement pour les distances plus élevées.

7.4.2 Nombre de nœuds

La dépendance entre l'atteignabilité et le nombre total de nœuds N est mise en évidence sur la figure 7.7. Pour cela, la diffusion de la requête est plafonnée à 6, 8 ou 10 sauts. Bien évidemment, l'atteignabilité diminue car, malgré l'augmentation du nombre de nœuds, la distance demeure fixe et restreint donc logiquement le nombre de nœuds atteignables. Il faut bien comprendre qu'il s'agit de la proportion de bots atteints et mesure donc l'efficacité de l'architecture. Une autre mesure est l'atteignabilité absolue de la figure 7.8 où le nombre exact de bots atteints est pris en compte. Dans ce cas, augmenter N permet effectivement d'atteindre plus de bots mais de manière inefficace car les pentes des courbes diminuent au fur et à mesure, preuve d'une atteignabilité décroissante (en proportion). Par ailleurs, une limite, après laquelle les performances sont dégradées, subsiste même si elle ne figure pas sur toutes les figures car N n'est pas assez grand. Ainsi, comme pour IRC, ces courbes peuvent aider à déterminer le nombre maximal de bots réellement contrôlable tout en indiquant au superviseur un nombre à ne pas dépasser qui n'engendrerait qu'un effet néfaste. Par exemple, si le superviseur fixe un facteur de branchement égal à 3 avec au plus huit sauts, il sait également que 4000 ou 5000 bots, au plus, pourront être supervisés avec une telle configuration.

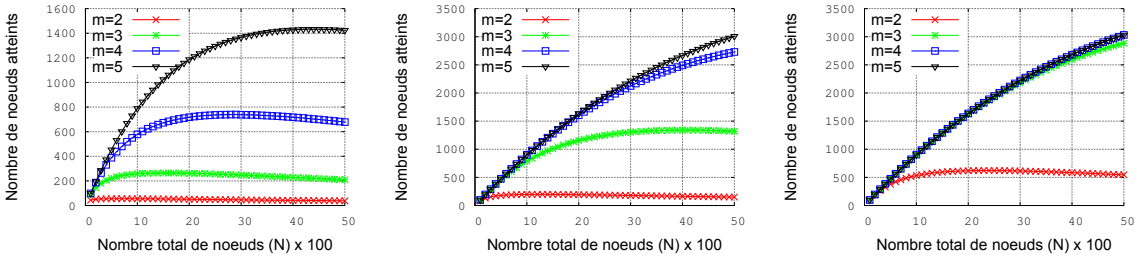
De plus, lorsque N augmente, les courbes associées aux plus grands facteurs de branchement convergent et sont une fois de plus limitées par la probabilité β qu'un nœud soit compromis. Cette observation confirme que pour un nombre de sauts suffisamment élevé, obtenir la meilleure atteignabilité possible est toujours envisageable quel que soit le nombre total de nœuds. Dans le cas contraire, les différences entre les courbes sont dues au facteur de disponibilité $\alpha(m)$.

D'un point de vue pratique, des tests supplémentaires ont montré qu'en continuant d'augmenter le



(a) $k = 6$ (b) $k = 8$ (c) $k = 10$

FIGURE 7.7 – Slapper — Influence du nombre de sauts k sur l'atteignabilité



(a) $i = 6$ (b) $i = 8$ (c) $i = 10$

FIGURE 7.8 – Slapper — Influence du nombre de sauts i sur l'atteignabilité absolue

nombre de sauts ($d > 10$), le facteur de branchement n'avait plus d'impact sur l'atteignabilité. Elle peut alors se schématiser de la façon suivante :

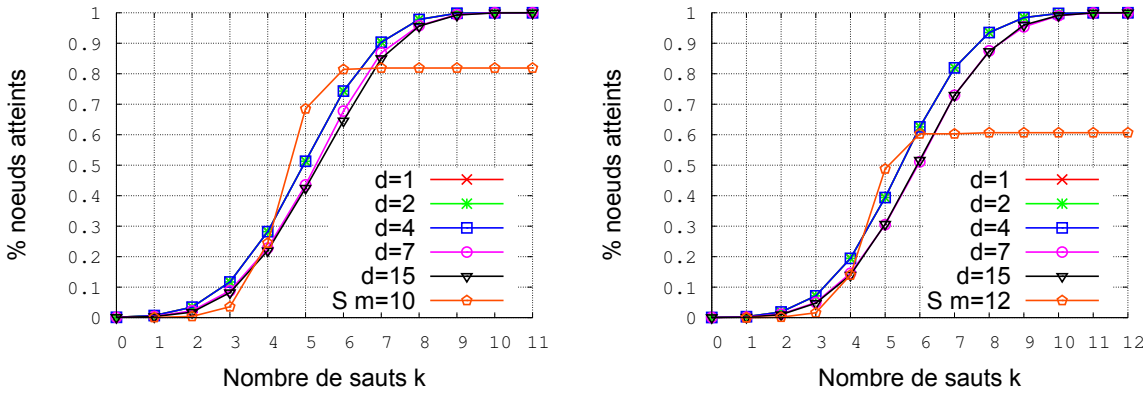
$$att(N,d) = \begin{cases} f_1(m,N) & , d \leq 20 \\ f_2(\beta,N) & sinon \end{cases} \tag{7.3}$$

avec f_1 une fonction croissante selon m et décroissante selon N comme le montre les premières observations ci-dessus tandis que f_2 est une fonction décroissante de N et modérée par β.

Enfin, les performances résultant d'un facteur de branchement $m = 2$ sont désastreuses et la forme générale de l'atteignabilité déduite précédemment n'est pas applicable pour ce cas précis. Alors que cette valeur est la valeur par défaut de Slapper, elle constitue un mauvais choix et est inutilisable en pratique dans une optique de supervision large échelle.

7.5 Chord

Les pairs sont répartis uniformément sur l'anneau de telle sorte que la distance entre les identifiants de deux pairs consécutifs soit en moyenne égale à d. Cette distance ne doit pas être confondue avec la distance en nombre de sauts qui séparent deux pairs et notée généralement k ci-après. Dans le premier cas, la distance sera explicitement qualifiée de distance entre identifiants alors que, dans le second cas, elle se dénommera distance entre deux pairs.



(a) $N = 2000$ (b) $N = 5000$

FIGURE 7.9 – Chord — Atteignabilité selon le nombre total de nœuds (S = Slapper)

7.5.1 Nombre de sauts

Les figures 7.9(a) et 7.9(b) illustrent l'atteignabilité dans Chord avec un nombre total N de 2000 ou 5000 pairs. La taille de l'espace des identifiants est donc $N_{MAX} = N \times d$ puisque les bots sont répartis aléatoirement dans ce dernier avec une distance d moyenne entre deux identifiants. Malheureusement, contrairement à la conception de Chord, N_{MAX} risque de ne pas être une puissance de deux. Cependant, il est préférable d'outrepasser cette limitation pour pouvoir comparer sur un même graphique différentes valeurs de d tout en gardant un nombre de pairs actifs N constant. En pratique, cela concorde avec un anneau Chord tronqué à la fin que les algorithmes, introduits dans la section 4.5.2, sont capables de gérer. Ainsi, le nombre d'entrées dans la table de routage est le premier entier inférieur ou égal à $log(N_{MAX})$.

La figure 7.9 montre très clairement que les performances espérées pour un botnet de type Chord sont très peu dépendantes de la distance moyenne d entre deux identifiants. A première vue, l'atteignabilité paraît meilleure pour les plus petites valeurs de d. Cependant, une analyse plus fine distingue deux classes de courbes :

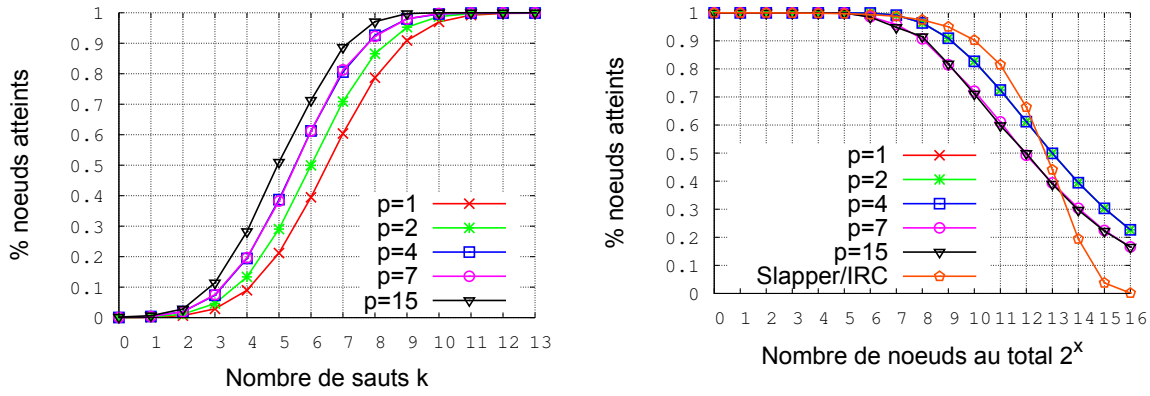
- C_0 pour d égale à une puissance de 2,
- C_1 lorsque d n'est pas une puissance de 2.

Cette caractérisation a un impact sur le nombre d'entrées différentes de la table de routage d'un pair. En effet, la formule 4.15 rappelée ci-dessous évalue ce nombre :

$$\#routes = log_2(N_{MAX}) - log_2(d) = log_2\left(\frac{N_{MAX}}{d}\right) = log_2(N)$$

Normalement, les termes $log(N_{MAX})$ et $log(d)$ sont censés être des entiers alors que ce n'est pas vérifié dans le cas des courbes C_1 pour lesquelles un terme logarithmique est remplacé par le plus grand entier inférieur à la valeur réelle. En conséquence, lorsque d est une puissance de 2 (C_0), aucune troncature n'est opérée. À contrario, la seconde classe implique une troncature et donc une réduction du nombre d'entrées dans la table de routage d'où une atteignabilité plus restreinte. Après avoir testé avec des valeurs de d de 1 à 16, l'existence de ces deux classes est bel et bien vérifiée.

Pour un réseau de taille équivalente, la première classe offre une meilleure atteignabilité indépendante de la distance d entre identifiants. Dans le second cas, l'atteignabilité est légèrement plus faible et les résultats se singularisent. Ainsi, elle est au maximum diminuée de 10 points par rapport à la courbe de la classe C_0. De plus, il est envisageable de diffuser la requête à l'ensemble du réseau (atteignabilité



(a) Espace des ids $N_{MAX} = 2^{14}$

(b) Nombre de sauts $i = 6$

FIGURE 7.10 – Chord — Atteignabilité (S = Slapper)

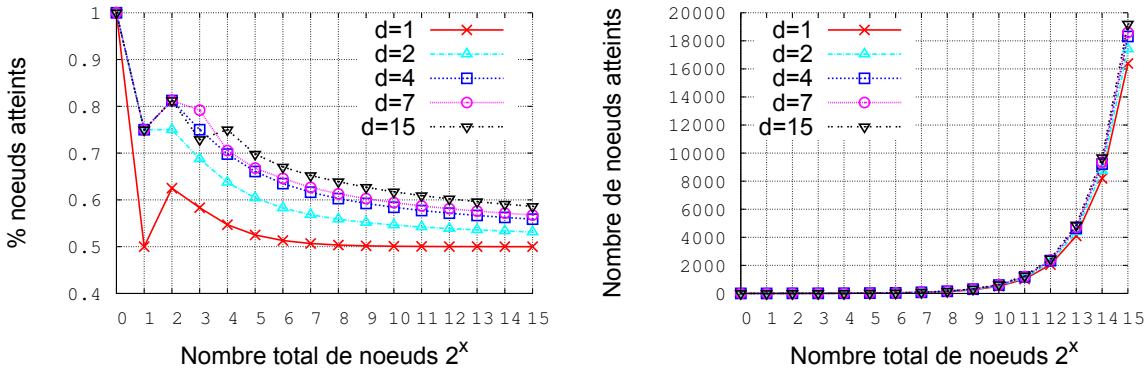
égale à 1). Cela prouve que, contrairement à IRC et Chord, l'impact de β est marginal car un attaquant désirant découvrir tout le réseau devrait, dans le meilleur des cas pour lui, être en mesure d'espionner les connexions de $\frac{N}{log(N)}$ nœuds (voir section 4.5.2).

Chord est donc la meilleure solution dans l'optique de diffuser une requête de supervision vers l'ensemble des équipements. Cette conclusion est accentuée par le parallèle avec l'atteignabilité de Slapper sur la figure 7.9. Plus précisément, le facteur de branchement de Slapper fut fixé à $log_2(N)$ pour pouvoir comparer équitablement les différents modèles. Enfin, il est préférable d'utiliser une architecture de type Slapper plutôt que Chord, plus compliquée dans sa phase de déploiement et de maintenance car, pour un nombre de sauts inférieur ou égal à 6, Slapper se caractérise par une atteignabilité équivalente voire meilleure.

7.5.2 Distance entre deux identifiants *d*

Alors que l'expérience précédente fixait le nombre de pairs actifs, celle-ci considère la taille de l'espace des identifiants comme invariable et étudie les performances selon la distance d moyenne entre deux identifiants. La figure 7.10(a) suppose $N_{MAX} = 2^{14}$ et montre clairement que l'atteignabilité est une fonction croissante de d. Effectivement, N_{MAX} étant fixé, N diminue lorsque d augmente pour respecter la contrainte $N = d \times N_{MAX}$. Hors, une propriété essentielle de Chord est son routage en $log(N_{MAX})$ indiquant que dans le pire des cas la requête devra être transmise $log(N_{MAX})$ fois. Ainsi, avec un routage tout aussi efficace et moins de nœuds à contacter, l'atteignabilité augmente en parallèle de d.

Plus précisément, une réduction de N s'accompagne forcément d'une réduction du nombre d'entrées différentes dans les tables de routage des nœuds car, comme précédemment mentionné, il y en a $log(N)$. Cependant la décroissance de la fonction logarithmique est moins élevée que celle linéaire. Ainsi, lorsque d est multipliée par k, le nombre d'entrées de la table de routage diminue de $log(\frac{N_{MAX}}{d}) - log(\frac{N_{MAX}}{d \times k}) = log(k)$. En même temps, le nombre de nœuds dans le réseau est quant à lui divisé par k. Des valeurs assez élevées de N traduisent un nombre de nœuds qui décroît plus vite que la taille des tables de routage. Finalement, même si moins de messages sont retransmis à chaque saut, moins de nœuds nécessitent d'être atteints d'où de meilleurs résultats. Par ailleurs, plus d augmente, plus le facteur de branchement diminue (nombre d'entrées différentes dans la table de routage) et $\alpha(m)$ également, accentuant le phénomène



(a) Atteignabilité moyenne

(b) Atteignabilité absolue moyenne

FIGURE 7.11 – Chord – Atteignabilité moyenne

précédent. Le même raisonnement est applicable à β mais ne présente que peu d'intérêt du fait de son faible impact dans un réseau de type Chord.

7.5.3 Nombre de nœuds

La figure 7.10(b) considère une distance de 6 sauts et l'atteignabilité est similaire pour toutes les valeurs de d (distance entre deux identifiants). En testant avec d'autres distances (nombre de sauts), les courbes demeurent similaires mais légèrement décalées vers la droite. Plus particulièrement, il a été observé que lorsque le nombre de sauts est multiplié par $\frac{5}{4}$, la courbe est légèrement déplacée à droite de 2^2 nœuds. Par exemple une atteignabilité a pour N est également observable pour $N+2^2$ lorsque le nombre de sauts est multiplié par $\frac{5}{4}$. La figure illustre une fois de plus une atteignabilité maximale proche de un. Logiquement, la distance fixée impose une atteignabilité décroissante lorsque le nombre de nœuds augmente.

Comme le facteur de branchement de Chord dépend directement de N, la courbe de Slapper correspond à l'atteignabilité maximale (meilleur cas) équivalente à celle d'un réseau IRC. La courbe est donc définie par $(1-\beta)^N$. A partir de 2^{12} équipements à superviser, Chord affiche clairement sa supériorité. Pour un réseau comportant entre 2^{10} et 2^{12} nœuds, Slapper ou IRC peuvent se révéler plus efficaces.

En fait, il est intéressant de remarquer que six sauts permettent d'atteindre la meilleure efficacité pour Slapper sur les figures 7.9(a) et 7.9(b) mais représentent également la limite à partir de laquelle l'utilisation de Chord est profitable. Ainsi même pour $N < 2^{12}$, Chord se révèle extrêmement bénéfique du fait de sa meilleure résistance aux attaques.

7.5.4 Atteignabilité moyenne

L'atteignabilité moyenne exprimée sur la figure 7.11(a) tient compte de la distance variant entre 0 (le nœud superviseur) et $log(N_{MAX})$ (distance maximale entre deux nœuds Chord). Malgré quelques variations légèrement chaotiques pour des réseaux de petites tailles, les courbes s'amoindrissent moins rapidement que l'augmentation du nombre de nœuds. De plus, plus la distance d entre deux identifiants

augmente, plus la diffusion d'une requête sera efficace car malgré un nombre de bots constants, l'espace des identifiants croît pour respecter les contraintes du modèle. De ce fait, les tables de routage des nœuds sont plus grandes et un pair peut donc avoir une connectivité plus grande. Cependant, les nœuds sont plus dispersés sur l'anneau d'où un fort niveau de redondance dans ces tables et une atteignabilité absolue analogue sur la figure 7.11(b). D'autre part, elle est toujours croissante et a une forme exponentielle contrairement à IRC car, comme précédemment montré, l'impact du facteur β est mineur avec Chord. Enfin, les variations observées sur la figure 7.11(a) sont principalement dues à des réseaux de faibles tailles ($N < 2^4$) ne méritant pas d'y porter intérêt dans le cadre de la supervision à large échelle.

7.6 Comparaison

Les études précédentes ont mis en évidence des propriétés spécifiques à chaque modèle tout en essayant de donner quelques indications à propos de leurs différences. Cette section résume ces résultats en mettant l'accent sur les singularités de chacune des architectures possibles.

7.6.1 Facteur de branchement

Contrairement à un réseau pair-à-pair, un facteur de branchement relativement élevé engendrait des conséquences néfastes directes dans le cas d'un réseau IRC. Ainsi avec $m = 10$, les performances sont plus réduites sur la figure 7.4. Ainsi, les botnets P2P sont plus résistants aux problèmes de disponibilité. Cependant, le facteur de branchement n'est pas directement paramétrable dans Chord et augmenter le facteur de branchement de Slapper à outrance peut dégrader les délais de diffusion d'une requête puisque, dans les cas extrêmes, cela revient à une architecture pseudo décentralisée telle qu'IRC.

Les études ont également montré qu'un bon facteur de branchement pour un botnet IRC était de l'ordre de 5. Pour comparer plus justement avec Slapper, le facteur de branchement de ce dernier est fixé à quatre comme expliqué dans la section 7.3. Ainsi les courbes des figures 7.4, 7.6 et 7.9 indiquent clairement qu'une architecture IRC nécessite moins de sauts pour atteindre l'atteignabilité maximale possible. En revanche, la figure 7.13 réunit les différents modèles et donne clairement l'avantage à Chord qui, malgré un nombre de sauts plus élevé pour atteindre la valeur maximale, est capable de diffuser une requête à un plus grand nombre de bots et, dans un temps plus court, à un nombre de bots équivalent par rapport aux autres modèles. En outre, Slapper semble l'architecture la moins efficace à facteur de branchement équivalent avec IRC.

7.6.2 Impact des attaques

Cette évaluation mesure la résistance des différents systèmes aux attaques. Comme expliqué auparavant, Slapper et IRC sont soumis de la même manière à la probabilité β qu'un nœud soit compromis car un seul nœud de ce type entraîne la découverte du réseau tout entier. Par conséquent, un attaquant connaît les N nœuds du réseau avec un probabilité $1-(1-\beta)^N$. En moyenne, il est donc possible de dire que sur plusieurs botnets de même taille, $detect_{slapper} = N(1-(1-\beta)^N)$ nœuds sont perçus par l'attaquant.

Dans le cas de Chord, le nombre moyen de nœuds attaqués est βN et chacun d'entre eux connaît $log_2(N)$ autres nœuds (formule (4.15)). En considérant le pire cas où la table de routage de chaque

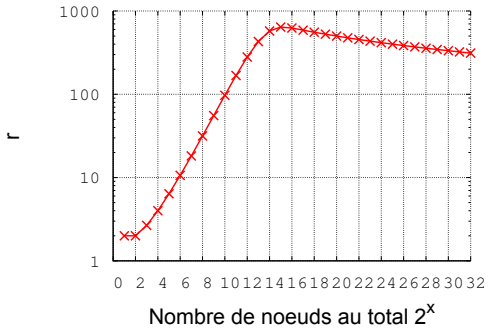


FIGURE 7.12 – Effet des attaques - $r(n)$

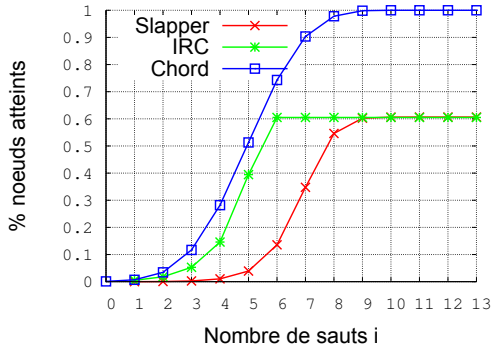


FIGURE 7.13 – Comparaison des modèles avec $N = 5000$

nœud compromis soit totalement différente, le nombre moyen de nœuds connus par un attaquant est : $detect_{chord} = \beta N \times log_2(N)$.

Pour pouvoir comparer rapidement et simplement les différents modèles, le ratio suivant est employé :

$$r(N) = \frac{detect_{slapper}}{detect_{chord}} = \frac{1-(1-\beta)^N}{\beta \times log_2(N)}$$

L'effet bénéfique de Chord est mis en exergue sur la figure 7.12 car le ratio augmente rapidement en parallèle du nombre de nœuds dans le réseau jusqu'à atteindre la valeur maximale $r(2^{15}) = 650$. Cela signifie qu'il y a 650 fois plus de nœuds connus avec IRC ou Slapper qu'avec Chord. La courbe amorce alors une légère décroissance vers sa limite en l'infini : $\lim_{N\to\infty} r(N) = 0$. Cependant pour un réseau de très grande taille telle que 2^{512} nœuds, le ratio vaut 20. De manière générale Chord est plus résistant aux attaques que Slapper ou IRC car les nœuds de Chord n'ont qu'une vue très réduite du réseau.

7.6.3 Délais

Étant donné un temps de transmission t_{trans} entre deux nœuds quelconques (serveur ou bots), un nombre de sauts permettant d'obtenir une atteignabilité similaire dans l'exemple de la figure 7.13 (5 pour IRC, 9 pour Slapper et 6 pour Chord) et toujours une moyenne de 100 bots par serveur IRC, les formules (4.10) et (4.16) estiment le délai nécessaire pour diffuser la requête aux nœuds :

- IRC : $(5+100) \times t_{trans}$
- Slapper : $9 \times m \times t_{trans}$
- Chord : $6 \times m \times t_{trans}$

Le meilleur compromis pour le facteur de branchement du modèle IRC est cinq impliquant un facteur de quatre pour Slapper. Le ratio entre les délais de ces deux architectures est donc de $\frac{105}{36}$ signifiant qu'une architecture Slapper est pratiquement trois fois plus rapide que IRC malgré un nombre de nœuds intermédiaires plus important.

Concernant Chord, le facteur de branchement dépend directement du nombre de nœuds dans le réseau et vaut $log_2(N)$ impliquant des délais plus élevés que dans Slapper car le ratio entre ces derniers est alors de $\frac{73}{36}$ dans le cas présent ($N = 5000$). Cependant les délais dans Chord restent plus courts que ceux avec une architecture IRC avec un ratio de $\frac{105}{73}$. A première vue, cette différence semble faible mais seul

Chord est capable de diffuser une requête à tous les nœuds. Par exemple IRC n'est capable d'atteindre que $0.6 \times 5000 = 3000$ bots (figure 7.13) avec un délai de $105 \times t_{trans}$ alors que Chord les atteint tous en 10 sauts soit un délai maximum de $10log_2(5000) \times t_{trans} = 122t_{trans}$. Par conséquent, Chord est, dans ce cas, légèrement plus lent car $\frac{122}{105} = 1.16$ mais parvient à transmettre la requête à 40% de bots en plus soit pratiquement le double

7.7 Conclusion

Le chapitre 4 avait, en plus de définir clairement les modèles, spécifié les objectifs à atteindre dans le cadre de la supervision à large échelle :

- passage à l'échelle : un grand nombre de machines à configurer,
- efficacité : nouvelle configuration déployée rapidement,
- anonymat : un attaquant ne doit pas pouvoir découvrir les machines supervisées ainsi que celles qui contrôlent le système pour éviter un détournement de l'usage du botnet,
- sécurité : une machine supervisée ne doit pas pouvoir lancer de fausses requêtes de supervision sur le réseau,
- garantie d'atteignabilité : l'administrateur doit pouvoir savoir quel est le potentiel résultat de l'envoi d'une requête et notamment le nombre de machines qui l'auront correctement reçu. Cela permet ensuite de prévoir des mesures supplémentaires si ce nombre n'est pas satisfaisant.

Les études réalisées précédemment vont donc aider à déterminer si les différentes architectures proposées sont des solutions viables. La garantie d'atteignabilité est clairement réalisée pour l'ensemble des modèles puisque leur objectif principal était de définir cette métrique. Ainsi, un administrateur souhaitant mettre en œuvre une telle solution peut finement évaluer les performances envisageables et ainsi paramétrer au mieux celles-ci.

La sécurité n'a pas fait l'objet des modèles car ils sont centrés principalement sur l'efficacité de la diffusion d'une requête. De plus, cette garantie de sécurité est réalisable en utilisant des techniques modernes de signatures numériques par chiffrement.

Les autres aspects sont quant à eux abordés dans le tableau 7.1. L'anonymat du contrôleur est bien préservé dans les réseaux pair-à-pair grâce aux nombreux nœuds intermédiaires. Le faible nombre de ces derniers dans IRC ainsi que la nécessité d'avoir une autorité centrale prenant en charge le déploiement des serveurs rend ce type de réseau beaucoup plus vulnérable. L'avantage indéniable des réseaux IRC est que seuls les serveurs ont besoin d'avoir un port ouvert. Les équipements à superviser ne se retrouvent pas exposés à d'autres vulnérabilités. Ce type d'architecture est donc idéal pour une entreprise. Les autres indications du tableau telles que le passage à l'échelle ou l'efficacité sont directement induites par les expérimentations de ce chapitre.

Par conséquent, un réseau IRC est une solution viable pour un réseau local d'entreprise tandis que Slapper est plutôt envisageable dans le cadre d'un réseau de partenaires sûrs. En effet, les délais sont limités mais sa faible résistance face aux attaques ne permet pas de le déployer n'importe où contrairement à Chord qui semble le mieux adapté au cas d'utilisation présenté dans le chapitre 4 où l'application visée est le déploiement d'un honeypot participatif totalement ouvert avec de très nombreuses machines.

	IRC	Slapper	Chord
Efficacité	plus petit nombre de sauts	plus courts délais	
Résistance	très contraints par une surcharge et par les attaques	très contraints par les attaques, peu de surcharge	résistance élevée, peu de surcharge, impact faible des attaques
Passage à l'échelle	$< 2^{12}$ nœuds		$\geq 2^{12}$ nœuds
Anonymat	le superviseur peut être détecté	difficile de retrouver le superviseur à cause des nœuds intermédiaires	
Intérêt	réseau large et fermé (une entreprise)	réseau large avec des participants connus et sûrs (honeypot distribué de recherche)	réseau ouvert et très large (honeypot public)

TABLE 7.1 – Comparaison des différentes solutions

8

Découverte automatique des types de messages

Sommaire

8.1 Introduction . . . 167
8.2 Protocoles testés . . . 168
8.2.1 SIP . . . 168
8.2.2 Courriels . . . 170
8.3 Évaluation de la classification . . . 171
8.3.1 Précision . . . 171
8.3.2 Nombre de types . . . 171
8.3.3 Sensibilité . . . 171
8.4 Expérimentations sur SIP . . . 172
8.4.1 Partitionnement hiérarchique . . . 172
8.4.2 Support Vector Clustering . . . 174
8.4.3 Méthode globale . . . 175
8.5 Autres protocoles . . . 176
8.5.1 Partitionnement hiérarchique . . . 176
8.5.2 Support Vector Clustering . . . 177
8.6 Paramétrage automatique de SVC . . . 177
8.7 Conclusion . . . 179

8.1 Introduction

Le chapitre 5 s'est focalisé sur la description compressée de nouvelles représentations de messages et donc plus efficace à manipuler en terme de complexité algorithmique dans l'objectif de déterminer automatiquement les types associés. De plus, la nouvelle méthode proposée ne s'appuie pas sur la grammaire du protocole et se fonde sur une technique récente de clustering. Plus exactement, un message peut être caractérisé de la façon suivante :

– la distribution de ses caractères ou octets,
– la distribution des positions pondérées des caractères ou octets,

– la distribution ordonnée des caractères ou octets robuste face à un chiffrement octet par octet grâce à un XOR.

La classification s'effectue en deux étapes :

– découverte de « méta-clusters » réunissant plusieurs types distincts,
– un partitionnement hiérarchique sur chacun de ces « méta-clusters ».

Plus précisément, la classification hiérarchique est de type ascendante en se basant sur le barycentre exprimant ainsi un groupe de messages par un seul. Ce chapitre a pour ambition de démontrer l'intérêt de cette nouvelle méthode de découverte des types de messages, étape essentielle à la rétro-ingénierie, en l'illustrant sur plusieurs protocoles. De plus, le gain de chaque étape de cette classification à double niveau sera évalué et comparé par rapport à l'application directe du partitionnement hiérarchique qui est une méthode bien connue et très utilisée dans de nombreux domaines.

Dans un premier temps, ce chapitre introduira les différents jeux de test employés incluant une brève description des protocoles associés ainsi que la définition des métriques d'évaluation avant de détailler les résultats.

8.2 Protocoles testés

8.2.1 SIP

Description

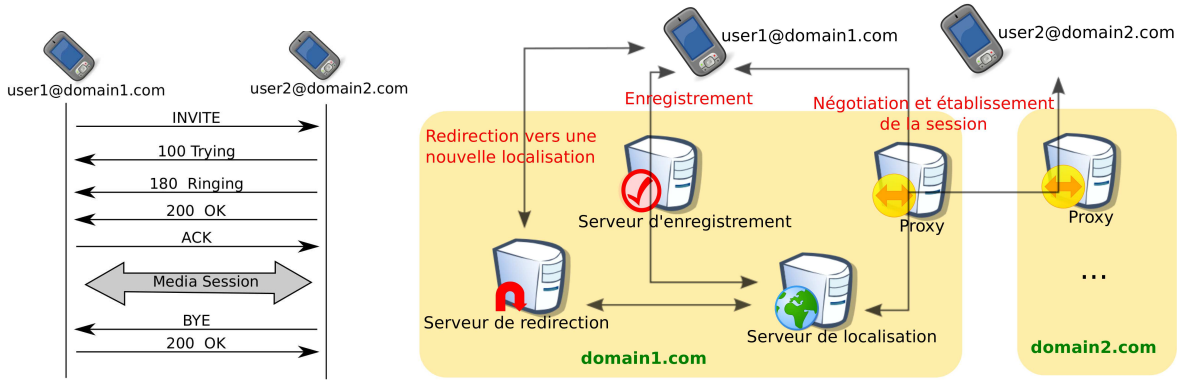
SIP [6] est un protocole de gestion de sessions multimédias indépendant de l'application sous-jacente mais démocratisé récemment par les réseaux VoIP. Il permet d'établir une session entre deux entités, d'en négocier les différents paramètres, pour parvenir à une communication acceptable par les différentes entités, et de terminer la session. SIP se charge notamment de l'authentification des participants ainsi que de leur localisation. Les analyses approfondies se consacreront essentiellement à ce protocole car il comprend de nombreuses opérations se traduisant directement en types de messages servant de base à une machine à états complexe.

Plus précisément, deux grands groupes de messages existent. Les requêtes commencent par un mot clé parmi les suivants : `REGISTER`, `OPTIONS`, `INVITE`,`UPDATE`, `CANCEL`, `ACK`, `BYE`, `SUBSCRIBE`, `NOTIFY`, `PRACK`, `PUBLISH`, `INFO`, `REFER`, `MESSAGE`. Les réponses se différencient quant à elle par un code numérique de trois chiffres dont le premier fournit une indication générale :

– 1xx : réponse informationnelle,
– 2xx : succès,
– 3xx : redirection,
– 4xx : problème client,
– 5xx : problème serveur,
– 6xx : problème général.

La fréquence d'utilisation de chacun des types de messages n'est pas similaire et permettra donc de tester si la nouvelle méthodologie proposée est apte à identifier ceux sous-représentés et pas seulement les autres. `INVITE` ou `REGISTER` sont des types fréquemment rencontrés afin de respectivement passer un appel téléphonique et s'enregistrer auprès d'un opérateur. De même, le message `200` est la réponse standard pour acquitter la requête d'un utilisateur lorsque aucun problème ne survient.

La figure 8.1(a) résume un cas simple et courant d'un dialogue SIP où *user1@domain1.com* souhaite appeler *user2@domain2.com*. Il commence alors par envoyer une requête `INVITE`. L'autre entité indique qu'elle va tenter de joindre cet utilisateur (`Trying`) puis l'avertit de l'appel entrant (`180 Ringing`).



(a) Exemple de dialogue (b) Architecture

FIGURE 8.1 – Généralités sur SIP

Finalement, l'utilisateur décroche entraînant une réponse `200 OK`. Enfin, le premier utilisateur indique de la terminaison de la transaction induite par la requête `INVITE` via l'envoi d'un `ACK`. L'appel téléphonique débute alors à proprement parler grâce à un quelconque protocole à l'instar de RTP Realtime Transport Protocol) [5]. Au final, *user2* raccroche envoyant ainsi le message `BYE` acquitté par une réponse `200 OK` une fois de plus. Bien évidemment, un dialogue réel est beaucoup plus compliqué et comprend en particuler la négociation des paramètres de la session multimédia tel que le codec à utiliser.

De plus, l'architecture SIP peut inclure un grand nombre de serveurs intermédiaires non représentés sur l'exemple. La figure 8.1(b) décrit brièvement une architecture générale de SIP. Les différents éléments sont :

- les User-Agent : équipements finaux tels que les téléphones ou les passerelles vers les réseaux de téléphonie classique,
- des serveurs :
  - les proxys transmettant les messages vers les équipements finaux,
  - les serveurs d'enregistrement (registrar) sur lesquels les utilisateurs se connectent pour s'identifier,
  - les serveurs de localisation (location server) indiquant où se trouve un utilisateur,
  - les serveurs de redirection (redirect server) enregistrant les demandes des clients pour faire suivre les messages (mobilité) et collaborant donc avec les serveurs de localisation.

Jeu de données

Dans le cadre des expérimentations, des traces SIP ont été générées à partir de 27 équipements différents de fabricants variés incluant tout aussi bien des terminaux dédiés (téléphones ou hardphones) que des logiciels de téléphonie (softphones). Ainsi, la tâche de découverte des types de messages est encore plus ardue car les implantations d'un protocole peuvent légèrement varier malgré une spécification commune. De plus, la figure 8.2 montre la distribution des 27 types différents que comporte le jeu de données qui n'est pas uniforme, reflétant bien un réseau réel. En effet, équilibrer arbitrairement les classes n'aurait pas été réaliste et cette distribution amplifie la complexité de la classification car il y a beaucoup de clusters de tailles très variées. Cependant, pour éviter qu'elle ne biaise les résultats, des métriques d'évaluation adéquates sont nécessaires et seront détaillées dans la section 8.3.

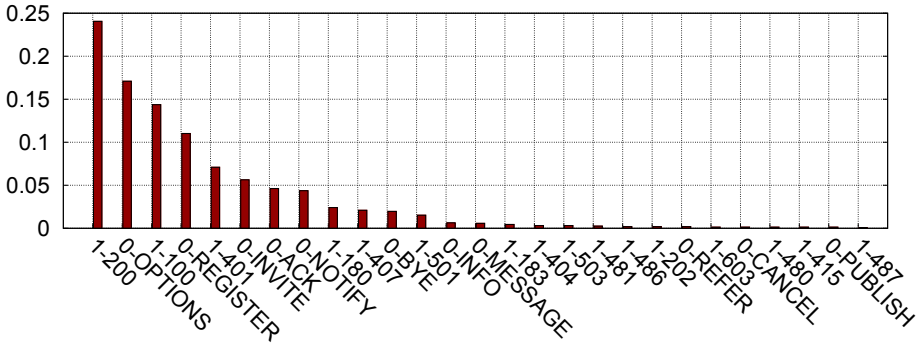
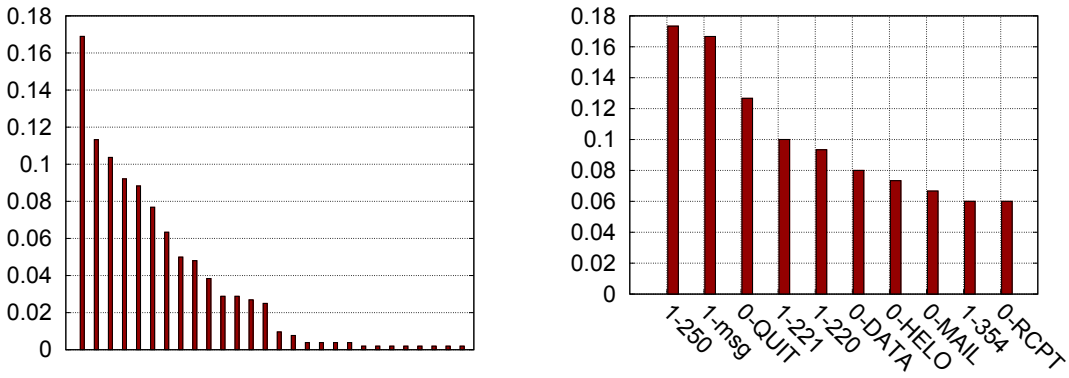


FIGURE 8.2 – Distribution empirique des message SIP (le préfixe 0 indique une requête tandis que 1 préfixe les réponses)



(a) IMAP (b) SMTP

FIGURE 8.3 – Distributions empiriques des autres protocoles

8.2.2 Courriels

Les deux autres protocoles sur lesquels la nouvelle technique de classification des types sera testée font partie du domaine de la gestion des courriels. Ils sont complémentaires entre eux puisque IMAP [4] est dédié à la réception et SMTP [1] à l'envoi.

Plus précisément, IMAP est un protocole supportant la consultation d'une boîte de courriels directement sur le serveur sans les télécharger. Le jeu de données est constitué de 521 paquets de 28 types différents comme le montre la figure 8.3(a) où la distribution est encore plus déséquilibrée que pour SIP.

SMTP est le protocole standard pour envoyer des courriels. Le jeu de données en question contient seulement dix types pour un total de 150 messages. Leur distribution beaucoup plus équilibrée (figure 8.3(b)) testera donc la méthodologie dans de nombreux cas bien différents augmentant ainsi la validité des résultats obtenus.

8.3 Évaluation de la classification

La qualité de la classification des messages est mesurée selon plusieurs critères prenant en compte la performance globale ainsi que celle par type. Les notations utilisées sont les suivantes :

- n : nombre total de messages,
- l'ensemble $M = \{m_1, \ldots, m_n\}$ de messages,
- r différents types (classes, clusters) découverts tel que $r \leq n$,
- l'ensemble $C = \{c_1, \ldots, c_n\}$ des clusters de messages,
- $c(m_i)$ le cluster ou type assigné à m_i,
- $t(x)$ où $x \in C \cup M$ représente le type réel d'un message ou d'un cluster,

Dans l'intention de pouvoir évaluer le type réel d'un message, il est obligatoire d'extraire ces types en utilisant la syntaxe du protocole. Comme expliqué dans le chapitre 5, la classification n'associe pas un type réel aux clusters puisqu'aucune grammaire n'est disponible et qu'elle vise seulement à regrouper les messages de même type ensemble. Dans le cadre de l'évaluation, un type réel leur est associé à partir des messages qu'il regroupe. En effet, le cluster est étiqueté par le type majoritaire qu'il contient. Cependant, si plusieurs clusters se révèlent être du même type, seul celui incluant le plus grand nombre de messages correspondant est conservé. Les autres sont alors ignorés et réunis dans un cluster « corbeille » dont tous les messages sont considérés comme mal identifiés. Cela renforcera encore les résultats car un type sur-représenté dans plusieurs clusters peut occulter les autres.

8.3.1 Précision

La proportion de messages assignés à leur type réel se définit comme la précision :

$$prec = \frac{\sum_{m_i \in M | t(m_i) = t(c(m_i))} 1}{n} \tag{8.1}$$

8.3.2 Nombre de types

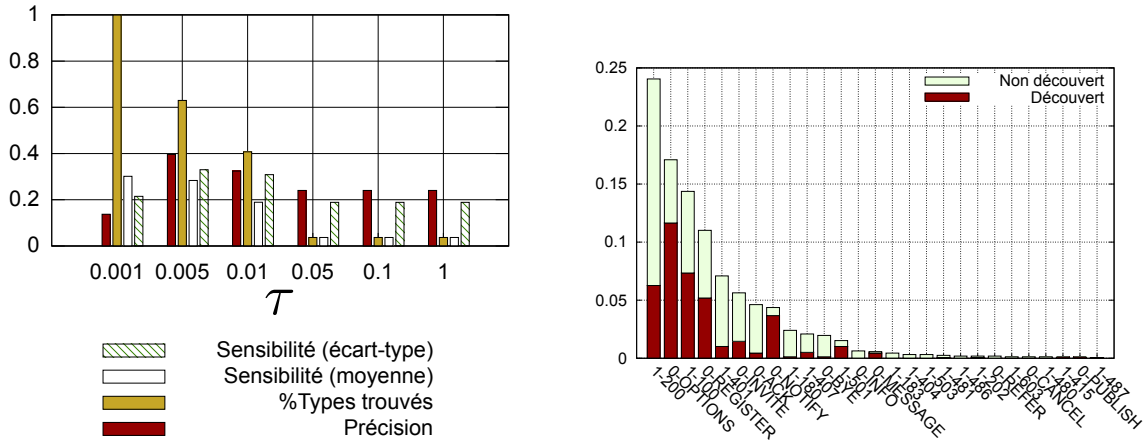
Cette seconde métrique mesure le nombre de classes trouvées par rapport au nombre réel :

$$\%types = \frac{r}{k} \tag{8.2}$$

8.3.3 Sensibilité

Alors qu'obtenir une bonne précision est aisé en se concentrant uniquement sur les types les plus courants et en ignorant les autres tel que les types `603`, `480`, `486`, `487` de la figure 8.2, la sensibilité mesure la précision individuelle de chaque classe, c'est-à-dire la proportion de messages d'un certain type correctement identifiés. Cette métrique peut aussi se trouver sous l'appellation « rappel » dans la littérature. Étant donnée une classe c, elle se définit comme :

$$sens(y) = \frac{\sum_{m_i \in M | t(m_i) = y} x_i}{\sum_{m_i \in M | t(m_i) = y} 1} \; avec \; x_i = 1 \; si \; t(m_i) = t(c(m_i)) \; sinon \; 0 \tag{8.3}$$



(a) Évaluation de la classification

(b) Identification des types avec $\tau = 0.005$

FIGURE 8.4 – Clustering hiérarchique – Distribution ordonnée des caractères

En pratique, mener une analyse approfondie sur la sensibilité de plusieurs dizaines de types est laborieux sans pour autant donner une indication claire sur les performances. Ainsi, il est préférable de calculer la moyenne et l'écart type sur toutes les classes et d'y adjoindre la composition exacte des classes identifiées si nécessaire. Par exemple, la figure 8.4(b) indique pour chaque type, la proportion de messages reconnus et ceux non identifiés tout en rappelant la proportion de ce type (hauteur des histogrammes) dans le jeu de données.

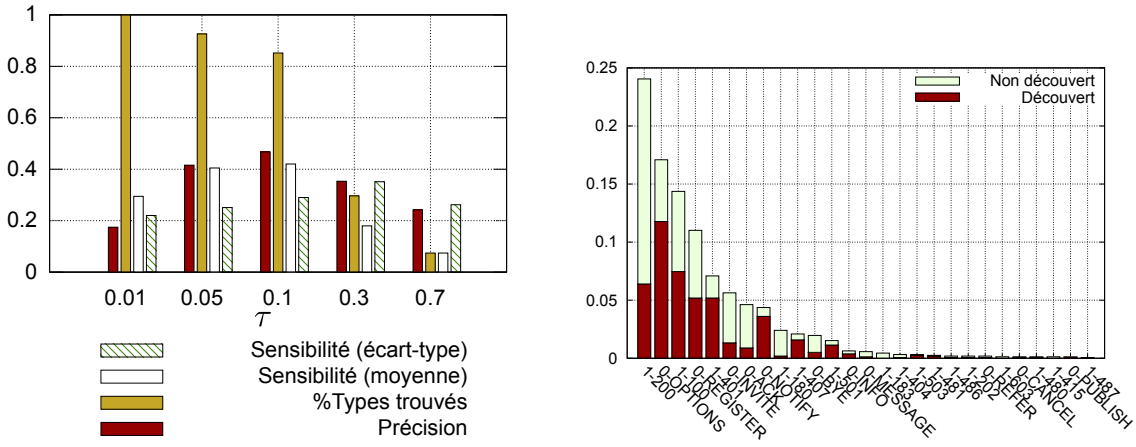
8.4 Expérimentations sur SIP

8.4.1 Partitionnement hiérarchique

Distribution ordonnée des caractères

La première expérience tente de classer les messages en utilisant la méthode du partitionnement hiérarchique grâce à la distribution ordonnée des caractères dont le principal intérêt est sa résistance face à un chiffrement simple avec XOR. Cet algorithme se base sur une distance maximale τ autorisée entre deux clusters à regrouper. La figure 8.4(a) présente les résultats de la classification pour plusieurs valeurs de τ. Malgré l'avantage évoqué précédemment, cette métrique n'est pas très performante. Le meilleur compromis atteignable ($\tau = 0.005$) n'est capable de distinguer que 60% des types et de correctement assigner 40% des messages. Cela implique donc une sensibilité égale à 0% pour les 40% de types ignorés d'où une sensibilité moyenne basse mais surtout un écart type très élevé lui même supérieur à la moyenne.

La composition des clusters correspondants est donné sur la figure 8.4(b) révélant clairement l'inutilité de la distribution ordonnée des caractères des messages car aucun type ne présente réellement de bons résultats, c'est-à-dire qu'une proportion de messages de chaque type subsiste non correctement assignée excepté pour `0-NOTIFY` qui n'est malheureusement pas le type le plus significatif. Par conséquent, tenter d'appliquer une seconde classification sur ces premiers clusters n'améliorera pas les performances. Effectivement, ce type de technique en deux passes cherche, dans un premier temps, à différencier des méta-clusters de plusieurs types mais néanmoins consistants afin de les éclater en plusieurs pendant la seconde passe.



(a) Évaluation de la classification

(b) Identification des types avec $\tau = 0.1$

FIGURE 8.5 – Clustering hiérarchique – Distribution des caractères

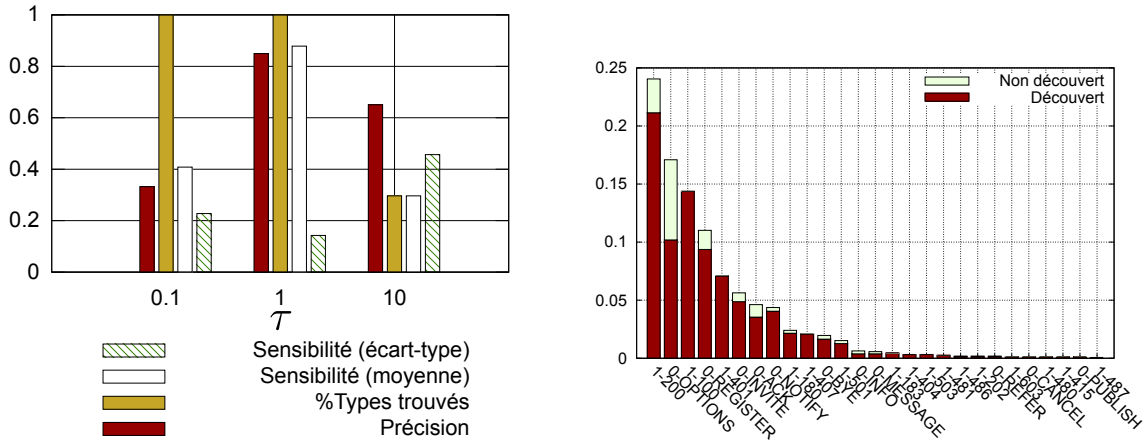
Enfin, cette première classification illustre le comportement du partitionnement hiérarchique lorsque son paramètre principal, τ, varie et dont l'impact sur les résultats est considérable comme le montre la figure 8.4(a). Ainsi, lorsque τ augmente, des clusters plus éloignés peuvent être groupés impliquant une augmentation de la largeur des clusters obtenus et donc une diminution de leur nombre. A contrario, l'évolution de la précision et de la sensibilité selon le paramètre τ n'est pas aussi évidente. Par exemple, au fur et à mesure de l'augmentation de τ, la précision commence par croître avant d'être dégradée. En effet, τ petit engendre la création de multiples micro-clusters éparpillant ainsi les messages de même type au sein d'entre eux mais aidant alors à découvrir l'ensemble des types réels. Cependant, seul un cluster par type est considéré comme juste et les autres ignorés d'où des résultats médiocres. Lorsque τ croît, ces derniers sont rassemblés d'où une amélioration de la précision. À partir d'un certain moment, les clusters contenant des messages différents sont alors fusionnés d'où de plus mauvais résultats.

Distribution des caractères

Au vu des faibles résultats précédents, la contrainte souple de la résistance face à un chiffrage est levée permettant ainsi l'usage de la métrique de distribution des caractères. En conséquence, la comparaison de deux messages se base sur la proportion même de chaque caractère précis alors que la distribution ordonnée ignorait cette information et comparait les proportions de deux caratères distincts. L'ajout d'information dans la représentation des messages devrait donc logiquement permettre d'atteindre de meilleures performances. Malheureusement, la précision représentant le nombre de messages correctement discernés demeure faible sur la figure 8.5(a) puisqu'elle vaut au maximum 46%. Cependant, la différence majeure réside dans sa faculté à trouver un plus grand nombre de types (90% lorsque $\tau = 0.05$), tout en conservant une précision similaire à celle obtenue dans le meilleur des cas avec la distribution ordonnée de la figure 8.4(a) qui ne distinguait que 60% des types. Par conséquent, la sensibilité moyenne est plus forte. Malheureusement, la composition des clusters pour un bon compromis ($\tau = 0.1$) ne met toujours pas en évidence des propriétés qui laisseraient envisager une deuxième passe de classification.

Distribution des positions pondérées des caractères

Pour rappel, la pondération des caractères tient compte du fait que les premiers octets d'un message spécifient généralement son type. SIP présente aussi ce modèle de conception. Par exemple, le message



(a) Évaluation de la classification

(b) Identification des types avec $\tau = 1$

FIGURE 8.6 – Clustering hiérarchique – Distribution des positions pondérées des caractères

`INVITE` contient deux «I» dans ces 7 premiers octets alors qu'un message «MESSAGE» n'en contient aucun. Cependant, les messages SIP contiennent la définition de nombreux champs comme des paramètres ou des identifiants uniformes de ressources (URI). Ces champs sont généralement plus longs que le mot-clé définissant le type et peuvent contenir une multitude de lettres «I» également d'où une préférence légitime pour l'utilisation de positions pondérées accordant plus d'importance aux premiers octets que l'usage des positions non pondérées.

Les résultats détaillés dans la figure 8.6(a) sont très encourageants car tous les types sont discernés tout en étiquetant correctement 85% des messages dans le meilleur des cas. La sensibilité moyenne est aussi clairement améliorée prouvant donc que ces bons résultats ne sont pas uniquement dûs au bon classement des messages les plus représentatifs. Cette dernière conclusion se voit renforcée par le faible écart type associé ainsi que la figure 8.6(b) révélant qu'il semble que ce soient les types les plus significatifs qui contiennent une part relativement importante de messages non identifiés.

8.4.2 Support Vector Clustering

La méthode SVC ou Suport Vector Clustering est paramétrée par deux paramètres principaux : le paramètre gaussien q amplifiant plus ou moins les distances entre les messages lorsqu'ils sont projetés dans l'espace de re-description et le paramètre C permettant de pénaliser plus ou moins de messages restant à l'extérieur de l'hypersphère pendant la phase d'apprentissage (voir la section A.3 pour de plus amples détails).

La distribution des positions se révélant être nettement plus profitable que les autres métriques, elle fera l'objet essentiel des tests suivants. Cependant, la distribution ordonnée des caractères est employée une fois de plus avec SVC car sa résistance face au chiffrement par XOR implique d'y porter également attention. La figure 8.7 présente des résultats généraux lors de l'application de SVC avec une telle métrique. La méthode fut testée avec de nombreuses valeurs de q et C (1 couple par valeur de l'axe des abscisses) mais ne nécessite pas d'y porter attention ici car les performances sont très mauvaises quelque soit la configuration. La précision est proche de 20% généralement avec peu de clusters trouvés et représente principalement le fait que seule la classe `1-200` soit trouvée. La découverte efficace des

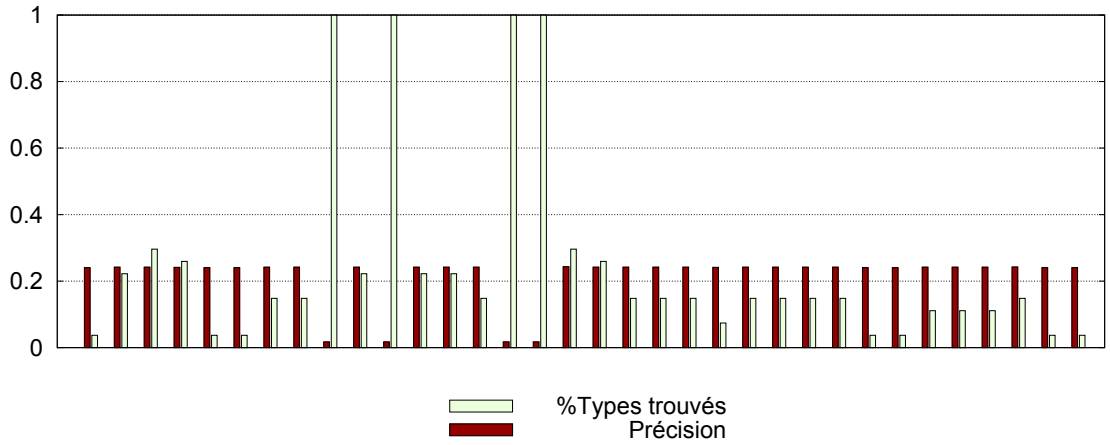


FIGURE 8.7 – SVC – Distribution ordonnée des caractères

différents types de messages se fait au détriment de la précision comme le montrent les 4 histogrammes se démarquant clairement sur le graphe. Par conséquent, la distribution ordonnée des caractères n'est pas envisageable avec SVC.

La mise en œuvre de la distribution des positions pondérées se révèle quant à elle incontestablement plus efficace comme l'illustre la figure 8.8. Dans le meilleur des cas, la précision est de 73% tout en parvenant à mettre en exergue l'ensemble des types de messages. Comme pour le partitionnement hiérarchique, la sensibilité moyenne élevée indique une classification identifiant correctement la plupart des types. Cependant une analyse détaillée des sensibilités par type montre que certains d'entre eux sont très difficilement discernés. Ainsi, les résultats obtenus sont légèrement inférieurs à la technique du partitionnement hiérarchique (précision = 85%).

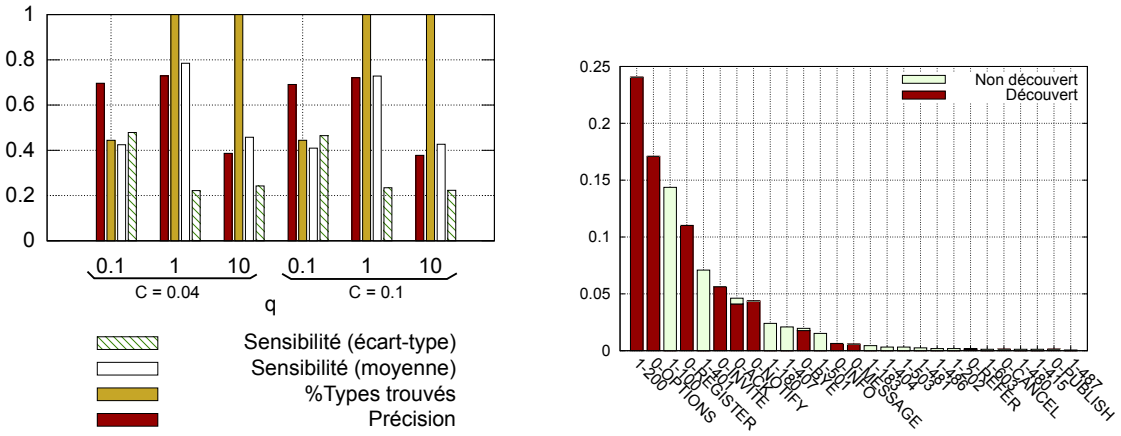
Concernant l'impact des différents paramètres de SVC, C a un impact très faible sur la classification comme le montre la figure 8.8(a). En réalisant d'autres tests plus approfondis, C n'a pas d'effet significatif tant que l'on reste dans une valeur adéquate à SVC introduite par la formule (5.14) du chapitre 5 :

$$\beta_init_i < C < 1$$

La seule contrainte supplémentaire pour obtenir une classification équivalente est de ne pas choisir des valeurs trop extrêmes, c'est-à-dire proche de un. q accroît les différences entre les messages d'où un effet considérable sur la classification. Effectivement, lorsque q augmente, les différences entre les messages aussi, ceux-ci sont de plus en plus séparés permettant ainsi de passer d'un grand cluster trop global à des clusters plus spécifique et donc une meilleure précision. A contrario, quand q augmente exagérément, les clusters sont encore éclatés dispersant les messages de même type et réduisant ainsi la précision de la classification. Le nombre de types découverts est quant à lui croissant car une augmentation du nombre de clusters s'accompagne logiquement de la découverte d'un plus grand nombre de types.

8.4.3 Méthode globale

Alors que le meilleur compromis avec SVC concordait avec $q = 1$, l'identification spécifique de chaque type dans le cas $q = 0.1$ se révèle intéressante sur la figure 8.8(b). En effet, contrairement au cas précédent, seuls quelques types sont réellement distingués mais de façon efficace. A contrario, les types



(a) Évaluation de la classification

(b) Identification des types avec $q = 0.1$

FIGURE 8.8 – SVC – Distribution des positions pondérées des caractères

médiocrement découverts sont associés à une sensibilité extrêmement faible voire nulle d'où un écart type élevé sur la figure 8.8(a) et ce malgré une précision presque maximale (70%). Cette observation indique donc que des méta-clusters regroupant plusieurs types ont été créés et appliquer une seconde passe de classification tel qu'un partitionnement hiérarchique est cette fois-ci justifié.

Par conséquent, les clusters construits précédemment sont alors découpés chacun de leur côté. Suite à plusieurs tests, le partitionnement paramétré par $\tau = 1.1$ s'est révélé extraordinairement performant puisque la précision devient alors égale à 91% avec 96% des types découverts.

8.5 Autres protocoles

Suite aux nombreuses expérimentations sur le protocole SIP, la nouvelle méthode de rétro-ingénierie de découverte automatique des types de messages fut testée avec les protocole IMAP et SMTP. Dans l'optique de pouvoir rapidement tester ces derniers, seules les positions pondérées des caractères sont considérées tout en y appliquant une normalisation classique en soustrayant la moyenne et en divisant par l'écart type. Ainsi, les paramètres utilisés seront identiques quelque soit le protocole.

8.5.1 Partitionnement hiérarchique

En premier lieu, un partitionnement hiérarchique est mis en œuvre occasionnant les résultats de la figure 8.9. La précision est similaire pour les deux protocoles et moins de 50% des messages sont correctement identifiés. Grâce à la normalisation des données, la valeur de τ donnant le meilleur compromis est identique quelque soit le protocole, c'est-à-dire $\tau = 20$. Toutefois, les résultats restent très modestes.

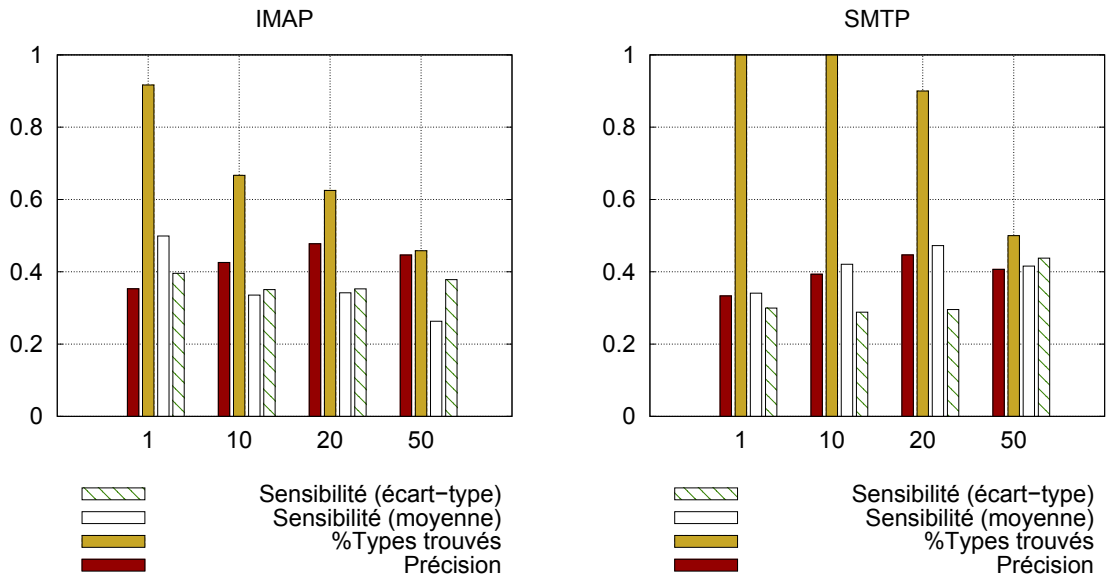


FIGURE 8.9 – Partitionnement hiérarchique – autres protocoles (τ en abscisse)

8.5.2 Support Vector Clustering

Les résultats obtenus par l'application de SVC sur le jeu de données IMAP sont proches de ceux obtenus avec le partitionnement hiérarchique. En effet, seuls 36% des messages sont affectés au bon cluster soulignant l'importance de la classification à deux niveaux pour laquelle ce chiffre s'élève à 49%. Ce résultat est légèrement supérieur au partitionnement hiérarchique mais son principal avantage est le nombre de types trouvés passant de 62 à 96% avec cette nouvelle méthode. En clair, la nouvelle méthode proposée est au moins aussi précise que le partitionnement hiérarchique tout en étant capable de découvrir la majorité des types réels.

SVC améliore clairement les performances sur le protocole SMTP car autant de types sont découverts que pour le partitionnement hiérarchique mais la précision est multipliée par 1,6 passant ainsi de 45 à 72%. En exécutant la second passe, la précision grimpe jusqu'à 74%. L'écart type reste relativement élevé car un type est totalement ignoré.

8.6 Paramétrage automatique de SVC

En plus d'adapter une nouvelle technique de classification, le chapitre 5 suggérait un paramétrage automatique de SVC. Concernant le paramètre C, les expérimentations précédentes ont confirmé son faible impact et il peut être fixé de la manière suivante :

$$\beta_init_i = \frac{1}{N} \lessapprox C$$

.

Étant donné le paramètre q, la métrique $evol_c(q)$ mesure l'évolution du nombre de clusters entre deux valeurs testées de q alors que $evol_w(q)$ est l'évolution de la plus grande largeur d'un cluster, elle-même définie comme la plus grande distance entre deux messages d'un même cluster. Cette distance

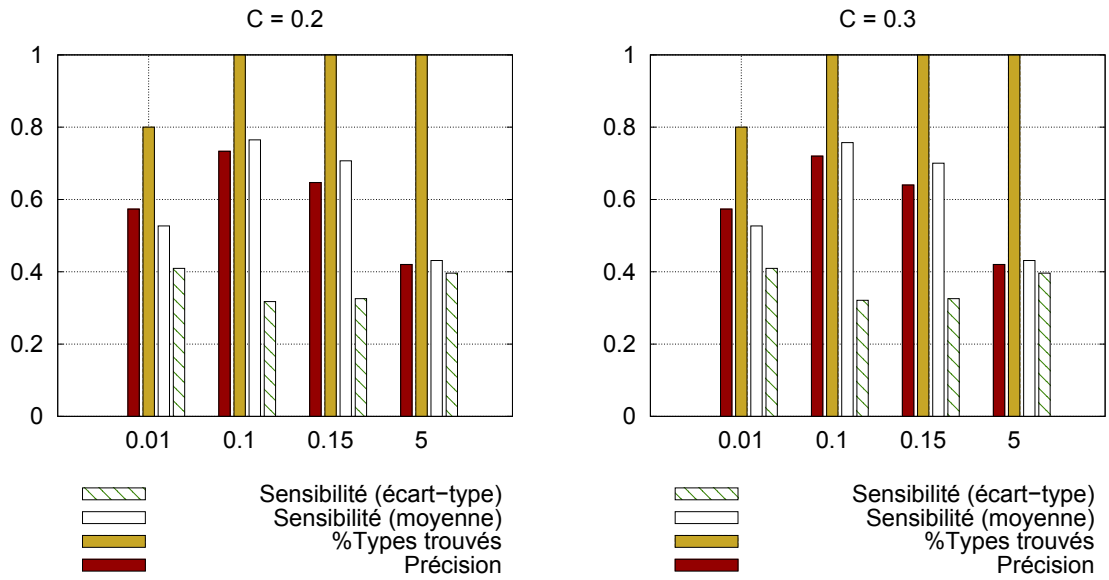


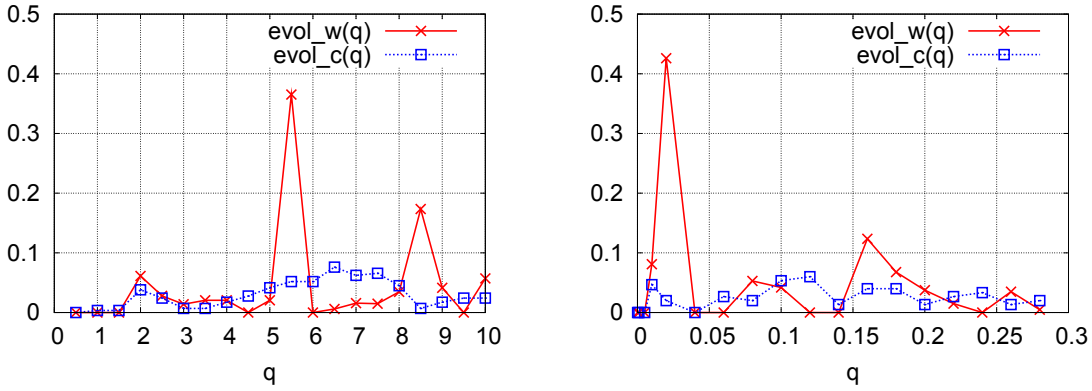
FIGURE 8.10 – SVC - SMTP dataset

doit bien évidemment être calculée de la même manière que pendant le clustering. Au vue des résultats précédents, l'utilisation des positions pondérées est conseillée.

Le rôle de la première passe de SVC est donc de créer des grands clusters pouvant regrouper plusieurs types sans toutefois être mélangés parmi plusieurs. Ainsi une augmentation de q doit impliquer une observation significative du nombre de cluster ainsi qu'une réduction de la largeur maximale indiquant que l'augmentation du nombre de clusters est bien dûe à une division franche d'un ou plusieurs gros clusters. $evol_w(q)$ est en fait l'évolution opposée de la largeur maximale permettant de tracer les deux métriques sur un même graphe à l'instar de la figure 8.11.

Le paramétrage d'IMAP révèle plusieurs pics pour $evol_w(q)$ dont le dernier pour la valeur $q = 8.5$ est facilement exclu car le nombre de clusters construits croît moins rapidement comme le démontre une valeur pratiquement nulle pour $evol_c(q)$. Le pic en $q = 5.5$ correspond à la valeur sélectionnée manuellement dans les tests précédents. Cependant, le premier pic demeure le plus intéressant car il représente une augmentation majeure des deux métriques. En appliquant de nouveau la technique de classification avec ce paramètre, les résultats sont légèrement meilleurs avec une précision de 50%. Ainsi, le bon paramètre semble être celui coïncidant avec une augmentation brutale des deux métriques. La figure 8.11(b) expose plusieurs pics abrupts mais aucun d'entre eux ne sont vraiment simultanés. Il est donc nécessaire d'introduire une certaine marge de liberté dans la simultanéité des deux métriques tout en considérant que le pic le plus élevé des deux spécifiera la valeur à choisir. Par conséquent, le premier pic clairement identifiable induit $q = 0.02$ qui, malgré sa différence par rapport à la valeur sélectionnée manuellement, permet d'augmenter la précision d'un point.

Concernant SIP, les plages de valeurs de $evol_w(q)$ et $evol_c(q)$ sont trop différentes et se retrouvent ainsi tracées sur deux graphiques différents sur la figure 8.12. L'évolution de la largeur du plus grand cluster ne dévoile qu'une seule augmentation brutale pour laquelle la courbe du nombre de clusters présente également un pic. Cette valeur sélectionnée automatiquement est $q = 0.05$ et, bien que différente de la valeur manuellement choisie, permet d'obtenir les mêmes performances.



(a) IMAP (b) SMTP

FIGURE 8.11 – SVC – Paramétrage automatique de q

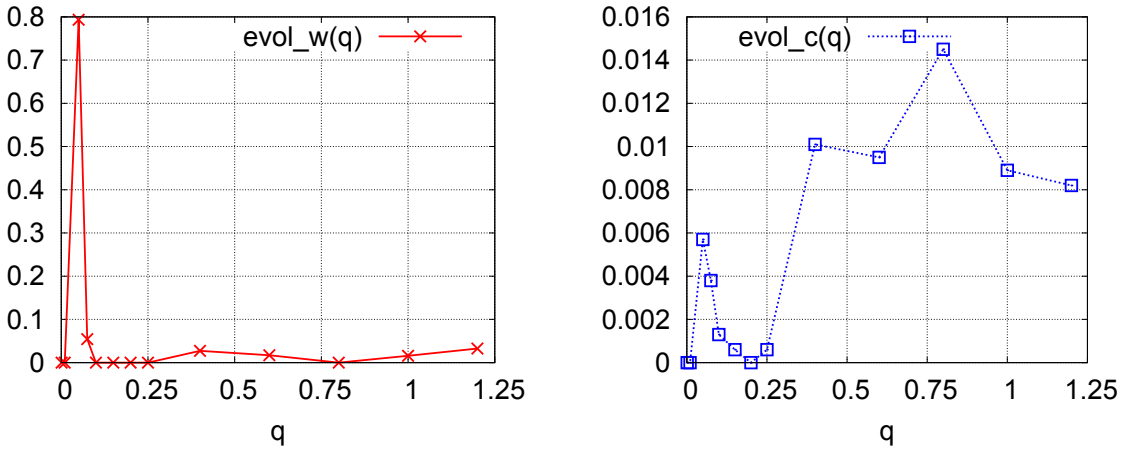


FIGURE 8.12 – tuningSIP

8.7 Conclusion

Les tests menés au cours de ce chapitre ont mis en évidence de bonnes performances pour la classification à deux niveaux proposée dans le chapitre 5 se basant sur SVC et une nouvelle représentation des messages. Effectivement, la distribution des positions pondérées des caractères s'illustre comme un critère déterminant dans la différenciation des types de messages. Cette étape est essentielle dans le cadre de la rétro-ingénierie car elle constitue la première étape pour appréhender le fonctionnement d'un protocole inconnu pour notamment en construire sa machine à états. Par exemple, il devient possible de déterminer le fonctionnement protocolaire d'un malware voire de cibler plus précisément les requêtes à injecter pour le rendre hors de nuire.

Enfin, la comparaison avec une méthode plus classique de partitionnement hiérarchique a mis en évidence l'intérêt de cette nouvelle méthode impliquant une précision au moins tout aussi bonne mais aussi une découverte plus efficace des types c'est-à-dire qu'elle ne se focalise pas uniquement sur les types les plus représentés dans le jeu de données.

En plus de la vérification de l'application de cette méthode, le chapitre 5 avait clairement mis en évidence son intérêt en terme de complexité. Cette méthode est performante, utilise une représentation simplifiée et condensée des messages et repose uniquement sur l'observation de traces réseaux.

Enfin, à l'instar des méthodes pour paramétrer correctement la classification hiérarchique automatiquement [11], établir un procédé visant cet objectif dans le cadre de SVC s'est avéré essentiel. Ainsi, en plus de contribuer au domaine de la rétro-ingénierie, ce nouveau procédé s'étend au domaine de la classification en générale et peut ainsi aider à démocratiser l'utilisation de SVC.

9

Identification des équipements

Sommaire

9.1 Introduction 181
9.2 Métriques d'évaluation 182
9.3 Fingerprinting comportemental 183
9.3.1 Jeu de données 183
9.3.2 Paramétrage 184
9.3.3 Réseau opérateur 188
9.4 Fingerprinting syntaxique 188
9.4.1 Jeux de données 190
9.4.2 Fingerprinting supervisé 191
9.4.3 Fingerprinting non supervisé 192
9.4.4 Opérateur 194
9.5 Conclusion 195

9.1 Introduction

Ce chapitre évalue la viabilité des différentes techniques de fingerprinting élaborées dans le chapitre 6 visant à déterminer le type d'un équipement pour un protocole donné. Dans cette étude, ces méthodes seront illustrées grâce au protocole SIP introduit dans la section 8.4. Pour rappel, ces méthodes reposent sur des descriptions sémantiquement riches d'un ou plusieurs messages dans l'optique d'être difficilement contrefaites.

Ainsi, la première représentation est dite comportementale car elle regroupe un ensemble de messages au sein d'un arbre TR-FSM illustrant les séquences possibles de messages et les délais entre eux. Cette caractérisation peut alors être testée avec la méthode des machines à vecteurs de support grâce à une fonction noyau spécialement définie. L'autre représentation possible s'appuie sur la structure hiérarchique et syntaxique des champs composants un message car une telle structure peut difficilement être contrefaite. Les machines à vecteurs de support seront également employées mais aussi complétées par un nouvel algorithme de classification totalement non supervisé, c'est-à-dire ne nécessitant pas de phase d'apprentissage.

Ce chapitre va donc montrer les performances des différentes méthodes ainsi que l'impact de leurs paramètres pour en faciliter une utilisation future. Cependant, les résultats ne sont pas directement comparables car chaque méthode ne s'applique pas au même cas selon l'information disponible. Ainsi, le fingerprinting comportemental est applicable en ne connaissant que les types de messages, tâche réalisable grâce à la méthode proposée et évaluée dans les chapitres 5 et 8, alors que le fingerprinting syntaxique demande la connaissance complète de la grammaire des messages du protocole. Parallèlement, un jeu d'apprentissage correct doit être fourni pour les techniques supervisées.

9.2 Métriques d'évaluation

La qualité de la classification se basera sur certaines métriques habituellement utilisées [21] dont une partie a déjà été décrite au chapitre précédent. Considérant x_d le nombre d'arbres associés à un équipement de type d, y_d le nombre d'arbres identifiés comme de type d et $z_{d_2 d_1}$ le nombre d'arbres de type d_2 classés comme d_1, N le nombre total d'arbres, la précision est la proportion générale d'équipements correctement identifiés :

$$acc = \sum_{d \in D} z_{dd} / N \tag{9.1}$$

La sensibilité est le nombre d'équipements d'un type d identifiés correctement :

$$sens(d) = z_{dd} / x_d \tag{9.2}$$

Le ratio suivant est le nombre r de différents types découverts par le fingerprinting non supervisé en comparaison du nombre k de types distincts réels :

$$\%types = \frac{r}{k} \tag{9.3}$$

De plus, la spécificité mesure la consistance d'un cluster en calculant le nombre d'arbres qu'il contient, correspondant au type du cluster :

$$spec(d) = z_{dd} / y_d \tag{9.4}$$

Les métriques individuelles ne donnent malheureusement pas un aperçu global des performances d'où l'introduction de valeurs moyennes :

$$\begin{aligned} avg_sens &= \sum_{d_i \in D} sens(d_i) / N \\ avg_spec &= \sum_{d_i \in D} spec(d_i) / N \end{aligned} \tag{9.5}$$

Alors que la précision donne un rapide aperçu des performances, elle peut facilement être biaisée. En effet, si une classe représente 80% des données à classer, il suffit de ranger toutes les données au sein de cette classe afin d'obtenir une précision de même envergure. En conséquence, la précision doit être modérée selon une métrique tenant compte de la répartition des données dans les classes à l'instar du coefficient d'information mutuelle :

$$IC = \frac{H(\mathbf{X}) + H(\mathbf{Y}) - H(\mathbf{Z})}{H(\mathbf{X})} \tag{9.6}$$

où H est l'entropie appliquée sur les distributions suivantes :

- $\mathbf{X} = x_i/N$
- $\mathbf{Y} = y_i/N$
- $\mathbf{Z} = z_{ij}/N$

Ce coefficient varie entre zéro et un (classification parfaite) et reflète à la fois la sensibilité et la spécificité. Dans le cas du petit exemple précédent avec une précision égale à 80%, *IC* vaut 0, synonyme d'une mauvaise classification.

Comme dans le chapitre précédent, la classification non supervisée ne peut en aucun cas associer un type réel aux clusters car ce n'est pas son objectif. L'évaluation nécessite néanmoins d'assigner un type à chaque cluster selon le procédé suivant :

1. chaque cluster se voit attribuer le type correspondant à la majorité des arbres qu'il contient,
2. pour chaque type réel différent, seul le cluster étiqueté par ce type et contenant le plus d'arbres correspondants est conservé.

La dernière étape construit donc un cluster collectant les données de tous ceux qui ont été éliminés. L'évaluation du coefficient d'information n'a alors pas de sens d'où l'utilisation du F-Score dans le cas non supervisé :

$$F - score = \frac{2 \times avg_sens \times avg_spec}{avg_spec + avg_sens} \tag{9.7}$$

$F - score$ combine la sensibilité et la spécificité mais ne reflète pas la distribution des messages dans le jeu de données. Toutefois, une découverte des types les plus représentés seulement induit un score médiocre.

9.3 Fingerprinting comportemental

L'implémentation du fingerprinting comportemental est réalisée en python interfacé avec LIBSVM [1] et KiF [34].

9.3.1 Jeu de données

Plusieurs jeux de données distincts serviront à l'évaluation du fingerprinting comportemental. Le premier fut généré via une plate-forme d'essai locale comprenant aussi bien des téléphones physiques de différentes marques (Cisco, Linksys, Snom, Thomson) que des logiciels de téléphonie comme `Twinkle` ou `Ekiga`. De plus, des serveurs sont également déployés comme `Asterisk`, `Cisco Call Manager`, `Openser`. Pour rendre la tâche encore plus difficile, un type d'équipement (pile protocolaire) peut être représenté par plusieurs versions différentes. Ce premier jeu de donnée sera dénoté *jeu d'essai* par la suite alors que les autres seront nommés *jeu opérateur* car ils proviennent d'un opérateur VoIP réel. Plus précisément, ils en existent quatre (de T1 à T4), chacun représentant environ 45Mo de traces réseaux. L'environnement est différent de la plate-forme d'essai car il est composé en majorité d'équipements physiques connectés via Internet d'où des délais plus élevés (tableau 9.1) et beaucoup plus bruités permettant ainsi d'estimer la robustesse du nouveau système de fingerprinting.

Le tableau 9.1 donne un aperçu global des caractéristiques des différents jeux de données en détaillant le nombre de messages SIP, le nombre de types d'équipements mais aussi le nombre d'appels tentés équivalent au nombre de messages `INVITE`. Bien que contenant beaucoup plus de messages et d'équipements différents, les jeux opérateurs renferment peu de messages de ce genre car la plupart des sessions SIP visent à enregistrer les différents utilisateurs (type `REGISTER`). Les jeux de données sont

	Essai	T1	T2	T3	T4
#types équipement	26	40	42	40	40
#messages	18066	96033	95908	96073	96031
#message INVITE	3183	1861	1666	1464	1528
#sessions	2686	30006	29775	30328	30063
Moyenne(#messages/session)	6.73	3.20	3.22	3.16	3.20
Moyenne(délai) (sec)	1.53	7.32	6.76	6.11	8.52

TABLE 9.1 – Caractéristiques des jeux de données

bien complémentaires car le jeu opérateur exprime le comportement d'utilisateurs n'appelant que de temps en temps et devant toutefois se réenregistrer pour maintenir l'association entre leur adresse IP et leur identifiant SIP (Address-Of-Record) valide via des sessions initiées par un message `REGISTER` et très courtes en termes de messages. Dans l'objectif de renforcer l'authentification, un fingerprinting efficace de ce type de session est un atout essentiel.

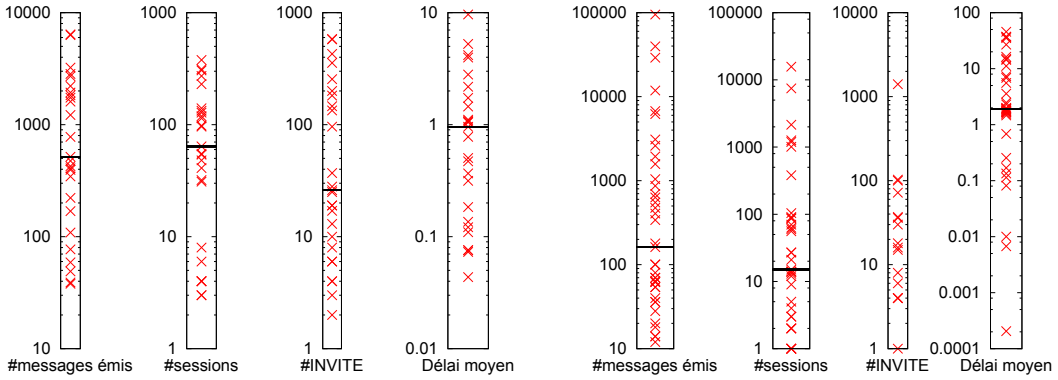
La figure 9.1 présente les caractéristiques de chaque type d'équipement pour le jeu d'essai et pour le premier jeu opérateur (les autres manifestent des valeurs similaires). Chaque point sur les figures représente un type précis et la barre horizontale symbolise la valeur médiane. La distribution du nombre de messages par type n'est uniforme dans aucun cas et est donc réaliste. Cette différence se reporte naturellement sur le nombre de sessions et le nombre de messages émis. Par ailleurs, les distributions concernées sont beaucoup plus étendues dans le cas de l'opérateur (figure 9.1(b)) : un type d'équipement n'a participé qu'à une seule session alors qu'un autre en comptabilise plus de 10.000 à lui seul.

Enfin, le dernier graphe de chaque figure indique le délai entre la réception d'un message et l'envoi du suivant. Le gain d'information de cette métrique est très clairement mis en valeur sur la figure 9.1(b) où quatre voire cinq catégories se démarquent nettement. Intuitivement, l'identification précise de chaque composant des catégories n'est pas faisable d'où l'observation obligatoire du comportement des équipements. Ainsi la méthode de fingerprinting comportementale et temporelle est justifiée. De manière identique au tableau 9.1, la figure 9.1 fait ressortir des délais beaucoup plus longs pour les jeux opérateurs à l'instar d'une valeur médiane deux fois plus grande. Enfin, comme la plupart des sessions de l'opérateur sont des sessions d'enregistrement, beaucoup d'équipements ne présentent pas de message `INVITE` d'où valeur médiane égale à zéro non représentable sur l'échelle logarithmique.

9.3.2 Paramétrage

Les premières expérimentations s'appuyant sur le jeu d'essai mesurent l'impact des différents paramètres afin de paramétrer finement le système de fingerprinting. En effet, le nombre relativement faible de messages permet une analyse rapide alors que les jeux opérateurs seront directement utilisés avec les paramètres déterminés par cette première étape.

La phase d'apprentissage repose sur 40% des sessions tandis que les 60% restant permettront de tester le fingerprinting. Pour augmenter la validité des observations, chaque expérience est accomplie dix fois en mélangeant aléatoirement les sessions à chaque fois. Ainsi, les résultats introduits seront généralement des valeurs moyennes. Cependant, la distribution des résultats obtenus s'avère souvent plus significative qu'une seule valeur moyenne d'où l'usage des quartiles avec les boîtes à moustache à l'instar de la figure 9.2(a) où chaque résultat se manifeste par 5 valeurs différentes : les extrêmes, la limite inférieure de la boîte indique que 25% des résultats observés se situent sous ce seuil alors que la limite supérieure



(a) Essai (b) Opérateur T1

FIGURE 9.1 – Fingerprinting comportemental – Caractéristiques des jeux de données (échelle logarithmique, un point par type, valeur médiane représentée par une barre horizontale)

en représente 75%. Ainsi cette boite renferme 50% des résultats obtenus et est sectionnée par la valeur médiane matérialisée par une barre horizontale. Hormis dans la section 9.3.2, $\alpha = 1000$ dans la fonction de noyau utilisée (voir 6.4.3 pour de plus amples détails sur cette dernière) :

$$K(t_i,t_j) = \sum_{\substack{p\in I_{ij} \\ noeuds(chemin_k^i)=p \\ noeuds(chemin_l^j)=p}} \sum_{n\in p} e^{-\alpha|f_{delai}(n,p_1)-f_{delai}(n,p_2)|}$$

Tailles d'apprentissage et de test

Plusieurs sessions auxquelles participe un même équipement constituent un TR-FSM nécessitant donc de paramétrer le nombre de sessions utilisées pour réaliser cette tâche. Plus particulièrement, le nombre de sessions dans le cadre de l'apprentissage, dénommé *taille d'apprentissage*, et celui dans le cadre des tests, dénommé *taille de test*, sont ajustables et s'expriment en nombre de sessions. La *taille de test* est extrêmement importante puisque plus elle est petite, plus le fingerprinting sera rapide car n'impliquant pas de collecter un grand nombre de sessions.

Le tableau 9.2 expose la précision du fingerprinting selon plusieurs valeurs de ces paramètres grâce à une légende colorée permettant de rapidement visualiser les bonnes et mauvaises configurations. Par exemple, la nouvelle technique de fingerprinting est inapte à discerner convenablement un équipement en se basant sur une seule session comme l'illustre une première colonne très claire. La ligne la plus sombre correspond à une taille d'apprentissage faible égale à cinq suggérant un apprentissage ne nécessitant pas forcément de collecter beaucoup de sessions. Dans le meilleur des cas, plus de 90% des équipements sont convenablement reconnus via une *taille de test* de 10. La nouvelle méthode de fingerprinting n'est donc pas applicable sur une seule session sans pour autant en nécessiter un nombre exagéré. En outre, une précision de 85% est atteignable avec une taille de test limitée à cinq. La précision est stable quelque soit l'expérimentation comme le démontrent les faibles valeurs de l'écart type.

Pour évaluer si les différents types d'équipements sont bien identifiés ou si certains sont totalement ignorés, la sensibilité est reportée dans le tableau 9.3. La meilleure configuration est équivalente à la précédente avec une sensibilité moyenne de 65%. Ce résultat mitigé est dû à quelques types totalement ignorés car peu représentés dans le jeu de données (voir figure 9.1(a)). En effet, une taille d'apprentissage de cinq et une sélection de 40% des sessions pour l'apprentissage impose un nombre minimal de

Taille d'apprentissage	Taille de test				
	1	5	10	20	40
1	0.682 (0.009)	0.819 (0.013)	0.830 (0.013)	0.805 (0.031)	0.745 (0.034)
5	0.469 (0.028)	0.858 (0.013)	0.905 (0.011)	0.883 (0.025)	0.800 (0.035)
10	0.376 (0.044)	0.809 (0.011)	0.894 (0.013)	0.873 (0.021)	0.819 (0.035)
20	0.272 (0.028)	0.656 (0.028)	0.821 (0.015)	0.864 (0.015)	0.837 (0.012)
40	0.221 (0.027)	0.469 (0.026)	0.627 (0.030)	0.764 (0.037)	0.762 (0.038)

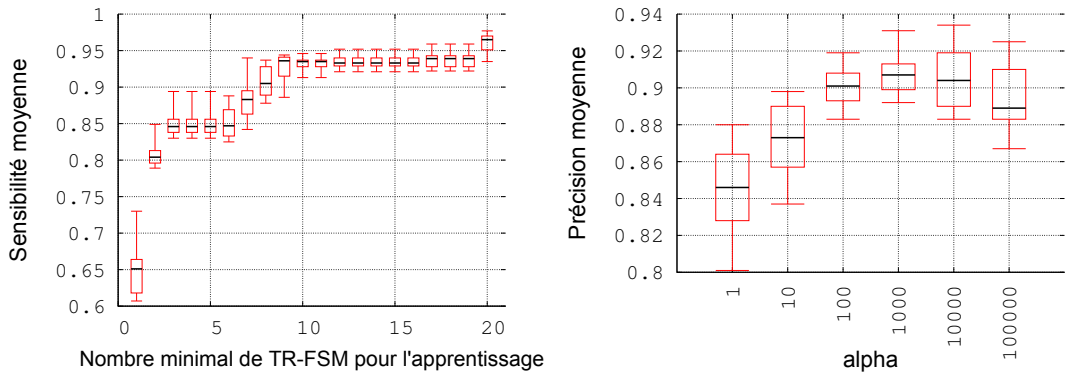
$< 50\%$	50-70%	70-80%	80-85%	85-90%	$\geq 90\%$

TABLE 9.2 – Fingerprinting comportemental – Jeu d'essai : précision moyenne (écart type entre parenthèses)

Taille d'apprentissage	Taille de test				
	1	5	10	20	40
1	0.504 (0.011)	0.542 (0.034)	0.553 (0.032)	0.535 (0.044)	0.529 (0.043)
5	0.294 (0.026)	0.605 (0.035)	0.647 (0.035)	0.648 (0.047)	0.580 (0.045)
10	0.224 (0.028)	0.550 (0.017)	0.625 (0.023)	0.636 (0.024)	0.599 (0.047)
20	0.145 (0.021)	0.452 (0.050)	0.572 (0.030)	0.615 (0.045)	0.622 (0.027)
40	0.109 (0.028)	0.316 (0.030)	0.399 (0.032)	0.505 (0.050)	0.522 (0.038)

$< 30\%$	30-40%	40-50%	50-55%	55-60%	$\geq 60\%$

TABLE 9.3 – Fingerprinting comportemental – Jeu d'essai : sensibilité moyenne (écart type entre parenthèse)



(a) Nombre minimal de TR-FSM nécessaire pour l'apprentissage ($\alpha = 1000$)

(b) impact du paramètre α

FIGURE 9.2 – Paramétrage du fingerprinting comportemental (taille de test = 10, taille d'apprentissage = 5)

$\lceil 5/0.4 \rceil = 13$ sessions. Malheureusement, comme le montre la figure 9.1(a), six types d'équipements ne contribuent pas suffisamment pour respecter cette contrainte. Par ailleurs, cette valeur minimale implique la création d'une seule signature (TR-FSM) alors que les techniques d'apprentissage ont besoin de plusieurs exemples pour pouvoir classer correctement les instances testées par la suite. La sous-section suivante visera donc à déterminer l'impact du nombre minimal de signatures utilisées pendant l'apprentissage.

De manière générale, l'approche de fingerprinting proposée semble être plus performante avec des tailles d'apprentissage et de tests différentes pouvant paraître illogiques mais s'expliquant simplement par le fait que cette technique évalue individuellement les chemins communs des arbres TR-FSM indépendamment de leur taille (équations (6.5)-(6.8) de la section 6.4.3). Enfin, en testant la même configuration optimale décelée précédemment mais en ignorant les données temporelles, la précision atteint 83%, soulignant le bénéfice de ce type d'information.

Nombre de signatures nécessaires pour l'apprentissage

L'expérimentation précédente souligne une sensibilité très altérée par des types d'équipements dont le nombre de sessions sélectionnées pour l'apprentissage était trop faible. Ainsi, pour chacun d'eux, ce nombre est fixé à une valeur absolue et non plus à un pourcentage. L'axe des abscisses de la figure 9.2(a) représente le nombre de TR-FSM minimal servant pour l'apprentissage variant de 1 à 20 dans le cadre de la meilleure configuration obtenue précédemment. Ainsi dès que le nombre minimal de TR-FSM est seulement de deux (10 sessions), la sensibilité moyenne atteint 80%, ce qui est beaucoup plus acceptable que les 65% obtenus auparavant. Parvenir à 90% est concevable en imposant un minimum de huit TR-FSM. Enfin, les boîtes à moustache correspondantes sont étroites signifiant que la sensibilité est stable. Dans la suite de cette étude, SVM forcera l'apprentissage sur deux arbres minimum.

Impact du paramètre α

Le paramètre α de la fonction noyau joue un rôle majeur car plus il est grand, plus les différences entre les petits délais sont amplifiées. La figure 9.2(b) étudie cet aspect en modulant ce paramètre entre un et 100.000. D'un de point de vue général, la courbe suit une forme parabolique. En effet, une différence de délai entre une et quatre secondes doit être interprétée différemment d'une différence entre 56 et

59 secondes d'où l'obligation d'avoir un facteur $\alpha > 1$. Cependant, dès lors qu'α est trop élevé, les différences minimes sont trop exagérées. Cette expérimentation aide à choisir le bon compromis et met en exergue trois valeurs acceptables : 100, 1000 ou 10.000. Toutefois, des résultats stables, c'est-à-dire concentrés autour de la médiane, sont préférables car ils signalent que les bons résultats ne sont pas dépendants du processus aléatoire de sélection des sessions d'apprentissage d'où une préférence pour $\alpha = 1000$.

9.3.3 Réseau opérateur

Suite aux résultats précédents, une taille d'apprentissage de dix et une taille de test de cinq sont considérées. Le tableau 9.4 résume l'ensemble des statistiques et des résultats sur les jeux opérateurs tout en reprenant les résultats précédents. Néanmoins, les jeux opérateurs étant plus grands, SVM fut entraîné en n'employant que 10% des sessions seulement tout en répétant trois fois chaque expérience.

La première partie du tableau témoigne de quelques statistiques sur les jeux de données après construction des arbres TR-FSM. Hormis le nombre d'arbres qui est automatiquement fixé par les paramètres définis précédemment et reste invariable à chaque expérience, les autres métriques sont évaluées selon leur moyenne et leur écart type sur toutes les instances d'expériences réalisées. La structure des arbres est très instable car les écarts types des cardinalités et des hauteurs maximales et moyennes sont très élevés attestant d'une multitude de configurations et donc d'une plus grande validité des résultats. La technique de fingerprinting comportemental est visiblement une méthode prometteuse car, malgré ces nombreuses configurations, les performances (précision, sensibilité et spécificité) sont très stables (faible écart type).

En regardant les performances de plus près pour les jeux de données opérateurs, la précision se situe aux alentours des 86%, c'est-à-dire légèrement en dessous du jeu d'essai dans lequel 91% des équipements étaient correctement identifiés. Cependant, cette légère différence est logique et peu élevée en comparaison de la différence entre un réseau local et un environnement bruité tel qu'Internet. Grâce au paramétrage finement réalisé auparavant, le coefficient d'information mutelle prouve que, dans le cas du jeu d'essai, les bons résultats ne sont pas uniquement le fruit de quelques types d'équipements sur-représentés et bien classés. Cependant, le problème des types complètement sous-représentés est encore plus gênant dans le cas de l'opérateur avec un coefficient d'information mutuelle aux alentours de 0.64. Effectivement, la figure 9.1(b) en comparaison avec la figure 9.1(a) laissait entrevoir que certains types ne seraient pas découverts car vraiment trop peu représentés par rapport à la taille globale du jeu de données. Ils sont alors classés avec une sensibilité oscillant entre 45% et 58% dégradant fortement la sensibilité générale moyenne. Les équipements mal reconnus sont cependant répartis dans toutes les autres classes car la spécificité reste élevée.

Enfin, l'évaluation de la classification suppose que l'identité d'un équipement soit contenue dans le champ SIP User-Agent pouvant ainsi facilement être modifiée. Alors que pour le jeu d'essai, cette information est sûre, elle n'est malheureusement pas vérifiable pour l'opérateur d'autant plus que certains équipements ont un type inconnu.

9.4 Fingerprinting syntaxique

Pour rappel, les différentes méthodes peuvent utiliser plusieurs distances entre deux arbres t_1 et t_2 rappelées ci-dessous :

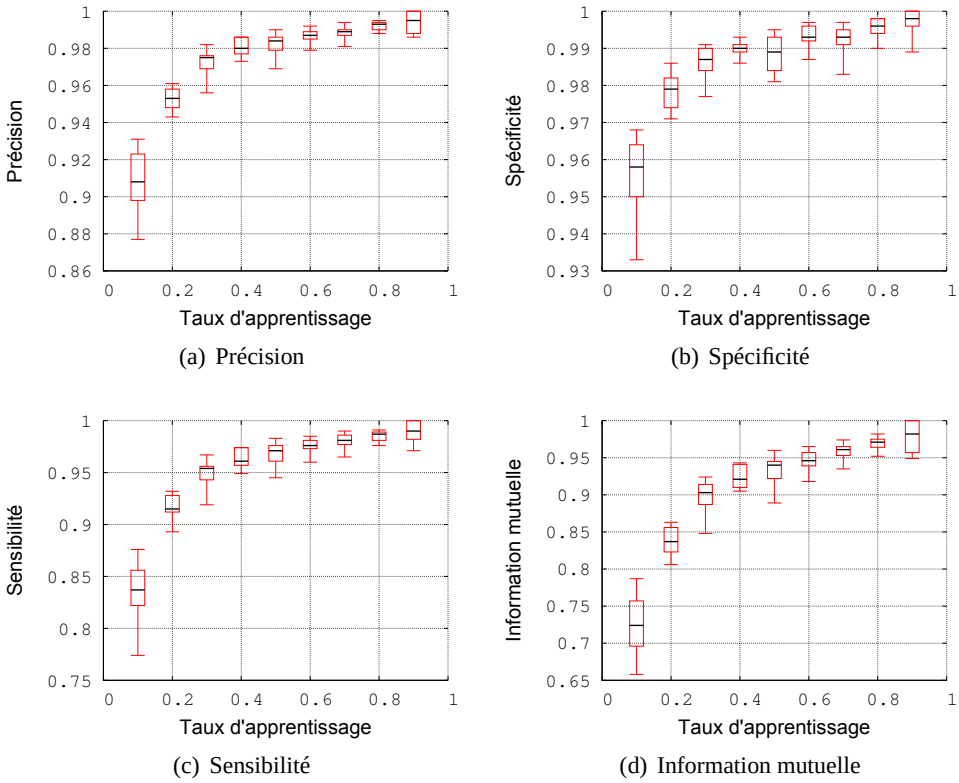
$$d1(t_1,t_2) = |t_1| + |t_2| - 2W(\phi_{12}) \text{ (non normalisée supervisée ou non)}$$

Métrique	Essai	T1	T2	T3	T4
#TR-FSM pour l'apprentissage	440	1223	1217	1237	1224
#TR-FSM testés	332	5409	5367	5471	5423
Hauteur maximale d'un TR-FSM	71.95 (32.03)	464.67 (41.35)	476.33 (38.58)	420.33 (30.56)	431.33 (0.94)
Hauteur minimale d'un TR-FSM	1.9 (0.30)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)	1.00 (0.00)
Hauteur moyenne d'un TR-FSM	9.53 (2.13)	8.80 (1.53)	8.85 (1.89)	8.70 (1.73)	9.05 (1.38)
Cardinalité maximale d'un TR-FSM	89.00 (35.72)	492.67 (44.68)	491.17 (47.65)	540.84 (157.00)	464.84 (21.52)
Cardinalité minimale d'un TR-FSM	3.95 (1.56)	2.67 (0.47)	2.00 (0.00)	2.00 (0.00)	3.00 (0.00)
Cardinalité moyenne d'un TR-FSM	18.97 (4.69)	12.93 (2.68)	12.94 (3.09)	12.85 (2.98)	13.23 (2.56)
Précision	0.91 (0.011)	0.81 (0.004)	0.86 (0.001)	0.85 (0.002)	0.83 (0.004)
Sensibilité	0.64 (0.030)	0.53 (0.019)	0.58 (0.026)	0.54 (0.012)	0.43 (0.015)
Spécificité	0.91 (0.035)	0.79 (0.001)	0.81 (0.025)	0.77 (0.028)	0.77 (0.028)
Information mutuelle	0.87 (0.012)	0.64 (0.001)	0.65 (0.001)	0.65 (0.003)	0.63 (0.004)

TABLE 9.4 – Fingerprinting comportemental – Statistiques et résultats généraux ($\alpha = 1000$, taille de test = 10,taille d'apprentissage = 5). Valeurs moyennes (écart type entre parenthèses)

Type d'équipement	#messages	Hauteur			#nœuds		
		Max	Min	Moyenne	Max	Min	Moyenne
Asterisk_v1.4.21	1081	28	23	25	2517	883	1284
Cisco-7940_v8.9	168	25	23	24	2784	812	1352
Thomson2030_v1.59	164	28	23	24	2576	793	1391
Twinkle_v1.1	195	25	23	23	2457	805	1299
Linksys_v5.1.8	195	28	23	25	2783	852	1248
SJPhone_v1.65	288	30	23	24	2330	951	1133

TABLE 9.5 – Fingerprinting syntaxique – Statistiques sur les arbres syntaxiques



(a) Précision (b) Spécificité

(c) Sensibilité (d) Information mutuelle

FIGURE 9.3 – Fingerprinting syntaxique supervisé, distance d_1

$$d2(t_1,t_2) = 1 - \frac{W(\phi_{12})}{max(|t_1|,|t_2|)} \ (normalisée\ non\ supervisée)$$

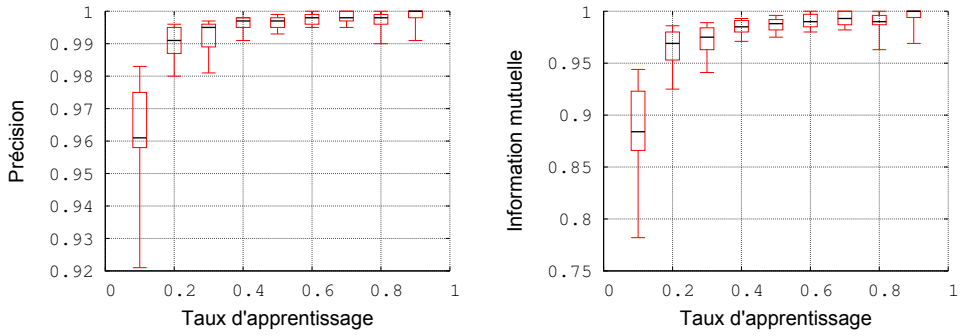
$$d3(t_1,t_2) = 1 - \frac{W(\phi_{12})}{|t_1|+|t_2|-W(\phi_{MAX})} \ (normalisée\ supervisée)$$

où $W(\phi_{12})$ mesure la similarité entre deux arbres à partir du plus grand sous-arbre commun.

9.4.1 Jeux de données

L'évaluation du fingerprinting syntaxique s'appuie sur deux jeux de données distincts. Le premier, dont les caractéristiques sont reportées dans le tableau 9.5, fut généré grâce à une plateforme d'essai et les expérimentations se baseront généralement dessus à moins que ce ne soit mentionné. Il comprend six types d'équipements distincts pour un total de 2091 messages. De plus, les caractéristiques révèlent que les arbres syntaxiques réels sont beaucoup plus compliqués (800 nœuds, hauteur de 25-30 nœuds) que ceux utilisés dans la chapitre 3 à titre d'exemple. Modifier la structure d'un tel arbre sans en changer la sémantique est donc réellement très compliqué d'où la légitimité de l'approche proposée.

Le second jeu de donné provient du même opérateur VoIP que précédemment (T1) avec les 40 types d'équipements distincts. Cependant, ceux qui ne sont pas assez représentatifs sont éliminés (moins de 20 messages). De cette manière, seuls 36 types sont conservés en ne gardant qu'un maximum de 100 messages pour chacun d'eux.



(a) Précision (b) Information mutuelle

FIGURE 9.4 – Fingerprinting syntaxique supervisé, distance d_3

9.4.2 Fingerprinting supervisé

Taux d'apprentissage

Dans le cadre du fingerprinting supervisé, une partie des messages est extraite du jeu de données pour entraîner SVM et se définit comme un pourcentage du nombre total de messages nommé taux d'apprentissage. Comme les résultats ne sont pas forcément identiques selon les messages sélectionnés pour l'apprentissage, ils sont volontairement mélangés avant d'être séparés entre jeux d'apprentissage et de test. De plus, les expérimentations seront répétées dix fois pour renforcer leurs validité. Un fingerprinting efficace implique un taux d'apprentissage faible d'où la figure 9.3 qui cherche à mesurer les performances selon différents taux en utilisant la distance d_1 (non normalisée). La figure 9.3(a) révèle que cette nouvelle technique de fingerprinting est très efficace car, malgré un taux d'apprentissage faible tel que 10%, 90% des équipements sont correctement reconnus. Bien évidemment, plus le taux augmente, plus le système est mieux entraîné d'où de meilleurs résultats et, dès un taux d'apprentissage de 20%, la représentation des quartiles indiquent des résultats peu fluctuants. Par conséquent, réussir à identifier un équipement avec 95% de chances est largement envisageable. La sensibilité sur la figure 9.3(c) est également élevée mais légèrement plus faible signifiant que la bonne classification générale n'est pas dûe à quelques types majeurs seulement. La spécificité indique quant à elle une dispersion des arbres mal identifiés dans toutes les classes. Le coefficient d'information mutuelle de la figure 9.3(d) récapitule ces observations en une seule et se voit alors légèrement diminué car il « combine » la sensibilité et la spécificité. Cette métrique est la plus sévère en amplifiant les effets négatifs (valeurs inférieures à 1) de la sensibilité et de la spécificité. Par conséquent, cette métrique sera préférée par la suite tout en conservant la précision générale donnant également une vision globale intéressante des performances.

Distances

Contrairement à la distance d_1, d_3 est normalisée rendant son utilisation plus simple car les techniques d'apprentissage telles que celles mises en œuvre par SVM requièrent des valeurs normalisées. Les performances obtenues surpassent clairement les précédentes comme le met en lumière la figure 9.4. Par exemple, avec un taux d'apprentissage de 20% seulement, le type de 99% des équipements est identifié justement. Par ailleurs, l'information mutuelle est également très élevée. Finalement, à précision égale, un taux d'apprentissage de 20% avec d_3 concorde avec 80% avec d_1. En clair, le fingerprinting supervisé utilisant les SVM est très efficace grâce à l'utilisation de d_3.

γ (#voisins)	τ (distance minimale entre voisins)				
	0.01	0.05	0.1	0.15	0.2
1	0.559 (0.674)	0.767 (0.805)	0.721 (0.697)	0.307 (0.339)	0.302 (0.614)
5	0.480 (595)	0.748 (0.787)	801 (0.781)	0.306 (0.336)	0.306 (0.399)
10	0.454 (0.570)	0.742 (0.784)	0.872 (0.879)	0.307 (0.293)	0.307 (0.293)
15	0.424 (0.542)	0.727 (0.767)	0.862 (0.902)	0.525 (0.489)	0.307 (0.293)
20	0.370 (0.497)	0.698 (0.744)	0.804 (0.852)	0.524 (0.488)	0.307 (0.293)

$< 40\%$	40-60%	60-70%	70-80%	80-85%	$\geq 85\%$

TABLE 9.6 – Fingerprinting syntaxique non supervisé en regroupant les messages proches «temporellement», $\rho = 5$ sec. ,distance d_2 - Précision (F-Score entre parenthèses)

9.4.3 Fingerprinting non supervisé

En appliquant directement ROCK (version modifiée) sur les données brutes, les résultats obtenus furent très modérés en atteignant une précision de 60% au maximum car, du fait d'une syntaxe riche, les messages émis par un même type d'équipement peuvent se révéler très différents, notamment selon la nature de ces derniers. Par exemple, les messages SIP pour initier un appel ou s'authentifier ne présentent pas la même sémantique et ne contiennent donc pas les mêmes champs impliquant des arbres syntaxiques trop divergents pour pouvoir faire partie du même cluster. Dans la section précédente, le jeu d'apprentissage pouvait contenir ces différents types de messages d'où de bonnes performances. Envisager un autre type de fingerprinting non supervisé impose :

- d'identifier les équipements en n'analysant qu'un seul type de messages comme par exemple `INVITE` ou `REGISTER`,
- de créer des micro-clusters au début en supposant qu'une même adresse IP et un même port représentent un même équipement tant que les messages émis par ce dernier sont séparés par un faible laps de temps (quelques secondes). Cette hypothèse est réaliste car ces caractéristiques ne changent pas aussi fréquemment.

Enfin, comme l'algorithme de classification est déterministe et qu'aucun jeu d'apprentissage ne doit être sélectionné, répéter les expérimentations est tout à fait inutile car les résultats seraient constants. Comme expliqué dans la section 9.2, les résultats basés sur le F-Score compléteront la précision générale.

Fenêtre temporelle

Les messages de la même machine (adresse IP et port) se suivant dans un intervalle de temps limité défini par ρ égal à cinq secondes sont regroupés. De cette manière, les micro-clusters préétablis contiennent une moyenne de 2,8 messages. Ainsi, les autres paramètres de ROCK vont pouvoir être testés :

- τ : la distance maximale entre deux messages voisins,
- γ : le nombre minimal de voisins pour que deux messages fassent partie de la même catégorie.

τ	0.01	0.05	0.1	0.15	0.2
#clusters	314	108	61	33	14
Nombre d'arbres dans un cluster (minimum)	1	1	1	1	1
Nombre d'arbres dans un cluster (maximum)	126	218	222	480	720
Nombre d'arbres dans un cluster (moyenne)	2.33	6.79	12.02	22.212	52.357

TABLE 9.7 – Caractéristiques des clusters avec $\gamma = 15$

γ	1	5	10	15	20
#clusters	18	38	47	61	82
Nombre d'arbres dans un cluster (minimum)	1	1	1	1	1
Nombre d'arbres dans un cluster (maximum)	353	224	223	222	220
Nombre d'arbres dans un cluster (moyenne)	40.72	19.29	15.60	12.02	8.94

TABLE 9.8 – Caractéristiques des clusters avec $\tau = 0.1$

Excepté pour les quatre valeurs encadrées dans le tableau 9.6, toutes les autres configurations sont parvenues à découvrir la totalité des types réels existants. Les colonnes très claires ($\tau = 0.01, 0.15, 0.2$) sont synonymes d'une mauvaise configuration et montrent que la précision n'est pas une fonction monotone par rapport à τ. Le dégradé non continu des colonnes révèle aussi qu'elle n'est pas non plus monotone en fonction de γ. Plus en détails, 87% des équipements sont reconnus en utilisant $\tau = 0.1$ et $\gamma = 10$ soit dix points de plus que dans le meilleur des cas avec $\gamma = 1$. Cette observation est primordiale car elle prouve l'avantage certain du nouvel algorithme proposé dans le chapitre 3 par rapport à QROCK, équivalent à $\gamma = 1$. Son utilisation peut donc être envisagée pour d'autres domaines d'application. Les mesures de F-score élevées associées aux configurations obtenant les meilleures précisions signalent une fois de plus que ces bons résultats ne sont pas juste la conséquence de la surreprésentativité de certains types d'équipements.

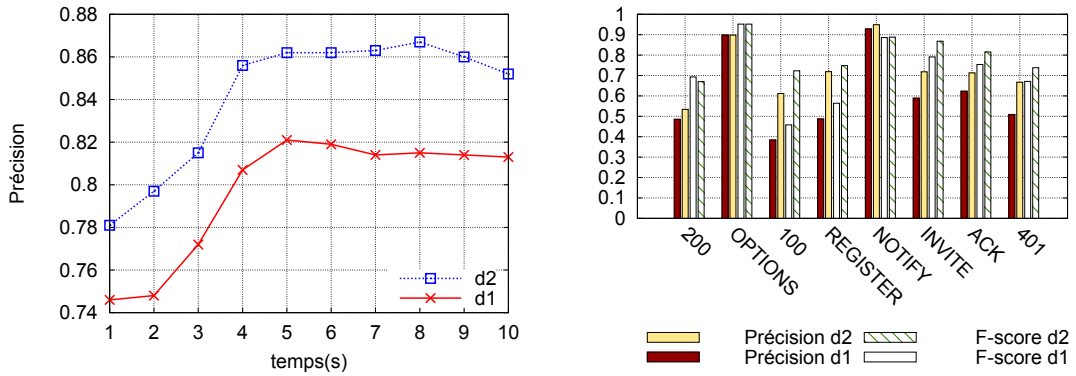
Malgré une précision légèrement inférieure, la configuration fixant $\gamma = 15$ et $\tau = 0.1$ est accréditée d'un F-score remarquablement plus élevé par rapport à la diminution de la précision d'où ce paramétrage par la suite hormis pour les tests suivants :

- tableau 9.7 : $\gamma = 15$ et τ varie,
- tableau 9.8 : $\tau = 0.5$ et γ varie.

Le but n'est plus d'étudier la performance de la classification, car l'expérience précédente y était dédiée, mais de voir la structure des clusters selon ces deux paramètres complétant ainsi l'analyse de cette nouvelle technique de clustering permettant de bien comprendre l'impact de chaque paramètre.

Premièrement, il existe toujours des clusters d'un seul arbre constituant ainsi un ensemble de messages difficilement identifiables. Lorsque τ augmente, les messages ont plus de voisins permettant d'en fusionner une plus grande partie d'où une diminution du nombre de clusters ainsi qu'une augmentation de leurs tailles. Un accroissement de γ impose aux messages de partager plus de voisins en commun pour pouvoir être rassemblés dans le même cluster. Ainsi, moins de messages sont regroupés au sein d'un même cluster engendrant les effets inverses de l'augmentation de τ. Enfin, les clusters pour $\gamma = 1$ comparés avec $\gamma = 15$ sont totalement différents aussi bien par leurs tailles que par leur nombre passant de 18 à 61. Surévaluer le nombre de catégories est donc nécessaire pour obtenir de bonnes performances car plusieurs d'entre-eux sont composés uniquement d'un message. Plus exactement, 53 clusters ne contiennent qu'un message avec la configuration optimale.

Enfin, la distance d_2 est confrontée à la distance non normalisée d_1 dans la figure 9.5(a) qui étudie également l'influence de ρ. L'avantage de la distance normalisée d_2 est sans équivoque alors qu'un regroupement initial de messages trop excessif (supérieur à cinq secondes) n'a pas d'impact positif.



(a) Regroupement temporel

(b) Par types

FIGURE 9.5 – Fingerprinting syntaxique non supervisé amélioré

Types de messages

La classification non supervisée est ici opérée sur un seul type de messages à chaque fois parmi les plus répandus : `100`, `200`, `401`, `OPTIONS`, `REGISTER`, `NOTIFY`, `INVITE` et `ACK`. Différentes distances sont testées sur la figure 9.5(b). Une fois encore, la distance normalisée, d_2, parvient à mieux distinguer les équipements quelque soit le cas mais la figure expose également des différences remarquables selon le type de messages. En effet, un message `OPTION` renseigne sur les capacités d'un équipement et y est donc très dépendant à l'inverse du message `200` assimilable à un acquittement. Ainsi une identification efficace des équipements est susceptible de se baser sur les messages `OPTION` ou `NOTIFY` car ils présentent une précision et un F-score proche de 90%.

Dans le cadre d'une authentification renforcée, un équipement est correctement reconnu à 70% grâce à un message `REGISTER`. Bien que ce résultat soit trop faible pour discrimer des utilisateurs normaux d'intrus, il peut aider à cibler les machines pour lesquelles des vérifications supplémentaires sont nécessaires.

9.4.4 Opérateur

Le fingerprinting syntaxique fut également entrepris sur le jeu de données opérateur. La classification supervisée identifie correctement 70% des équipements alors que la technique non supervisée parvient à en distinguer 75% en regroupant les messages proches dans le temps et même 90% en classant uniquement les messages `OPTIONS`, confirmant que la structure syntaxique d'un tel message est très dépendante de l'équipement l'émettant. Cependant, les précisions relativement faibles demandent d'en investiguer plus profondément les causes. Certains équipements ne sont pas discernables tels que `CiscoATA186v3.1.0` et `CiscoATA186v3.1.1` à cause du peu voire de l'absence de différence d'implantation de la pile protocolaire. Cependant, ne pas détecter des variations mineures d'un type d'équipement n'est pas critique car ils présentent les mêmes fonctionnalités et sont soumis aux mêmes failles de sécurité. Dans ce cas, le fingerprinting est donc efficace puisque ces deux aspects relèvent des applications visées par ce type de fingerprinting.

En outre, n'étant pas maître de ce jeu de données, vérifier sa justesse et notamment l'identification des équipements grâce au champ User-Agent n'est pas faisable.

9.5 Conclusion

Comme introduit au début de ce chapitre, différentes techniques de fingerprinting complémentaires en termes de connaissances nécessaires (jeu d'apprentissage, grammaire) furent évaluées. Elles présentent toutes de bon résultats prouvant leur capacité à être utilisées pour des applications réelles : inventaire automatique des équipements, authentification renforcée, détection d'intrus, offre de services personnalisée...

Ce chapitre a donc mis en évidence les bonnes performances des SVM dans ce domaine d'application ainsi que la viabilité de la nouvelle représentation comportementale grâce aux TR-FSM tout en validant également la fonction noyau applicable sur ces derniers. D'un autre côté, la représentation des messages par un arbre syntaxique est beaucoup plus efficace car elle ne nécessite qu'un seul message mais demande néanmoins de connaître intégralement la grammaire du protocole contrairement aux TR-FSM. De plus, le nouvel algorithme de classification non supervisé s'est montré également performant pour classer chaque message un à un comparé à sa version initiale.

Enfin, les nombreux tests ont mis en évidence l'impact des différents paramètres des méthodes pouvant aider par la suite à mieux configurer de telles méthodes.

Conclusion générale

Face à l'essor d'Internet aussi bien en termes de nombre de machines connectées que de nombre d'applications existantes aujourd'hui, l'introduction de cette thèse avait mis en avant plusieurs défis majeurs :

- conception efficace d'une application communicante,
- sécurité,
- économie (rentabilité),
- supervision.

Bien entendu, cette thèse n'apporte pas toutes les réponses à ces questions mais contribue à leur résolution. Alors que beaucoup d'applications communicantes sont généralement confrontées au problème du passage à l'échelle, lui-même directement dépendant de la conception, les différents modèles d'applications communicantes constituant une des contribution de cette thèse aident à mieux l'évaluer. Une bonne conception d'un protocole passe également par une bonne compréhension des protocoles qui peut s'apparenter parfois à une étape de rétro-ingénierie, domaine abordé par la seconde contribution de cette thèse.

La réponse au problème de la sécurité est relatif à plusieurs des travaux exposés à l'instar de la modélisaton des botnets, permettant une meilleure compréhension du danger associé, mais également grâce au fingerprinting (troisième contribution) qui peut servir à repérer des intrus sur le réseau. Cette dernière contribution peut également jouer un rôle économique car l'identification précise d'un équipement peut servir de support pour fournir un service personnalisé à son utilisateur.

Enfin, la modélisation des botnets a été l'occasion de proposer une nouvelle solution de supervision pour résoudre le problème du passage à l'échelle.

En clair, les trois contributions de cette thèse sont :

1. modélisation de botnets,
2. découverte automatique des types de messages (rétro-ingénierie),
3. fingerprinting d'équipements.

1 Résumé des contributions

Modélisation des applications communicantes

La modélisation des applications communicantes présentée dans cette thèse s'est focalisée sur les botnets mais demeure néanmoins générique. Effectivement, les nouvelles métriques définies (les différentes mesures d'atteignabilités et de délais) peuvent être employées pour évaluer toute application

dans le contexte de l'envoi massif d'une information. Dans ce sens, les modèles aident à mesurer le passage à l'échelle des applications communicantes.

De plus, les modèles ne sont pas spécifiques aux botnets puisque leurs architectures ne furent pas créées de toute pièce mais reposent sur des bases existantes telles qu'un protocole pair-à-pair ou le protocole IRC. Les deux modèles choisis représentent quant à eux deux types d'architecture bien différents : pseudo-centralisé (IRC) ou totalement distribué (P2P). Enfin, un des types de réseau P2P modélisé se caractérise par une table de hachage distribuée représentant la majorité des protocoles P2P utilisés actuellement alors que le contexte de cette thèse n'engendre pas de différences notables sur les métriques selon les différents protocoles de table de hachage distribuée. Par ailleurs, différentes méthodes formelles furent mises en œuvre pouvant servir d'inspiration à l'élaboration d'autres modèles dans l'avenir.

Les expérimentations menées évaluent les performances des différents systèmes selon de nombreux paramètres afin de bien comprendre l'impact des différents paramètres qui prennent en compte également plusieurs contraintes. Ces dernières sont elles aussi suffisamment générale pour être utilisables avec tout type d'applications communicantes : perte de connectivité, attaque et anonymat. En conséquence, privilégier une architecture ou une autre s'en retrouve facilité pour éviter d'être confronté par la suite aux problèmes relatifs du passage à l'échelle ou de la sécurité.

Enfin, dans le contexte des botnets représentant une menace actuelle importante, les modélisations proposées ainsi que leurs évaluations sont un réel atout pour mieux les appréhender afin de mieux s'en prémunir. Par exemple, une contre-mesure consiste à forcer certains bots à se déconnecter pour ainsi affaiblir le botnet. Les travaux réalisés dans cette thèse montrent quel est l'impact de ces déconnexions puisque, dans ce cas, cela correspond à des attaques envers le botnet.

Rétro-ingénierie des protocoles

Cette thèse traite d'une partie fondamentale de la rétro-ingénierie visant à déterminer le type d'un message, une information nécessaire pour la reconstruction de la machine à états d'un protocole. La technique introduite dans le chapitre 4 est novatrice dans deux sens.

Premièrement, le clustering grâce à la méthode SVC est peu utilisé dans le domaine des réseaux contrairement aux méthodes de classification classiques. Basé sur les machines à vecteurs supports dont les performances sont très bonnes, SVC a l'avantage d'être non supervisé et est donc applicable avec un protocole totalement inconnu. Alors que de premier abord, son utilisation semble être difficile car définir les paramètres n'est pas une tâche aisée s'il est impossible de vérifier la précision de la classification sur un jeu d'apprentissage (constructible si et seulement si le protocole est connu), cette thèse expose également une méthode d'identification automatique des paramètres. Par conséquent, l'applicabilité de cette technique s'en retrouve grandement améliorée et ne se limite pas au contexte de cette thèse (rétro-ingénierie). En effet, le procédé décrit dans cette thèse peut probablement être instancié pour d'autres travaux de divers domaines employant la méthode SVC. Enfin, les expérimentations menées ont mis l'accent sur l'avantage d'une combinaison de SVC avec du partitionnement hiérarchique.

Sur un autre plan complémentaire, le chapitre 4 s'est efforcé de souligner la nécessité de définir une nouvelle représentation des messages pour pouvoir rapidement les comparer les uns aux autres. En effet, les trois contraintes imposées sont :

- aucune hypothèse forte n'est formulée sur la syntaxe du protocole comme l'utilisation de certains caractères délimiteurs ou la grammaire,
- la méthode doit être non supervisée puisque le contraire signifierait que la construction d'un jeu d'apprentissage est réalisable, ceci n'étant possible que si le protocole est connu,

– aucun analyse par tainting, consistant à observer comment une application analyse le message, ne peut être mise en œuvre. Ainsi les coûts sont diminués puisqu'il n'y a pas besoin d'installer le logiciel en question, ni d'instrumenter la machine, ni de générer les différents cas d'utilisations pour être sûr d'observer tous les types de messages.

Face à ces contraintes, les méthodes actuelles se basent essentiellement sur l'alignement de séquences qui a la particularité d'être appliqué sur la totalité des messages rendant ainsi cette approche très coûteuse en temps de calcul. Partant de ce constat, le but fut de proposer une représentation compressée des messages tout en préservant un maximum de sémantique afin d'être capable d'en extraire le type. Plusieurs représentations ont été introduites avec un descriptif détaillé de leurs avantages et leurs inconvénients. Les expérimentations montrent également les différences notables dans la découverte des types de messages selon la représentation choisie, ce qui met clairement en évidence les différents degrés de sémantique associée à chacun d'entre elles. Ainsi, les différents tests réalisés ne se sont pas uniquement contentés de mettre en exergue la faisabilité de l'approche proposée.

Fingerprinting d'équipements

Concernant le fingerprinting d'équipements, cette thèse énonce deux méthodes différentes. La première est totalement nouvelle et se propose d'identifier un équipement par son comportement. La seconde est également innovatrice en se basant sur la structure syntaxique d'un message et se rapproche ainsi des travaux exposés dans [8], à la différence près que sa complexité est très largement réduite et que son application est plus rapide car les signatures sont rapidement mises à jour sans nécessiter l'utilisation d'une grille de calcul comme ce fut le cas pour [8].

La méthode comportementale considère qu'un équipement est identifiable par la manière dont il échange des messages avec les autres : envois de requêtes et réponses. Cette technique repose uniquement sur les types de messages ainsi que les délais entre eux. Par conséquent, la seconde contribution (rétro-ingénierie) présentée précédemment prend tout son sens ici rendant ainsi cette nouvelle méthode de fingerprinting applicable à un protocole quelconque. Son emploi s'en retrouve facilité, que se soit pour un protocole dont les spécifications sont inconnues ou lorsque ces dernières sont complexes et nécessitent de concevoir un analyseur, tâche qui peut être elle-même difficile et coûteuse en temps. Dans ce dernier cas, ce type de fingerprinting est vraiment utile pour réaliser une étude préliminaire sur l'utilisation d'un service. Par exemple, si un grand nombre d'utilisateurs adoptent un nouveau service, leur proposer un support peut être intéressant. Dans ce cas, le fingerprinting d'équipements peut indiquer quel est le type d'équipements le plus utilisé. Cet exemple est par ailleurs valable avec toute méthode de fingerprinting. Celle basée sur le comportement définit une fonction noyau dont la validité a été formellement prouvée et entraînant une précision d'identification élevée grâce aux machines vecteurs supports.

La seconde technique de fingerprinting repose sur la construction des arbres syntaxiques des messages et s'inspire de travaux récents énonçant de nouvelles métriques efficaces pour calculer la similitude entre deux arbres. Il devient alors possible d'appliquer les machines à vecteurs supports avec des résultats très performants en identifiant pratiquement intégralement l'ensemble des équipements. Par ailleurs, un nouvel algorithme totalement non supervisé fut proposé résultant d'un compromis entre précision et rapidité d'exécution. Les résultats obtenus sont également très encourageants.

Bien entendu, l'évaluation des différentes méthodes ne se restreint pas à donner les meilleurs résultats mais montre aussi les limites des différentes méthodes tout en évaluant l'impact des différents paramètres. Ainsi, l'application réelle d'une méthode ou d'une autre n'est pas faite au hasard.

2 Perspectives

Les travaux de recherche réalisés dans le cadre de cette thèse ouvrent plusieurs perspectives. Certaines d'entre elles sont à court et moyen terme comme une validation renforcée des techniques ou le passage d'outils expérimentaux à des logiciels utilisables par quiconque. Dans une optique à long terme, elle ouvre la voie à de nouvelles méthodes de fingerprinting en montrant que la sémantique discriminante d'un équipement ne se restreint pas à extraire certaines valeurs spécifiques d'un message.

Instanciation des modèles des botnets

Les modèles de botnets proposés dans cette thèse comportent plusieurs paramètres pour lesquels les expérimentations ont mis en évidence leurs impacts. Ainsi, instancier ces modèles selon des paramètres réels permettrait de prévoir l'effet d'un nouveau botnet par exemple. Toutefois, cette tâche est très difficile car elle impose plusieurs conditions comme par exemple la nécessité de capturer le malware dans un premier temps pour l'étudier et ainsi découvrir le modèle le mieux adapté. La plus grande difficulté relève du paramétrage, comme les risques de déconnexion ou d'attaque car ces derniers sont en fait très dépendants de l'environnement (types de réseaux, fuseaux horaires, réactivité des antivirus...). Il est clair que ce travail relève d'une grande difficulté mais l'instanciation des modèles avec des paramètres réels permettrait également de valider plus précisément l'approche en comparant les résultats obtenus avec des mesures réelles.

Approfondissement des modèles des botnets

Les modélisations proposées dans cette thèse visent à mesurer les performances atteignables selon différentes contraintes génériques. Cependant, l'aspect humain n'est pas pris en compte, comme par exemple le comportement de l'attaquant ou celui de ceux qui se défendent. En effet, un attaquant ne cherchera pas forcément à atteindre l'ensemble des bots s'il n'en n'a pas besoin car il s'exposerait ainsi à un plus grand risque de détection. Une évolution de ce travail serait alors de prendre en compte ces aspects comportementaux en utilisant la théorie des jeux en intégrant les deux parties (attaquant et défenseur) pour enrichir le modèle actuel.

Adapter à d'autres protocoles ou d'autres domaines

Dans le cadre de la rétro-ingénierie et d'autant plus pour le fingerprinting, appliquer les méthodes à un grand nombre de protocoles différents renforcerait leur validation. Cela aiderait à mettre en avant les limitations de chacune d'elles afin de les améliorer. Dans le cadre de la rétro-ingénierie, les problèmes relatifs à un protocole crypté ont d'ailleurs été clairement mentionnés. Par ailleurs, certains protocoles permettent d'encapsuler différentes commandes dans un seul message qui ne peut alors être assigné à un unique type contrairement à ce qui est fait actuellement.

Enfin, SVC n'étant qu'une technique de classification indépendante du domaine, évaluer si l'approche de détermination automatique des paramètres est applicable dans d'autre cas contribuerait à étendre son utilisation.

Vers un fingerprinting totalement automatique

Les approches de fingerprinting exposées dans cette thèse ne sont pour l'instant applicables que de manière expérimentale avec plusieurs outils. Par conséquent, l'utilisation courante d'une telle approche n'est pas faisable. Dans cet objectif, plusieurs points sont à considérer.

Tout d'abord, un premier axe consiste à optimiser au maximum les différents algorithmes utilisés rendant alors les méthodes plus rapides et donc applicables en temps réel. De plus, certaines approximations sont réalisables pour les différents algorithmes de classification. Par exemple, SVC oblige à tout recalculer depuis le début à chaque nouvel arbre à classer. Néanmoins, ce type de mise à jour « totale » ne pourrait être exécutée que périodiquement alors que la classification en temps réel appliquerait un algorithme beaucoup plus simple en assignant tout nouvel arbre au plus proche voisin. De même, les techniques basées sur les vecteurs supports permettent de définir des formes de clusters irrégulières pouvant être malgré tout réduites à quelques points seulement. Ce ne sont bien sûr que quelques pistes qui méritent un approfondissement.

Deuxièmement, une des méthodes proposée se base uniquement sur les types de messages pouvant être extraits en appliquant une étape de rétro-ingénierie telle que celle introduite dans le chapitre 4. Cette ultime étape est donc réalisable et mérite d'être achevée pour notamment voir l'impact des erreurs dues à la rétro-ingénierie sur le fingerprinting lui-même.

Enfin, la réalisation d'un logiciel global permettant de connaître en temps réel les différents équipements sur le réseau peut aussi se révéler intéressante technologiquement notamment si ce dernier intègre une fonctionnalité pour suggérer automatiquement des services dédiés aux utilisateurs selon leur équipement (service de support).

Passage à l'échelle

Bien que dans le cadre de la rétro-ingénierie et du fingerprinting, les travaux exposés s'efforcent de réduire au maximum la complexité, leur application sur des réseaux très hauts débits peut demeurer difficile en terme de passage à l'échelle de la même manière que pour les systèmes de détection d'intrusion. A l'instar de ces derniers, plusieurs techniques peuvent être mises en œuvre pour améliorer ce point : parallélisation des algorithmes, échantillonnage ou encore en utilisant les processeurs des cartes graphiques (GPU computing)...

Affinement et nouvelles méthodes de fingerprinting

Alors que beaucoup de méthodes de fingerprinting actuelles reposent sur la valeur de certains champs du message, cette thèse s'est efforcée de démontrer qu'il était possible d'extraire de l'information sémantique discriminante d'un autre type (comportement, structure syntaxique). Dans ce sens, elle ouvre la voie vers de nouvelles solutions pouvant par exemple combiner ces différentes approches. Au vue du nombre d'applications différentes s'exécutant sur une même machine, il est aussi possible de corréler le fingerprinting sur plusieurs protocoles. En effet, le fingerprinting d'équipement doit permettre d'identifier le nom mais aussi la version d'un logiciel. A titre d'exemple, certains logiciels sont dédiés certains systèmes d'exploitations et il est possible d'analyser l'utilisation simultanée de plusieurs logiciels pour voir si cela est habituel ou relève plutôt d'une erreur d'identification. Ainsi, le fingerprinting et le profiling des applications utilisés par un utilisateur pourrait être corrélés.

De plus, cette thèse s'est placée dans le cas le plus contraignant d'une approche passive mais les méthodes peuvent être adaptées au cas actif voire semi-passif pour augmenter la précision de l'identification des équipements. Dans le cas d'un fingerprinting semi-passif, un affinement consisterait à déterminer des intervalles de confiance pour réussir à correctement identifier quelles requêtes envoyer en priorité pour obtenir un intervalle plus petit.

Glossaire

ABNF : Augmented Backus-Naur Form
ADSL : Asymmetric Digital Subscriber Line
APTA : Augmented Prefix Tree Acceptor
ASCII : American Standard Code for Information Interchange

DDoS : Distributed Denial of Service
DNS : Domain Name System
DoS : Denial of Service

FTTH : Fiber To The Home

GCL : Guarded Command Language
GPU : Graphics Processing Unit

HTTP : HyperText Transfer Protocol

ICMP : Internet Control Message Protocol
IDS : Intrusion Detection System
IMAP : Internet Message Access Protocol
IP : Internet Protocol
IRC : Internet Relay Chat

MTA : Mail Transfert Agent

NAT : Network address translation

OSI : Open Systems Interconnection

P2P : Peer-to-Peer

RFC : Request For Comments
RPC : Remote Procedure Call
RTMP : Real Time Messaging Protocol
RTP : Real-time Transport Protocol

SIP : Session Initiation Protocol
SMB : Server Message Block
SMTP : Simple Mail Transfer Protocol
SVC : Support Vector Clustering

SVM : Support Vector Machine

TCP : Transmission Control Protocol

UDP : User Datagram Protocol
URI : Uniform Resource Identifier

VoIP : Voice over IP

Publications relatives

Journal francophone

[1] J. François, R. State, and O. Festor, “Botnets IRC et P2P pour une supervision à large échelle,” *Technique et Science Informatiques (TSI)*, vol. 28, pp. 433–458, 2009. [Online]. Available : http://hal.inria.fr/inria-00392573/en/

Conférences internationales

[1] J. François, H. Abdelnur, R. State, and O. Festor, “Automated behavioral fingerprinting,” in *International Symposium on Recent Advances in Intrusion Detection (RAID)*. St Malo, France : Springer, September 2009.

[2] J. François, R. State, and O. Festor, “Towards malware inspired management frameworks,” in *Network Operations and Management Symposium IEEE/IFIP Network Operations and Management Symposium (NOMS)*. Salvador Brésil : IEEE, April 2008. [Online]. Available : http://hal.inria.fr/inria-00310913/en/

[3] J. François, R. State, and O. Festor, “Botnet based scalable network management,” in *IFIP/IEEE Distributed Systems : Operations and Management (DSOM)*. San Jose, USA : Springer, October 2007.

[4] J. François, R. State, and O. Festor, “Malware models for network and service management,” in *(short paper) International Conference on Autonomous Infrastructure, Management and Security (AIMS)*. Oslo Norvège : Springer, June 2007. [Online]. Available : dx.doi.org/10.1007/978-3-540-72986-0_23http://hal.inria.fr/inria-00168415/en/

Conférence francophone

[1] J. François, R. State, and O. Festor, “Les botnets et la supervision à large échelle,” in *9èmes Journées Doctorales en Informatique et Réseaux - JDIR 2008*, Lille France, January 2008. [Online]. Available : http://hal.inria.fr/inria-00274977/en/

Rapports de recherche

[1] J. François, H. Abdelnur, R. State, and O. Festor, “Behavioral and Temporal Fingerprinting,” INRIA, Research Report RR-6995, 2009. [Online]. Available : http://hal.inria.fr/inria-00406482/en/

[2] J. François, H. Abdelnur, R. State, and O. Festor, “Advanced Fingerprinting For Inventory Management,” INRIA, Research Report RR-7044, 2009. [Online]. Available : http://hal.inria.fr/inria-00419766/en/

Bibliographie

La validité des URLs a été vérifiée le 30 septembre 2009.

1 Malwares, attaques, défenses

[1] “2006 spam treand report : Year of the zombies,” http://www.commtouch.com/documents/commtouch_2006_spam_trends_year_of_the_zombies.pdf.

[2] L. O. Murchu, “Silentbanker,” http://www.symantec.com/security_response/writeup.jsp?docid=2007-121718-1009-99.

[3] P. Henry, “Bigger, nastier botnets : Now cheaper to rent on the black market,” http://blog.lumension.com/?p=1676.

[4] “Vers/botnet prettypark,” http://www.symantec.com/security_response/writeup.jsp?docid=2000-121508-3334-99.

[5] K. Hayashi, “vers/botnet nugache,” http://www.symantec.com/security_response/writeup.jsp?docid=2006-043016-0900-99.

[6] “Enregistreurs de frappe, keylogger,” http://www.keylogger.org/.

[7] “Elk cloner,” http://www.skrenta.com/cloner/.

[8] “Eggdrop, robot irc,” http://www.eggheads.org/.

[9] “Dutch suspects hacked 1.5m pcs,” http://www.theage.com.au/news/breaking/dutch-suspects-hacked-15m-pcs/2005/10/21/1129775914143.html.

[10] D. Moore and C. Shannon, “The spread of the code-red worm (crv2),” http://www.caida.org/research/security/code-red/coderedv2_analysis.xml.

[11] T. Wilson, “Competition may be driving surge in botnets,” http://www.darkreading.com/security/management/showArticle.jhtml?articleID=208803799.

[12] J. Nazario, “Blackenergy ddos bot analysis,” http://atlas-public.ec2.arbor.net/docs/BlackEnergy+DDoS+Bot+Analysis.pdf.

[13] “backorifice,” http://www.cultdeadcow.com/tools/bo.html.

[14] A. Yamamoto, “Cheval de troie botnet aladinz,” http://www.symantec.com/security_response/writeup.jsp?docid=2004-011518-3235-99.

[15] B. B. Kang, E. Chan-Tin, C. P. Lee, J. Tyra, H. J. Kang, C. Nunnery, Z. Wadler, G. Sinclair, N. Hopper, D. Dagon, and Y. Kim, “Towards complete node enumeration in a peer-to-peer botnet,” in *International Symposium on Information, Computer, and Communications Security (ASIACCS)*. Sydney, Australia : ACM, March 2009.

[16] B. Wang, Z. Li, H. Tu, Z. Hu, and J. Hu, “Actively measuring bots in peer-to-peer networks,” in *International Conference on Networks Security, Wireless Communications and Trusted Computing (NSWCTC)*. Wuhan, China : IEEE Computer Society, April 2009.

[17] S. Chang, L. Zhang, Y. Guan, and T. Daniels, “A framework for p2p botnets,” in *International Conference on Communications and Mobile Computing (CMC)*. Kunming, Yunnan, China : IEEE Computer Society, January 2009.

[18] W. Wang, B. Fang, Z. Zhang, and C. Li, “A novel approach to detect irc-based botnets,” in *International Conference on Networks Security, Wireless Communications and Trusted Computing*. Wuhan, China : IEEE Computer Society, April 2009.

[19] Z. Li, A. Goyal, Y. Chen, and V. Paxson, “Automating analysis of large-scale botnet probing events,” in *International Symposium on Information, Computer, and Communications Security (ASIACCS)*. Sydney, Australia : ACM, March 2009.

[20] F. Leder and T. Werner, “Know your enemy : Containing conficker,” The Honeynet Project, USA, Tech. Rep., 2009.

[21] A. Pathak, F. Qian, Y. C. Hu, Z. M. Mao, and S. Ranjan, “Botnet spam campaigns can be long lasting : evidence, implications, and analysis,” in *International joint conference on Measurement and modeling of computer systems (SIGMETRICS)*. Seattle, WA, USA : ACM, June 2009.

[22] M. D. Bailey, E. Cooke, F. Jahanian, Y. Xu, and M. Karir, “A Survey of Botnet Technology and Defenses,” in *Cybersecurity Applications & Technology Conference For Homeland Security (CATCH)*. Waltham, MA, USA : IEEE Computer Society, March 2009.

[23] R. Hund, M. Hamann, and T. Holz, “Towards next-generation botnets,” in *European Conference on Computer Network Defense (EC2ND)*, Dublin, Ireland, December 2008.

[24] W. Yu, S. Chellappan, X. Wang, and D. Xuan, “Peer-to-peer system-based active worm attacks : Modeling, analysis and defense,” *Computer Communications*, vol. 31, no. 17, November 2008.

[25] E. Van Ruitenbeek and W. Sanders, “Modeling peer-to-peer botnets,” in *International Conference on Quantitative Evaluation of Systems (QEST)*, Williamsburg, USA, September 2008.

[26] C. R. Davis, S. Neville, J. M. Fernandez, J.-M. Robert, and J. Mchugh, “Structured peer-to-peer overlay networks : Ideal botnets command and control infrastructures ?” in *European Symposium on Research in Computer Security (ESORICS)*. Málaga, Spain : Springer-Verlag, October 2008.

[27] I. Castle and E. Buckley, “The automatic discovery, identification and measurement of botnets,” in *International Conference on Emerging Security Information, Systems and Technologies (SECURWARE)*. Cap Esterel, France : IEEE Computer Society, August 2008.

[28] J. Liang, N. Naoumov, and K. W. Ross, "Zhen li and qi liao and aaron striegel," in *Worshop on the Economics of Information Security (WEIS)*, Hanover, USA, June 2008.

[29] C. Kanich, C. Kreibich, K. Levchenko, B. Enright, G. M. Voelker, V. Paxson, and S. Savage, "Spamalytics : an empirical analysis of spam marketing conversion," in *Conference on Computer and communications security (CCS)*. Alexandria, Virginia, USA : ACM, October 2008.

[30] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer : clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Security Symposium (SS)*. San Jose, CA : USENIX Association, July 2008, pp. 139–154.

[31] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling, "Measurements and mitigation of peer-to-peer-based botnets : a case study on storm worm," in *Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEET)*. San Francisco, California : USENIX Association, Month 2008.

[32] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov, "Spamming botnets : signatures and characteristics," *SIGCOMM*, vol. 38, no. 4, pp. 171–182, August 2008.

[33] A. Networks, "Worldwide ISP security report," Arbor, Lexington, MA, USA, Tech. Rep., 2008.

[34] H. Abdelnur, O. Festor, and R. State, "KiF : A stateful SIP Fuzzer," in *International Conference on Principles, Systems and Applications of IP Telecommunications (IPTComm)*. New York City, USA : ACM SIGCOMM, July 2007.

[35] W. Cui, M. Peinado, H. Wang, and M. Locasto, "Shieldgen : Automatic data patch generation for unknown vulnerabilities with informed probing," in *IEEE Symposium on Security and Privacy (SP)*. Berkeley/Oakland, USA : IEEE, May 2007.

[36] R. Vogt, J. Aycock, and M. J. Jacobson, "Army of botnets," in *Network and Distributed System Security Symposium (NDSS)*. San Diego, USA : Internet Society, February 2007.

[37] D. Dagon, G. Gu, C. Lee, and W. Lee, "A taxonomy of botnet structures," in *Annual Computer Security Applications Conference (ACSAC)*. Miami Beach, USA : IEEE Computer Society, December 2007.

[38] J. Goebel and T. Holz, "Rishi : identify bot contaminated hosts by irc nickname evaluation," in *First Workshop on Hot Topics in Understanding Botnets (HotBots)*. Cambridge, MA : USENIX Association, April 2007.

[39] A. Karasaridis, B. Rexroad, and D. Hoeflin, "Wide-scale botnet detection and characterization," in *First Workshop on Hot Topics in Understanding Botnets (HotBots)*. Cambridge, MA : USENIX Association, April 2007.

[40] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "Bothunter : detecting malware infection through ids-driven dialog correlation," in *USENIX Security Symposium (SS)*. Boston, MA : USENIX Association, August 2007.

[41] C. Schiller and J. Binkley, *Botnets : The Killer Web Applications*. Syngress Publishing, Fevruary 2007.

[42] P. Wang, S. Sparks, and C. C. Zou, "An advanced hybrid peer-to-peer botnet," in *First Workshop on Hot Topics in Understanding Botnets (HotBots)*. Cambridge, MA : USENIX Association, April 2007.

[43] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon, "Peer-to-peer botnets : Overview and case study," in *First Workshop on Hot Topics in Understanding Botnets (HotBots)*. Cambridge, MA : USENIX Association, April 2007.

[44] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "My botnet is bigger than yours (maybe, better than yours) : why size estimates remain challenging," in *First Workshop on Hot Topics in Understanding Botnets (HotBots)*. Cambridge, MA : USENIX Association, April 2007.

[45] P. Barford and V. Yegneswaran, *An inside look at Botnets*. Springer, March 2007, ch. 8.

[46] R. Ford and S. Gordon, "Cent, five cent, ten cent, dollar : hitting botnets where it really hurts," in *Workshop on New security paradigms (NSPW)*. Germany : ACM, September 2006.

[47] Z. Trabelsi and K. Shuaib, "Man in the middle intrusion detection," in *IEEE Global Telecommunications Conference (GLOBECOM)*. San Francisco, USA : IEEE, November 2006.

[48] K. Ramachandran and B. Sikdar, "Modeling malware propagation in gnutella type peer-to-peer networks." Rhodes Island, Greece : IEEE, April 2006.

[49] J. Liang, N. Naoumov, and K. W. Ross, "The index poisoning attack in p2p file sharing systems," in *International Conference on Computer Communications (INFOCOM)*. Barcelona, Spain : IEEE, April 2006.

[50] A. Ramachandran, N. Feamster, and D. Dagon, "Revealing botnet membership using dnsbl counter-intelligence," in *Conference on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*. San Jose, CA : USENIX Association, July 2006.

[51] E. Alata, V. Nicomette, M. Ka ?niche, M. Dacier, and M. Herrb, "Lessons learned from the deployment of a high-interaction honeypot," in *European Dependable Computing Conference (EDCC)*. Cambria, Portugal : IEEE Computer Society, October 2006.

[52] J. R. Binkley and S. Singh, "An algorithm for anomaly-based botnet detection," in *Conference on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*. San Jose, CA : USENIX Association, July 2006.

[53] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in *Conference on Internet measurement (IMC)*. Rio de Janeriro, Brazil : ACM, October 2006.

[54] D. Dagon, C. Zou, and W. Lee, "Modeling botnet propagation using time zones," in *Network and Distributed System Security Symposium (NDSS)*. San Diego, USA : Internet Society, February 2006.

[55] P. Oehlert, "Violating assumptions with fuzzing," in *Security & Privacy (SP)*. Berkeley/Oakland, USA : IEEE, March-April 2005.

[56] C. Del Grosso, G. Antoniol, M. Di Penta, P. Galinier, and E. Merlo, "Improving network applications security : a new heuristic to generate stress testing data," in *Conference on Genetic and evolutionary computation (GECCO)*. Washington DC, USA : ACM, June 2005.

[57] M. Mannan and P. C. van Oorschot, "On instant messaging worms, analysis and countermeasures," in *Workshop on Rapid malcode (WORM)*. Fairfax, VA, USA : ACM, November 2005.

[58] E. Cooke, F. Jahanian, and D. Mcpherson, "The zombie roundup : Understanding, detecting, and disrupting botnets," in *Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*. Cambridge, USA : USENIX Association, July 2005.

[59] J. Canavan, "The evolution of malicious irc bots," in *Virus Bulletin Conference (VB)*, Dublin, Ireland, October 2005.

[60] P. Szor, *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional, February 2005.

[61] E. FILIOL, *Computer viruses : from theory to applications*. Springer, June 2005.

[62] P. Bâcher, T. Holz, M. Kötter, and G. Wicherski, "Know your enemy : Tracking botnets. http://www.honeynet.org/papers/bots/," 2005.

[63] S. Schechter, J. Jung, and A. W. Berger, "Fast Detection of Scanning Worm Infections," in *Symposium on Recent Advances in Intrusion Detection (RAID)*. French Riviera, France : springer, September 2004.

[64] S. Staniford, D. Moore, V. Paxson, and N. Weaver, "The top speed of flash worms," in *Workshop on Rapid malcode (WORM)*. Washington DC, USA : ACM Press, October 2004.

[65] L. McLaughlin, "Bot software spreads, causes new worries," *Distributed Systems Online*, vol. 5, no. 6, June 2004.

[66] C. Douligeris and A. Mitrokotsa, "Ddos attacks and defense mechanisms : classification and state-of-the-art," *Comput. Netw.*, vol. 44, no. 5, pp. 643–666, October 2004.

[67] I. Arce and E. Levy, "An analysis of the slapper worm," *IEEE Security and Privacy*, vol. 1, no. 1, pp. 82–87, January 2003.

[68] C. Zou, W. Gong, and D. Towsley, "Worm propagation modeling and analysis under dynamic quarantine defense," in *Workshop on Rapid malcode (WORM)*. Washington DC, USA : ACM Press, October 2003.

[69] J. Nazario, *Defense and Detection Strategies against Internet Worms*. Artech House, Inc., October 2003.

[70] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the slammer worm," *IEEE Security and Privacy*, vol. 1, no. 4, pp. 33–39, July 2003.

[71] C. Krügel, T. Toth, and E. Kirda, "Service specific anomaly detection for network intrusion detection," in *Symposium on Applied computing (SAC)*. Madrid, Spain : ACM, March 2002.

[72] J. R. Douceur, "The sybil attack," in *International Workshop on Peer-to-Peer Systems (IPTPS*. Cambridge, USA : Springer-Verlag, March 2002.

[73] C. Zou, W. Gong, and D. Towsley, "Code red worm propagation modeling and analysis," in *Conference on Computer and Communications Security (CCS)*, Washington, USA, November 2002.

[74] S. Staniford, V. Paxson, and N. Weaver, "How to own the internet in your spare time," in *Security Symposium*. San Francisco, USA : USENIX Association, August 2002.

[75] M. Bishop, “Analysis of the iloveyou worm,” Department of Computer Science, University of California, USA, Tech. Rep., 2000.

[76] M. Roesch, “Snort : Lightweight intrusion detection for networks.” in *Systems Administration Conference (LISA)*. Seattle, USA : USENIX, November 1999.

[77] E. H. Spafford, “The internet worm incident,” in *European Software Engineering Conference (ESEC)*. Coventry, UK : Springer-Verlag, September 1989.

[78] L. Adleman, “An abstract theory of computer viruses (invited talk),” in *Advances in Cryptology (CRYPTO)*. Santa Barbara, California, United States : Springer-Verlag New York, Inc., August 1988.

[79] F. Cohen, “Computer viruses : Theory and experiments,” *Computers & Security*, vol. 6, no. 1, pp. 22–35, February 1987.

[80] J. P. Anderson, “Computer security technology planning study,” US Air Force Electronic Systems Division, USA, Tech. Rep., 1972.

[81] W. O. Kermack and A. G. McKendrick, “A contribution to the mathematical theory of epidemics,” *Royal Society of London Proceedings Series A*, vol. 115, pp. 700–721, August 1927.

2 Modélisation des botnets

[1] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi, “Efficient broadcast in structured p2p networks.” in *International Workshop on Peer-to-Peer Systems (IPTPS)*. Berkeley, USA : Springer-Verlag, February 2003.

[2] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, “Making gnutella-like p2p systems scalable,” in *Conference on Applications, technologies, architectures, and protocols for computer communications*. Karlsruhe, Germany : ACM SIGCOMM, August 2003.

[3] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, “Chord : A scalable Peer-To-Peer lookup service for internet applications,” in *SIGCOMM Conference*. San Diego, USA : ACM, August 2001.

[4] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, “Wide-area cooperative storage with CFS,” in *Symposium on Operating Systems Principles (SOSP)*. Chateau Lake Louise, Banff, Canada : ACM, October 2001.

[5] M. E. J. Newman, S. H. Strogatz, and D. J. Watts, “Random graphs with arbitrary degree distribution and their applications,” Santa Fe Institute, Working Papers, July 2000. [Online]. Available : http://ideas.repec.org/p/wop/safiwp/00-07-042.html

[6] A. L. Barabasi and R. Albert, “Emergence of scaling in random networks,” *Science*, vol. 286, no. 5439, pp. 509–512, October 1999.

[7] D. J. Watts and S. H. Strogatz, “Collective dynamics of /‘small-world/’ networks,” *Nature*, vol. 393, no. 6684, pp. 440–442, June 1998.

[8] V. N. Sachkov, *Probabilistic methods in combinatorial analysis*. Cambridge University Press, May 1997, ch. 6 - Random Graphs and Random Mappings.

[9] P. Erdos and A. Renyi, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, pp. 17–61, 1960.

3 Rétro-ingénierie

[1] "Wine," http://www.winehq.org/.

[2] "Tcpreplay," http://tcpreplay.synfin.net/trac/.

[3] "Stunnix c++ obfuscator," http://www.stunnix.com/prod/cxxo/overview.shtml.

[4] E. Morphy, "Skype protocol reverse engineered," http://www.technewsworld.com/story/dprmytjNX3SP9V/Chinese-Engineers-Reportedly-Crack-Skypes-Code.xhtml?wlc=1246868321.

[5] A. Tridgell, "How samba was written," http ://samba.org/ftp/tridge/misc/french_cafe.txt.

[6] M. Beddoe, "Protocol informatics," http ://www.4tphi.net/ awalters/PI/PI.html.

[7] "Ida : Interactive disassembler," http://www.hex-rays.com/idapro/.

[8] W. Cui, "Discoverer : Automatic protocol reverse engineering from network traces," pp. 199–212.

[9] P. Milani omparetti, G. Wondracek, C. Kruegel, and E. Kirda, "Prospex : protocol specification extraction," in *Symposium on Security and Privacy*. Oakland, USA : IEEE, May 2009.

[10] Z. Lin, X. Jiang, D. Xu, and X. Zhang, "Automatic protocol format reverse engineering through conectect-aware monitored execution," in *Symposium on Network and Distributed System Security (NDSS)*. San Diego, USA : Internet Society, February 2008.

[11] W. Cui, M. Peinado, K. Chen, H. J. Wang, and L. Irun-Briz, "Tupni : automatic reverse engineering of input formats," in *Conference on Computer and communications security (CCS)*. Alexandria, Virginia, USA : ACM, October 2008.

[12] G. Wondracek, P. M. Comparetti, C. Kruegel, and E. Kirda, "Automatic network protocol analysis," in *Symposium on Network and Distributed System Security (NDSS)*. San Diego, USA : Internet Society, February 2008.

[13] Z. Wang, X. Jian, W. Cui, and X. Wang, "Reformat : Automatic reverse engineering of encrypted messages," North Carolina State University, USA, Tech. Rep., 2008.

[14] N. Borisov, D. J. Brumley, and H. J. Wang, "A generic application-level protocol analyzer and its language," in *Symposium on Network and Distributed System Security (NDSS)*. San Diego, USA : Internet Society, March 2007.

[15] J. Caballero, H. Yin, Z. Liang, and D. Song, "Polyglot : automatic extraction of protocol message format using dynamic binary analysis," in *Conference on Computer and communications security (CCS)*. Alexandria, Virginia, USA : ACM, October 2007.

[16] M. Shevertalov and S. Mancoridis, “A reverse engineering tool for extracting protocols of networked applications.” Vancouver, Canada : Springer, Oct. 2007.

[17] M. N. Gagnon, S. Taylor, and A. K. Ghosh, “Software protection through anti-debugging,” *Security and Privacy*, vol. 5, no. 3, pp. 82–84, May 2007.

[18] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker, “Unexpected means of protocol inference.” in *Internet Measurement Conference (IMC)*. Rio de Janeiro, Brazil : ACM SIGCOMM, October 2006.

[19] K. Gopalratnam, S. Basu, J. Dunagan, and H. J. Wang, “Automatically extracting fields from unknown network protocols,” June 2006.

[20] J. Newsome, D. Brumley, J. Franklin, and D. Song, “Replayer : automatic protocol replay by binary analysis,” in *Conference on Computer and communications security (CCS)*. Alexandria, Virginia, USA : ACM, October 2006.

[21] W. Cui, V. Paxson, N. Weaver, and R. H. Katz, “Protocol-independent adaptive replay of application dialog,” in *Symposium on Network and Distributed System Security (NDSS)*. San Diego, USA : Internet Society, February 2006.

[22] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, “ACAS : automated construction of application signatures.” in *MineNet*. Philadelphia, USA : ACM SIGCOMM, August 2005.

[23] C. Leita, K. Mermoud, and M. Dacier, “Scriptgen : an automated script generation tool for honeyd,” *Annual Computer Security Applications Conference (ACSAC)*, December 2005.

[24] M. Bugalho and A. L. Oliveira, “Inference of regular languages using state merging algorithms with search,” *Pattern Recognition*, vol. 38, no. 9, pp. 1457 – 1467, March 2005.

[25] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. P. Vadhan, and K. Yang, “On the (im)possibility of obfuscating programs,” in *Annual International Cryptology Conference on Advances in Cryptology (CRYPTO)*. Santa Barbara, USA : Springer-Verlag, August 2001.

[26] P. Garcia, A. Cano, and J. Ruiz, “A comparative study of two algorithms for automata identification,” in *International Colloquium on Grammatical Inference (ICGI)*. Lisbon, Portugal : Springer-Verlag, September 2000.

[27] D. Lee and K. Sabnani, “Reverse-engineering of communication protocols,” in *International Conference on Network Protocols (ICNP)*. San Francisco, California : IEEE, October 1993.

[28] E. J. Chikofsky and J. H. Cross II, “Reverse engineering and design recovery : A taxonomy,” *IEEE Software*, vol. 7, no. 1, pp. 13–17, January 1990.

[29] R. L. Rivest and R. E. Schapire, “Inference of finite automata using homing sequences,” in *Symposium on Theory of computing (STOC)*. Seattle, Washington, United States : ACM, May 1989.

[30] D. Angluin, “Learning regular sets from queries and counterexamples,” *Information and computation*, vol. 75, no. 2, pp. 87–106, September 1987.

[31] E. M. Gold, “Complexity of automaton identification from given data,” *Information and Control*, vol. 37, no. 3, pp. 302–320, 1978.

[32] E. W. Dijkstra, “Guarded commands, nondeterminacy and formal derivation of programs,” *Communications of the ACM*, vol. 18, no. 8, pp. 453–457, August 1975.

4 Fingerprinting

[1] "Xprobe," http://xprobe.sourceforge.net/.

[2] "P0f," http://lcamtuf.coredump.cx/p0f.shtml.

[3] "Internet assigned numbers authority," http://www.iana.org/.

[4] "Httpprint," http://www.net-square.com/httprint/httprint_paper.html.

[5] "Amap," http://freeworld.thc.org/thc-amap/.

[6] G. F. Lyon, *Nmap Network Scanning : The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure Press, January 2009.

[7] J. Burroni and C. Sarraute, "Using neural networks to improve classical operating system fingerprinting techniques," *Electronic Journal of SADIO*, vol. 8, no. 1, pp. 509–512, March 2008.

[8] H. Abdelnur, R. State, and O. Festor, "Advanced Network Fingerprinting," in *Recent Advances in Intrusion Detection (RAID)*. Boston United States : Springer, September 2008.

[9] L. C. C. Desmond, C. C. Yuan, T. C. Pheng, and R. S. Lee, "Identifying unique devices through wireless fingerprinting," in *Conference on Wireless network security (WiSec)*. Alexandria, VA, USA : ACM, March 2008.

[10] N.-F. Huang, G.-Y. Jai, and H.-C. Chao, "Early identifying application traffic with application characteristics," in *International Conference on Communications (IEEE)*. Beijing, China : IEEE, May 2008.

[11] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified : myths, caveats, and the best practices," in *CoNEXT Conference*. Madrid, Spain : ACM, December 2008.

[12] P. Auffret, "SinFP, unification de la prise d'empreinte active et passive des systèmes d'exploitation," in *Symposium sur la Sécurité des Technologies de l'Information et des Communications (SSTIC)*. Rennes, France : École Supérieure et d'Application des Transmissions, June 2008.

[13] S. Bratus, C. Cornelius, D. Kotz, and D. Peebles, "Active behavioral fingerprinting of wireless devices," in *Conference on Wireless network security (WiSec*. Alexandria, VA, USA : ACM, March 2008.

[14] J. Caballero, S. Venkataraman, P. Poosankam, M. G. Kang, D. Song, and A. Blum, "FiG : Automatic Fingerprint Generation," in *Symposium on Network and Distributed System Security (NDSS)*. San Diego, USA : Internet Society, March 2007.

[15] C. V. Wright, L. Ballard, F. Monrose, and G. M. Masson, "Language identification of encrypted voip traffic : Alejandra y roberto or alice and bob ?" in *Security Symposium (SS)*. Boston, MA : USENIX Association, August 2007.

[16] R. Fink, "A statistical approach to remote physical device fingerprinting," in *Military Communications Conference, MilCom*. Orlando, Florida : IEEE, October 2007.

[17] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, "Traffic classification through simple statistical fingerprinting," *Computer Communication Review*, vol. 37, no. 1, pp. 5–16, January 2007.

[18] Y. Liu, W. Li, and Y.-C. Li, "Network traffic classification using k-means clustering." Iowa City, USA : IEEE Computer Society, August 2007.

[19] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, "Offline/realtime traffic classification using semi-supervised learning," *Performance Evaluation*, vol. 64, no. 9-12, pp. 1194–1213, October 2007.

[20] G. Szabo, I. Szabo, and D. Orincsay, "Accurate traffic classification," in *International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. Helsinki, Finland : IEEE, June 2007.

[21] L. G. Greenwald and T. J. Thomas, "Toward undetected operating system fingerprinting," in *Workshop on Offensive Technologies (WOOT)*. Boston, MA : USENIX Association, August 2007.

[22] G. Shu and D. Lee, "Network protocol system fingerprinting - a formal approach," in *International Conference on Computer Communications (INFOCOM)*. Barcelona, Spain : IEEE, April 2006.

[23] H. Yan, K. Sripanidkulchai, H. Zhang, Z. yin Shae, and D. Saha, "Incorporating Active Fingerprinting into SPIT Prevention Systems," in *Annual VoIP Security Workshop*, Berlin, Germany, June 2006.

[24] H. Scholz, "SIP Stack Fingerprinting and Stack Difference Attacks," in *Black Hat Briefings*, Las Vegas, USA, July 2006.

[25] N. Williams, S. Zander, and G. Armitage, "A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification," *Computer Communication Review*, vol. 36, no. 5, pp. 5–16, October 2006.

[26] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *Computer Communication Review*, vol. 36, no. 2, pp. 23–26, April 2006.

[27] L. Bernaille, R. Teixeira, and K. Salamatian, "Early application identification," in *CoNEXT conference*. Lisboa, Portugal : ACM, December 2006.

[28] J. Erman, M. Arlitt, and A. Mahanti, "Traffic classification using clustering algorithms," in *MineNet*. Pisa, Italy : ACM SIGCOMM, September 2006.

[29] W. Li, D. Zhang, and J. Yang, "Remote os fingerprinting using bp neural network," in *International Symposium on Neural Networks (ISNN)*. Chongqing, China : Springer, 367-372 2005.

[30] T. Kohno, A. Broido, and K. Claffy, "Remote physical device fingerprinting," *Transactions on Dependable and Secure Computing (TDSC)*, vol. 2, no. 2, pp. 93–108, April 2005.

[31] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *International conference on Measurement and modeling of computer systems (SIGMETRICS)*. Banff, Alberta, Canada : ACM, June 2005.

[32] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning," in *Conference on Local Computer Networks (LCN)*. Zürich, Switzerland : IEEE Computer Society, October 2005.

[33] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, "Blinc : multilevel traffic classification in the dark," in *Conference on Applications, technologies, architectures, and protocols for computer communications*. Philadelphia, Pennsylvania, USA : ACM SIGCOMM, August 2005.

[34] A. W. Moore and K. Papagiannaki, "Toward the accurate identification of network applications," in *Passive and Active Network Measurement (PAM)*. Boston, USA : Springer, March 2005.

[35] Z. J. Wang, M. Wu, W. Trappe, and K. J. R. Liu, "Group-oriented fingerprinting for multimedia forensics," *EURASIP Journal on Applied Signal Processing*, vol. 2004, no. 14, pp. 2153–2173, 2004.

[36] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield, "Class-of-service mapping for qos : a statistical signature-based approach to ip traffic classification," in *Internet measurement Conference (IMC)*. Taormina, Sicily, Italy : ACM SIGCOMM, October 2004.

[37] R. Beverly, "A robust classifier for passive tcp/ip fingerprinting," in *Passive and Active Network Measurement (PAM)*. Antibes Juan-les-Pins, France : Springer, April 2004.

[38] S. Sen, O. Spatscheck, and D. Wang, "Accurate, scalable in-network identification of p2p traffic using application signatures," in *International conference on World Wide Web (WWW)*. New York, NY, USA : ACM, May 2004.

[39] S. M. Bellovin, "A technique for counting natted hosts," in *SIGCOMM Workshop on Internet measurment (IMW)*. Marseille, France : ACM SIGCOMM, November 2002.

[40] V. Paxson, "Automated packet trace analysis of tcp implementations," in *Conference on Applications, technologies, architectures, and protocols for computer communication*. Cannes, France : ACM SIGCOMM, September 1997.

[41] Douglas Comer and John C. Lin, "Probing TCP Implementations," in *USENIX Summer*. Boston, Massachusetts : USENIX Association, June 1994.

5 Classification

[1] C.-C. Chang and C.-J. Lin, *LIBSVM : a library for support vector machines*, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[2] H. Dahmouni, S. Vaton, and D. Rossé, "A markovian signature-based approach to ip traffic classification," in *Workshop on Mining network data (MineNet)*. San Diego, California, USA : ACM, June 2007.

[3] A. Moschitti, "Making tree kernels practical for natural language learning," in *International Conference of European Chapter of the Association for Computational Linguistics (EACL)*, Trento, Italy, April 2006.

[4] P. Berkhin, "A survey of clustering data mining techniques," in *Grouping Multidimensional Data*. Springer, November 2006, pp. 25–71.

[5] M. Dutta, A. K. Mahanta, and A. K. Pujari, "Qrock : a quick version of the rock algorithm for clustering of categorical data," *Pattern Recognition Letter*, vol. 26, no. 15, pp. 2364–2373, November 2005.

[6] A. Torsello, D. Hidovic-Rowe, and M. Pelillo, "Polynomial-time metrics for attributed trees," *Transactions on Pattern Analysis and Machine Intelligence (TPMAI)*, vol. 27, no. 7, pp. 1087–1099, July 2005.

[7] *Support Vector Machines : Theory and Applications*, ser. Studies in Fuzziness and Soft Computing. Springer, August 2005, vol. 177.

[8] D. Buttler, "A Short Survey of Document Structure Similarity Algorithms," in *International Conference on Internet Computing (ICOMP)*, Quebec, Canada, June 2005.

[9] P. Bille, "A survey on tree edit distance and related problems," *Theoretical Computer Science*, vol. 337, no. 1-3, pp. 217–239, June 2005.

[10] R. Debnath, N. Takahide, and H. Takahashi, "A decision based one-against-one method for multi-class support vector machine," *Pattern Analysis & Applications.*, vol. 7, no. 2, pp. 164–175, July 2004.

[11] S. Salvador and P. Chan, "Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms," in *ICTAI '04 : Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence*. Washington, DC, USA : IEEE Computer Society, 2004, pp. 576–584.

[12] L. Yu and H. Liu, "Feature selection for high-dimensional data : A fast correlation-based filter solution," in *International Conference on Machine Learning (ICML)*. Washington : AAAI Press, August 2003.

[13] L. R. Welch, "Hidden markov models and the baum-welch algorithm," *Information Theory Society Newsletter*, vol. 53, no. 4, December 2003.

[14] M. Collins and N. Duffy, "New ranking algorithms for parsing and tagging : Kernels over discrete structures, and the voted perceptron," in *Association for Computational Linguistics Conference (ACL)*, Pennsylvania, USA, July 2002.

[15] S. Vishwanathan and A. Smola, "Fast kernels on strings and trees." in *Neural Information Processing Systems (NIPS)*, Vancouver, Canada, December 2002.

[16] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *Transactions on Neural Networks (TNN)*, vol. 13, no. 2, pp. 415–425, March 2002.

[17] I. Rish, "An empirical study of the naive bayes classifier," in *International Joint Conference on Artificial Intelligence (IJCAI) workshop on "Empirical Methods in AI"*, Seattle, USA, August 2001.

[18] A. Ben-hur, D. Horn, H. T. Siegelmann, and V. Vapnik, "Support vector clustering," *Journal of Machine Learning Research*, vol. 2, pp. 125–137, December 2001.

[19] A. Ben-Hur, D. Horn, H. Siegelmann, and V. Vapnik, "A support vector clustering method," in *Internation Conference on Pattern Recognition (ICPR)*. Barcelona, Spain : IEEE, September 2000.

[20] N. Cristianini and J. Shawe-Taylor, *An introduction to support Vector Machines : and other kernel-based learning methods*. Cambridge University Press, March 2000.

[21] P. Baldi, S. Brunak, Y. Chauvin, C. A. Andersen, and H. Nielsen, “Assessing the accuracy of prediction algorithms for classification : an overview,” *Bioinformatics*, vol. 16, no. 5, pp. 412–24, February 2000.

[22] S. Guha, R. Rastogi, and K. Shim, “Rock : A robust clustering algorithm for categorical attributes,” in *International Conference on Data Engineering (ICDE)*. Sydney, Australia : IEEE Computer Society, March 1999, p. 512.

[23] Y. Freund and R. E. Schapire, “A short introduction to boosting,” in *International Joint Conference on Artificial Intelligence (IJCAI)*. Stockohlm, Sweden : Morgan Kaufmann, 1999, pp. 1401–1406.

[24] K. Zhang, R. Statman, and D. Shasha, “On the editing distance between unordered labeled trees,” *Information Processing Letters*, vol. 42, no. 3, pp. 133–139, May 1992.

[25] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, “Basic local alignment search tool,” *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, October 1990.

[26] R. Fletcher, *Practical methods of optimization ; (2nd ed.)*. New York, NY, USA : Wiley-Interscience, 1987.

[27] W. H. Day and H. Edelsbrunner, “Efficient algorithms for agglomerative hierarchical clustering methods,” *Journal of Classification*, vol. 1, no. 1, pp. 7–24, December 1984.

[28] T. F. Smith and M. S. Waterman, “Identification of common molecular subsequences,” *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, March 1981.

[29] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, March 1970.

[30] P. Jaccard, “The distribution of the flora in the alpine zone,” *New Phytologist*, vol. 11, no. 2, pp. 37 – 50, 1912.

6 Supervision des réseaux et services

[1] SNMP and Research, “The mid-level manager,” http ://www.snmp.com/products/mlm.html.

[2] R. Badonnel, R. State, I. Chrisment, and O. Festor, “A management platform for tracking cyber predators in peer-to-peer networks,” in *International Conference on Internet Monitoring and Protection (ICIMP)*. San Jose, California : IEEE Computer Society, July 2007.

[3] R. State and O. Festor, “Malware : a future framework for device, network and service management,” *Journal in Computer Virology*, vol. 3, no. 1, pp. 51–60, March 2007.

[4] R. Ahmed and R. Boutaba, “Distributed Pattern Matching : a Key to Flexible Efficient P2P Search,” *Journal on Selected Areas in Communications*, vol. 25, no. 1, pp. 73–83, January 2007.

[5] R. Badonnel, R. State, and O. Festor, “Probabilistic management of ad-hoc networks,” in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*. Vancouver, Canada : IEEE, April 2006.

[6] M. S. Artigas, P. G. López, and A. F. Gómez-Skarmeta, “Deca : A hierarchical framework for decentralized aggregation in dhts,” in *IFIP/IEEE International Workshop on Distributed Systems : Operations and Management (DSOM)*. 246-257 : Springer, October 2006.

[7] K.-S. Lim and R. Stadler, “Real-time views of network traffic using decentralized management,” in *IFIP/IEEE International Symposium on Integrated Network Management*. Nice, France : IEEE, May 2005.

[8] K.-S. Lim and R. Stadler, “A navigation pattern for scalable internet management,” in *IFIP/IEEE International Symposium on Integrated Network Management*. Seattle, USA : IEEE, May 2001.

[9] B. Schwartz, A. W. Jackson, W. T. Strayer, W. Zhou, R. D. Rockwell, and C. Partbridge, “Smart packets : applying active networks to network management,” *Transactions on Computer Systems*, vol. 18, no. 1, pp. 67–88, February 2000.

[10] H. Ohno and A. Shimizu, “Improved network management using nmw (network management worm) system,” in *Annual Conference of the Internet Society (INET)*. Honololu, USA : ISOC, June 1995.

[11] G. Goldszmidt and Y. Yemini, “Distributed management by delegation,” in *International Conference on Distributed Computing Systems (ICDCS)*. Vancouver, Canada : IEEE Computer Society, May 1995.

7 RFC

[1] J. Klensin, “Simple Mail Transfer Protocol,” RFC 5321 (Draft Standard), Oct. 2008. [Online]. Available : http://www.ietf.org/rfc/rfc5321.txt

[2] D. Crocker and P. Overell, “Augmented BNF for Syntax Specifications : ABNF,” RFC 5234 (Standard), Jan. 2008. [Online]. Available : http://www.ietf.org/rfc/rfc5234.txt

[3] T. Ylonen and C. Lonvick, “The Secure Shell (SSH) Protocol Architecture,” RFC 4251 (Proposed Standard), Jan. 2006. [Online]. Available : http://www.ietf.org/rfc/rfc4251.txt

[4] M. Crispin, “Internet Message Access Protocol - Version 4rev1,” RFC 3501 (Proposed Standard), Mar. 2003, updated by RFCs 4466, 4469, 4551, 5032, 5182. [Online]. Available : http://www.ietf.org/rfc/rfc3501.txt

[5] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, “RTP : A Transport Protocol for Real-Time Applications,” RFC 3550 (Standard), Jul. 2003. [Online]. Available : http://www.ietf.org/rfc/rfc3550.txt

[6] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, “SIP : Session Initiation Protocol,” RFC 3261 (Proposed Standard), Jun. 2002, updated by RFCs 3265, 3853, 4320, 4916, 5393. [Online]. Available : http://www.ietf.org/rfc/rfc3261.txt

[7] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext Transfer Protocol – HTTP/1.1,” RFC 2616 (Draft Standard), Jun. 1999, updated by RFC 2817. [Online]. Available : http://www.ietf.org/rfc/rfc2616.txt

[8] J. Oikarinen and D. Reed, “Internet Relay Chat Protocol,” RFC 1459 (Experimental), May 1993, updated by RFCs 2810, 2811, 2812, 2813. [Online]. Available : http://www.ietf.org/rfc/rfc1459.txt

[9] S. Microsystems, “RPC : Remote Procedure Call Protocol specification : Version 2,” RFC 1057 (Informational), Jun. 1988. [Online]. Available : http://www.ietf.org/rfc/rfc1057.txt

[10] P. Mockapetris, “Domain names - concepts and facilities,” RFC 1034 (Standard), Nov. 1987, updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592.

[11] J. Postel and J. Reynolds, “File Transfer Protocol,” RFC 959 (Standard), Oct. 1985, updated by RFCs 2228, 2640, 2773, 3659. [Online]. Available : http://www.ietf.org/rfc/rfc959.txt

[12] J. Postel, “Internet Control Message Protocol,” RFC 792 (Standard), Sep. 1981, updated by RFCs 950, 4884. [Online]. Available : http://www.ietf.org/rfc/rfc792.txt

8 Divers

[1] “Unicode,” http://www.unicode.org/.

[2] “tcpdump,” http://www.tcpdump.org/.

[3] “Skype interdit de séjour dans l’enseignement supérieur et la recherche,” http://www.zdnet.fr/actualites/internet/0,39020774,39267873,00.htm.

[4] “Skype,” http://www.skype.com/.

[5] “Python programming language – official website,” http://www.python.org/.

[6] “Overnet,” http://www.zeropaid.com/overnet/.

[7] “Oscar unofficial specifications,” http://oilcan.org/oscar/.

[8] “Microsoft msn unofficial specifications,” http://www.hypothetic.org/docs/msn/.

[9] “Maxima, a computer algebra system,” http://maxima.sourceforge.net/.

[10] “Internet information services,” http://www.iis.net/.

[11] “Adobe flash,” http://www.adobe.com/products/flashplayer/.

[12] J. F. Kurose and K. W. Ross, *Computer Networking : A Top-Down Approach (4th Edition)*. Addison Wesley, April 2007.

[13] J. B. Horrigan, “Broadband adoption at home in the united states : Growing but slowing,” in *Telecommunications Policy Research Conference (TPRC)*, Arlington, USA, September 2005.

[14] J. Chow, B. Pfaff, T. Garfinkel, K. Christopher, and M. Rosenblum, “Understanding data lifetime via whole system simulation,” in *Security Symposium*. San Diego, USA : USENIX, August 2004.

[15] P. Maymounkov and D. Mazières, “Kademlia : A peer-to-peer information system based on the xor metric,” in *International Workshop on Peer-to-Peer Systems (IPTPS)*. Cambridge, USA : Springer, March 2002.

[16] A. M. Turing, “On computable numbers, with an application to the entscheidungsproblem,” *London Mathematical Society*, vol. 2, no. 42, pp. 230–265, 1937.

A

Partitionnement non supervisé de données

Sommaire

A.1 Rappel des notations 223
A.2 Clustering hiérarchique ascendant 223
A.3 Support vector Clustering 225
A.3.1 Apprentissage 225
A.3.2 Découverte des clusters 228

Cette annexe présente en détails les deux méthodes non supervisées de partitionnement de données qui sont employées dans le cadre de la rétro-ingénierie de protocole dont fait l'objet le chapitre 5.

A.1 Rappel des notations

Le partitionnement de données ou clustering vise à construire des clusters à partir d'un ensemble de points $X = \{x_i, i = 1 \dots n\}$. De plus, le nombre de clusters ou classes n'est pas connu à priori. Étant donné que chaque point appartient à une classe donnée $C = \{c_i, i = 1 \dots m\}$, $T_i = \{x_{i1}, \dots, x_{ip}\}$ est l'ensemble des points de X appartenant à la classe i. Il est important de noter que chaque x est indexé par un unique indice (ik) et non deux, avec $i \in 1 \dots m$ représentant la classe correspondante et k variant entre un et le nombre d'éléments de cette classe. Par conséquent il faut préciser que $\forall\ T_i$, $\forall\ x \in T_i\ x \in X$, c'est-à-dire $\forall x_{kp}, 1 < kp < n$.

Un algorithme de clustering s'assimile à une fonction $\psi : X \to K$ où $K = k_i, i = 1, \dots, t$ c'est-à-dire une fonction qui associe une classe k_i à chaque point x_i. L'ensemble K est différent de l'ensemble C et leurs cardinalités peuvent notamment divergées selon l'efficacité de l'algorithme utilisé.

A.2 Clustering hiérarchique ascendant

Le clustering hiérarchique ascendant [27] est une méthode de partitionnement très connue. Son principe réside dans la notion de distance entre deux points telles que celles décrites dans la section précédente. En effet, la première étape construit un cluster pour chaque message individuel. Ensuite, les plus proches sont regroupés entre eux à chaque étape jusqu'à regrouper l'ensemble des points dans un même cluster. Ce type de méthode est dit hiérarchique car un arbre binaire est créé, où la racine est le

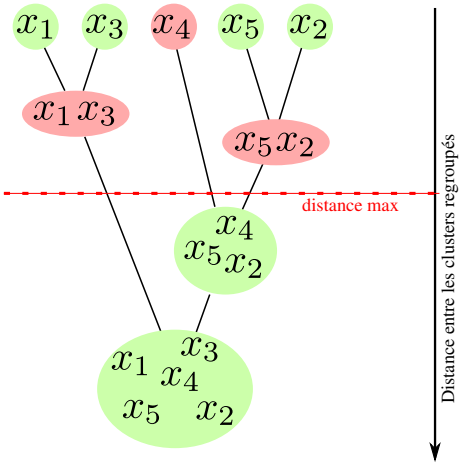


FIGURE A.1 – Partitionnement hiérarchique – les clusters en rouge sont les clusters finaux si une distance maximal représentée par la ligne rouge est spécifiée.

cluster le plus général (avec tous les messages) et les feuilles sont les clusters les plus simples (messages individuels). Chaque nœud de l'arbre représente donc le cluster composé des messages contenus dans les sous-branches comme l'illustre la figure A.1. Un tel arbre s'appelle un dendogramme où la hauteur d'un cluster équivaut à la distance minimale pour regrouper ces sous-branches. Généralement, les rassemblements sont opérés jusqu'à ce que la distance inter-cluster soit supérieure à un seuil. Dans ce cas, le dendogramme se limite aux sous-arbres sous cette distance comme signalé sur la figure A.1.

L'algorithme 7 est la mise en œuvre de cette technique. Il prend trois paramètres : l'ensemble X de points à partitionner, le seuil maximal τ pour regrouper deux clusters et d une fonction de distance entre deux clusters. L'initialisation commence donc par créer un cluster individuel pour chaque message (ligne 3) puis calcule la matrice de distances entre ces derniers 6. L'algorithme itère alors tant qu'il existe une distance inférieure à τ (ligne 8. Deux clusters sont regroupés de la façon suivante :

- les points d'un des deux sont ajoutés à ceux de l'autre (ligne 9) et l'autre est supprimé ;
- les distances associées au cluster supprimé le sont elles aussi ;
- les distances associées au cluster préservé sont mises à jour (boucle de la ligne 13).

L'évaluation de la distance entre deux clusters est un paramètre essentiel de l'algorithme. Comme celles introduites dans la section précédente sont définies pour des points et non des clusters (des ensembles de points), elles doivent être adaptées. Étant donné une telle distance d_{points}, plusieurs approches existent pour en déduire la distance $d(c_1, c_2)$ entre les clusters c_1 et c_2 dont les plus répandues sont :

- le saut minimal est la distance la plus petite entre deux points de chaque cluster :

$$d(c_1, c_2) = \min_{x_i inc_1, y_j \in c_2} (d_{points}(x_i, y_j))$$

- chaque cluster c_i est représenté par son barycentre $bar(c_i)$ sur lequel on applique la fonction de distance :

$$d(c_1, c_2) = (d_{points}(bar(c_1), bar(c_2))$$

La première, souvent dénommée la méthode des proches voisins, aide à identifier des clusters de formes arbitraires. Par exemple, des clusters linéaires comme sur la figure A.2(a) peuvent être reconnus alors que la méthode du barycentre aura tendance à regrouper de clusters concentrés autour d'un point à l'instar de la figure A.2(b). L'application de l'une ou l'autre des méthodes doit donc être mûrement réfléchie selon le cas d'utilisation.

Algorithme 7 clust_hierarchique(X, τ, d)

1: $dim(M)$ retourne la dimension de la matrice carrée M
2: $résultat = \{\}$
3: **pour** $i = 1, \ldots, dim(M)$ **faire**
4: $\quad résultat_i = \{x_i\}$
5: **fin pour**
6: remplir M tel que $M_{ij} = d(résultat_i, résultat_j)$

7: chercher la valeur minimale $min = M_{kt}$
8: **tantque** $min < \tau$ **faire**
9: $\quad résultat_k = résultat_k \cap résultat_t$
10: $\quad$ supprimer $résultat_t$
11: $\quad$ enlever la ligne M_t
12: $\quad$ enlever la colonne M_t
13: $\quad$ **pour** $i = 1, \ldots, dim(M)$ **faire**
14: $\quad\quad M_{ki} = d(résultat_k, résultat_i)$
15: $\quad\quad M_{ik} = M_{ki}$
16: $\quad$ **fin pour**
17: $\quad$ chercher la valeur minimale $min = M_{kt}$
18: **fin tantque**
19: retourner *résultat*

A.3 Support vector Clustering

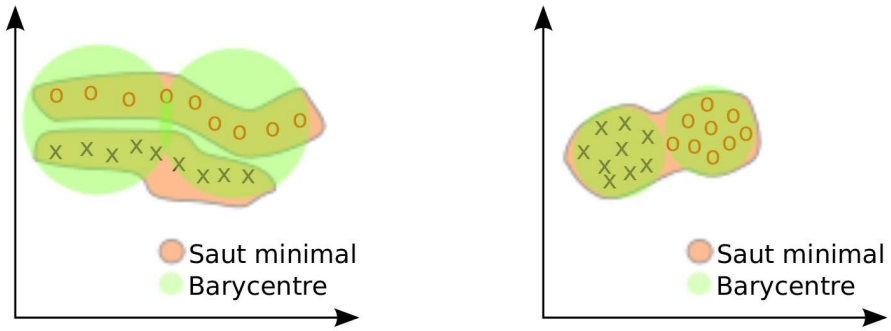
L'intérêt de algorithmes basés sur les machines à vecteurs supports est la séparation de points de données non séparables dans l'espace d'origine (figure A.3(a)). Ceux-ci sont donc initialement au sein de clusters de formes arbitraires difficilement séparables. L'astuce des SVM projette ces points dans un espace de plus grande dimension (figure A.3(b)), appelé espace de re-description, dans lequel les points seront facilement dissociables. Intuitivement, un espace plus grand caractérisera les données plus précisément pour mieux les séparer. Dans le cadre de SVC, les contours des clusters de l'espace d'origine se réduisent à une hypersphère dans l'espace de plus grande dimension. Son objectif est donc de trouver cette sphère (figure A.3(c)) puis de la repasser dans l'espace d'origine pour découvrir les clusters qu'elle forme (figure A.3(d)).

A.3.1 Apprentissage

Chaque point x_i de X est projeté dans un nouvel espace dimensionnel via l'application de la fonction Φ. L'hypersphère contenant les points projettés est spécifiée par son centre a et son rayon R. Par conséquent, cette sphère se définit à partir des points d'origine comme :

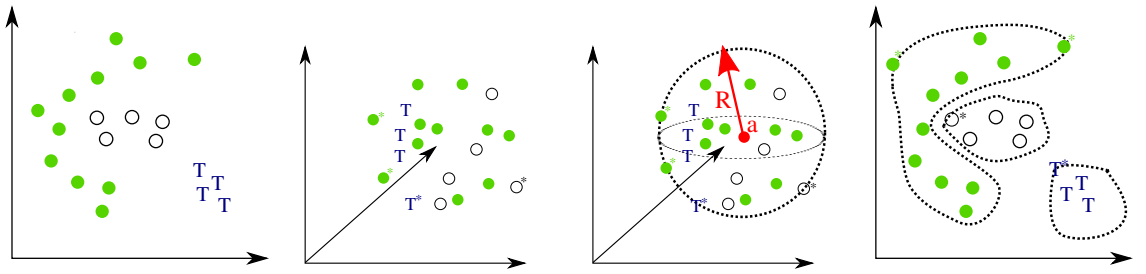
$$\|\Phi(x_i) - a\|^2 \leq R^2 \; \forall i \tag{A.1}$$

Cependant, certains points peuvent tout de même ne pas être facilement intégrés dans la sphère à moindre coût. En fait, SVC introduit des variables d'ajustement autorisant quelques points à ne pas répondre entièrement aux conditions imposées. Ce choix évite ainsi de biaiser les résultats à cause de



(a) Clusters linéaires (b) Clusters circulaires

FIGURE A.2 – Clustering hiérarchique selon deux types de distances inter-clusters



(a) Points initiaux (b) Projection dans l'espace de re-description (c) Recherche de l'hypersphère englobante (d) Projection de l'hypersphère dans l'espace d'origine

FIGURE A.3 – Les étapes de SVC (Les étoiles symbolisent les vecteurs supports)

quelques points de données non facilement identifiables voire totalement erronés. Par conséquent, l'hypersphère est plus formellement spécifiée par :

$$\|\Phi(x_i) - a\|^2 \leq R^2 + \xi_i \ \forall i \tag{A.2}$$

où les ξ_i sont les variables d'ajustement.

Étant donné l'objectif de trouver la sphère de rayon minimal R tout en respectant la contrainte précédente, les auteurs introduisent les multiplicateurs de Lagrange β_i et μ_i :

$$L = R^2 - \sum_i (R^2 + \xi_i - \|\Phi(x_i) - a\|^2)\beta_i - \sum_i \xi_i \mu_i + C \sum_i \xi_i \tag{A.3}$$

Le terme $C\sum_i \xi_i$ représente quant à lui la pénalité induite par les variables d'ajustement. De cette façon cette formulation prend en compte le fait d'avoir des valeurs d'ajustement les plus petites possibles dont l'impact est paramétrable grâce à la constante C. Une telle formulation transforme le problème d'optimisation en une recherche des valeurs telles que la dérivée du Lagrangien soit nulle pour chacune des variables du problème. En dérivant l'équation (A.3) respectivement selon R, a et ξ_i et en prenant la

valeur en zéro correspondante, le résultat est respectivement :

$$\begin{aligned} \sum_j \beta_j &= 1 \\ a = \sum_j \beta_j \Phi(x_j) &= 1 \\ \beta_j &= C - \mu_j \end{aligned} \tag{A.4}$$

Cependant, le problème d'optimisation en question est sous contrainte non linéaire d'inégalité nécessitant l'ajout des conditions KKT (Karush-Kuhn-Tucker) qui furent étudiées dans le cadre des SVM par Fletcher [26] :

$$\begin{aligned} \xi_j \mu_j &= 0 \\ (R^2 + \xi_j - ||\phi(x_j) - a||^2)\beta_j &= 0 \\ \beta_j &= C - \mu_j \end{aligned} \tag{A.5}$$

Le problème est alors complètement défini, est qualifié de primal et dévoile trois types de points :

- les vecteurs supports qui sont les points exactement à la surface de la sphère. Plus particulièrement ce sont ceux tels que $0 < \beta_i < C$ et ils forment l'ensemble SV ;
- les points en dehors de la sphère, c'est-à-dire tels que $\beta_i = C$;
- les autres poins à l'intérieur de la sphère.

Le problème est mutable en une version duale :

$$W = \sum_i \Phi(x_i)^2 \beta_i - \sum_{i,j} \beta_i \beta_j K(x_i, x_j) \tag{A.6}$$

avec comme contraintes :

$$0 \leq \beta_i \leq C \; ; \; \sum_i \beta_i = 1 \tag{A.7}$$

$K(x_i, x_j)$ est une fonction noyau respectant le théorème de Mercer [20]. Son introduction est l'autre point clé de toutes les méthodes basées sur les SVM car elle évite la définition de la fonction de transfomation Φ et se cantonne à une fonction applicable à deux points tel que le noyau Gaussien paramétré par q :

$$K(x_i, x_j) = e^{-q\|x_i - x_j\|^2} \tag{A.8}$$

En anglais, cette astuce se nomme *kernel trick*. Le problème d'optimisation se résume donc à trouver les valeur de β_i permettant de maximiser W grâce à des méthodes de résolution de problèmes de programmation quadratique. La distance d'un point x quelconque par rapport à la sphère est alors calculable :

$$R^2(x) = K(x,x) - 2\sum_j \beta_j K(x_j, x) + \sum_{i,j} \beta_i \beta_j K(x_i, x_j) \tag{A.9}$$

Le rayon de la sphère se calcule donc grâce à cette formule appliquée sur les vecteurs supports étant des points sur la surface de l'hypersphère par définition.

$$R = \{R(x_i)\}, x_i \in SV$$

La phase dite d'apprentissage se termine ainsi mais, malgré ce qualificatif, cette méthode reste totalement non supervisée et ne demande aucun jeu d'apprentissage ni le nombre de clusters à priori.

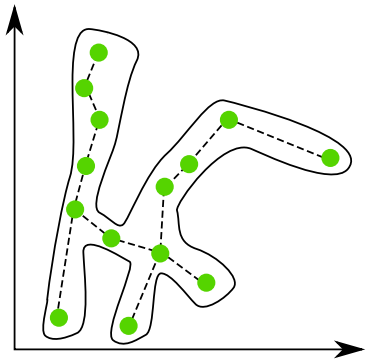


FIGURE A.4 – Forme aléatoire d'un cluster, les lignes en pointillés montrent que pour tout point du cluster, il existe un segment linéaire le reliant à un autre sans sortir de la forme du cluster

A.3.2 Découverte des clusters

La sphère découverte et re-projetée dans l'espace d'origine délimite les frontières des clusters. La méthode proposée initialement dans [18] s'appuie sur une approche géométrique mentionnant que deux points font partie du même cluster si l'ensemble des points intermédiaires (sur le segment entre les deux points dans l'espace d'origine) est aussi dans le même cluster, c'est-à-dire qu'ils se situent dans l'hyper-sphère. En effet, malgré une forme totalement aléatoire des clusters, celle-ci suit les contours formés par les points extérieurs. En conséquence, tout point d'une classe est forcément lié par un « segment » à un autre sans sortir du cluster (la figure A.4). Une matrice d'adjacence est ainsi construite :

$$A_{ij} = \begin{cases} 1 & si\ \forall y\ sur\ le\ segment\ [x_i, x_j], R(y) < R \\ 0 & else \end{cases}$$

Les composantes connexes du graphe décrites par cette matrice concordent alors avec les clusters.

B

Machines à vecteurs de support multi-classes

Sommaire

B.1 Apprentissage 229
B.2 Utilisation 231

Cette annexe détaille la classification multi-classe basée sur les machines à vecteurs de support utilisées dans le cadre du fingerprinting supervisé d'équipement dans le chapitre 6 duquel est repris l'ensemble des notations :

- l'ensemble de tests composé de N arbres : $T = \{t_1, \ldots, t_N\}$,
- le jeu d'apprentissage défini par un ensemble de M arbres $L = \{l_1, \ldots, l_M\}$,
- l'ensemble des types d'équipement : $D = \{d_1, \ldots, d_K\}$.

Pour rappel, les étapes clé de la méthode sont :

- apprentissage :
  - projection des points dans un espace de plus grande dimension,
  - découverte des hyperplans séparateurs.
- tests :
  - chercher de quel côté de chaque hyperplan le point à tester se trouve,
  - déterminer quelle est la classe (type) la plus probable.

B.1 Apprentissage

Premièrement, les points de l'espace initial c'est-à-dire les arbres de l'ensemble d'apprentissage sont projetés dans l'espace de re-description via une fonction $\varphi(l_i)$. Pour chaque paire de types d'équipements $< d_l, d_k >$, l'ensemble des arbres correspondant est formé :

$$\begin{aligned} L_l &= \{l_i \in L | real(l_i) = d_l\} \\ L_k &= \{l_i \in L | real(l_i) = d_k\} \end{aligned} \tag{B.1}$$

L'hyperplan séparateur H_{l-k} spécifié par $w^l k$ et $b^l k$ doit résulter en un partitionnement quasi parfait

de $L_l \cup L_k$ en introduisant certains arbres mal positionnés via des variables d'ajustement $\xi_{l_i}^{lk}$:

$$\begin{aligned} \forall l_i \in \{L_l \cup L_k\} \\ \langle \varphi(l_i) \cdot w^{lk} \rangle + b^{lk} \geq 1 - \xi_{l_i}^{lk},\ si\ l_i \in L_l \\ \langle \varphi(l_i) \cdot w^{lk} \rangle + b^{lk} \leq -1 + \xi_{l_i}^{lk},\ si\ l_i \in L_k \end{aligned} \tag{B.2}$$

Les variables d'ajustement contribuent à découvrir un hyperplan séparateur même si les classes ne sont pas entièrement séparables. Par exemple, si un point 0 se trouve au milieu des points X sur la figure 6.1(c), aucun hyperplan séparateur n'est envisageable. La formulation précédente induit une normalisation de manière à ce que les points soient à une distance $+1$ ou -1 selon le côté de l'hyperplan (sans prendre en compte les variables d'ajustement). La marge associée vaut alors $\frac{1}{||w^{lk}||}$. Le problème se résume alors à maximiser ou plutôt minimiser son inverse en prenant en compte la marge des deux côtés d'où le facteur $\frac{1}{2}$ dans la formule ci-dessous :

$$\min_{w^{lk}, b^{lk}, \xi_{l_i}^{lk}} \frac{1}{2}||w^{lk}||^2 + C \sum_{l_i \in \{L_l \cup L_k\}} \xi_{t_i}^{lk} \tag{B.3}$$

Ce problème d'optimisation ainsi que les contraintes précédentes justifie une fois de plus l'introduction des multiplicateurs de Lagrange :

$$L = \frac{1}{2}||w^{lk}||^2 - \sum_{l_i \in L_l} \alpha_{l_i}^{lk}(\langle \varphi(l_i) \cdot w^{lk} \rangle + b^{lk} - 1 + \xi_{l_i}^{lk}) + \sum_{l_i \in L_k} \alpha_{l_i}^{lk}(\langle \varphi(l_i) \cdot w^{lk} \rangle + b^{lk} + 1 - \xi_{l_i}^{lk}) \tag{B.4}$$

Après dérivation et calcul en zéro selon les différentes variables w^{lk}, b^{lk} et $\xi_{l_i}^{lk}$, le problème d'optimisation dual suivant est obtenu :

$$max \sum_{l_i \in \{T_l \cup T_k\}} \alpha_{l_i}^{lk} - \frac{1}{2} \sum_{\substack{l_i \in \{T_l \cup T_k\} \\ l_j \in \{T_l \cup T_k\}}} \alpha_{l_i}^{lk} \alpha_{l_j}^{lk} \rho_{l_i}^{lk} \rho_{l_j}^{lk} \langle \varphi(l_i).\varphi(l_j) \rangle \tag{B.5}$$

avec :

$$\rho_{l_i}^{lk} = \begin{cases} 1 & \forall l_i \in L_l \\ -1 & \forall l_i \in L_k \end{cases} \tag{B.6}$$

D'autres contraintes complètent le problème :

$$\begin{aligned} \sum_{l_i \in \{T_l \cup T_k\}} \alpha_{l_i}^{lk} \rho_{l_i}^{lk} = 0 \\ 0 \leq \alpha_{l_i}^{lk} \leq C,\ l_i \in \{T_l \cup T_k\} \end{aligned} \tag{B.7}$$

Comme pour SVC dans le chapitre précédent, l'usage d'une fonction noyau contourne le problème de la définition de la fonction φ :

$$K(l_i, l_j) = \langle\ \varphi(l_i).\varphi_(l_j)\ \rangle \tag{B.8}$$

Le produit scalaire de la formule B.5 peut ainsi être remplacé.

Les points étant exactement sur la surface de la marge, nommés vecteurs supports et formant l'ensemble SV^{lk} sont caractérisés, par un multiplicateur de Lagrange non nul :

$$\forall l_i \in SV \alpha_{l_i}^{lk} \neq 0$$

Le scalaire b^{lk} est directement calculable à partir de ces derniers de la façon suivante :

$$b^{lk} = \frac{1}{|SV^{lk}|} \sum_{l_i \in SV^{lk}} (\rho_{l_i}^{lk} - \sum_{l_j \in \{T_l \cup T_k\}} \alpha_{l_i}^{lk} \rho_{l_j}^{lk} K(l_j, l_i)) \tag{B.9}$$

B.2 Utilisation

Lors de l'utilisation, un nouveau point l_x doit être rangé dans l'une ou l'autre classe. L'équation B.2 indique explicitement que la classe d'un point ou, dans cette étude, l'équipement d'un arbre dépend de sa position par rapport à l'hyperplan. Plus précisément, la fonction de décision suivante s'appuie uniquement sur les vecteurs supports car pour les autres points l_i $\alpha_{l_i}^{lk} = 0$:

$$f_{lk}(l_x) = \sum_{l_i \in \{T_l \cup T_k\}} \alpha_{l_i}^{lk} \rho_{l_i} K(l_i, l_x) + b^{lk} \tag{B.10}$$

Par conséquent, le signe de la valeur retournée par cette fonction renseigne sur la position de l_x par rapport à l'hyperplan d'où la fonction $classe_{lk}(l_x)$ assignant une classe au point l_x :

$$classe_{lk}(l_x) = \begin{cases} d_l & si\ f_{lk}(l_x) \geq 0 \\ d_k & sinon \end{cases} \tag{B.11}$$

Comme un hyperplan sépare de deux classes, la classification nécessite autant d'hyperplans qu'il existe de paires de classes d'où le calcul de $\frac{K(K-1)}{2}$ fonctions de décision car K est le nombre d'équipements distincts et donc de classes.

Finalement, la fonction Ω construite pendant la phase d'apprentissage et caractéristique du fingerprinting supervisé teste l'appartenance d'un arbre par rapport à chaque hyperplan pour en déduire le type le plus probable :

$$\Omega(l_x) = arg \max_{d_i \in D} \left(\sum_{classe^{lk}(l_x) = d_i} 1 \right) \tag{B.12}$$

Résumé

La popularité des réseaux informatiques et d'Internet s'accompagne d'un essor des applications communicantes et de la multiplication des protocoles dont le fonctionnement est plus ou moins compliqué, ce qui implique également des performances différentes en termes de robustesse. Un premier objectif de cette thèse est d'approfondir plus en détails la robustesse de protocoles s'illustrant par d'extraordinaires performances empiriques tels que les botnets. Différents protocoles employés par les botnets sont donc modélisés dans cette thèse.

Par ailleurs, l'essor et la diversité des protocoles s'accompagnent d'un manque de spécification volontaire ou non que la rétro-ingénierie tente de retrouver. Une première phase essentielle est notamment de découvrir les types de messages. La technique mise en œuvre dans cette étude s'appuie sur les machines à vecteurs de supports tout en ayant au préalable spécifié de nouvelles représentations des messages dont la complexité de calcul est très réduite par rapport aux autres techniques existantes.

Enfin, il existe généralement un grand nombre d'applications distinctes pour un même protocole et identifier précisément le logiciel ou le type d'équipement utilisé (marque, version) est un atout essentiel dans plusieurs domaines tels que la supervision ou la sécurité des réseaux. S'appuyant uniquement sur les types de messages, le comportement d'un équipement, c'est-à-dire la manière dont il interagit avec les autres, est une information très avantageuse lorsqu'elle est couplée avec les délais entre les messages. Enfin, la grammaire d'un protocole connu permet de construire les arbres syntaxiques des messages, dont le contenu et la structure sémantiquement riche, avaient peu été étudiés jusqu'à maintenant dans le cadre de l'identification des équipements.

Mots-clés: botnet, robustesse, supervision, sécurité, fingerprinting, arbre comportemental, arbre syntaxique, machines à vecteurs supports, clustering, rétro-ingénierie des protocoles.

Abstract

The growth of computer networks like the Internet entailed a huge increase of networked applications and the apparition of multiple, various protocols. Their functioning complexity is very variable implying diverse performances. The first objective of this PhD thesis is to evaluate precisely the robustness of those networked applications, which are known to be very efficient and seem scalable, like for instance, the botnets. Hence, several botnets protocols are imitated.

Furthermore, protocol reverse engineering has skyrocketed because many protocols are not always well documented. In this domain, the first necessary step is to discover the message types and this work introduces a novel technique based on support vector machines and new simple message representations in order to reduce the complexity.

Finally, there are many distinct applications for a single protocol which can be identified thanks to device fingerprinting techniques whose the domain of application is related to security and network management. The first technique proposed in this PhD thesis can work with the previous contribution about reverse engineering because the devices could be identified only based on the types of messages exchanged which are aggregated into a temporal behavioral tree including message delays. Besides, the

syntactic tree structure of a message is also a good discriminative feature to distinguish the different devices but was very little considered until now.

Keywords: botnet, robustness, network management, security, fingerprinting, behavioral tree, syntactic tree, support vector machines, clustering, protocol reverse engineering