MASTER EN SCIENCES INFORMATIQUES

Exploitation de traces DNS pour l'identification de trafic malicieux à l'aide de méthodes de machine learning

Baudin, Régis

Award date:
2019

Awarding institution:
Universite de Namur

Université de Namur
Faculté d'informatique
Année académique 2018-2019

Exploitation de traces DNS pour l'identification de trafic malicieux à l'aide de méthodes de machine learning

Régis Baudin

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)
Jean-Noël Colin

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Résumé

D'après un rapport sur la cybersécurité publié en février 2019 par la compagnie Cisco Systems spécialisée dans les réseaux, 58% des menaces dans les réseaux proviennent des *botnets*. Cette insécurité met en danger l'intégrité des modes de communication présente sur bon nombre d'objets connectés et par le fait même du réseau internet. Ce réseau doit, entre autres, son fonctionnement au DNS qui se veut inoffensif au sein des réseaux mondiaux. De fait, son architecture et ses propriétés se révèlent être une aubaine pour les *botnets*. En s'appuyant sur le protocole DNS, les méthodes utilisées par les cybercriminels rendent leur trafic malicieux bien souvent indétectable.

Dans le cadre de ce mémoire, une description détaillée et très didactique du protocole DNS est réalisée en mettant en lumière ses forces mais aussi ses faiblesses. L'analyse des études scientifiques relatives au trafic malicieux, nous permet de mettre en chantier un outil de détection. À l'aide de méthodes de *machine learning*, nous mettons en œuvre cet outil nommé *pipeline*. Notre approche est consolidée par des *blacklists* et des *whitelists* qui servent de références pour distinguer différents types de trafics.

À l'aide d'une capture DNS, nous plongeons dans le réseau de l'UNamur, où nous appliquons notre *pipeline* en vue d'identifier tout trafic malicieux. Notre volonté est de travailler en toute transparence et de partager ces travaux avec un communauté plus large.

Mots clés

Protocole DNS, sécurité, trafic malicieux, détection, malware, botnet, machine learning, whitelist, blacklist, pipeline.

Abstract

According to a cyber security report published in February 2019 by Cisco Systems a market leader in network technology, 58% of all network threats come from *botnets*. The architecture of DNS was conceived in the very early days of the internet, and the creators did not foresee that it would ever be used maliciously. However, due to its ubiquitous usage on the global internet and lack of integral security, the protocol is an ideal medium for *botnets*. Further, what makes this method even more attractive for use by *botnets* is that malicious traffic which leverage the DNS protocol are almost undetectable.

This thesis provides a detailed and informative explanation of the DNS protocol's function and role in the internet, highlighting both its strengths and weaknesses. Following the analysis of previous scientific studies related to malicious traffic, a detection tool which uses machine learning was developed. The tool, named *pipeline*, employs whitelists and blacklists to distinguish differents types of traffic.

Using packet captures from the network at UNamur, *pipeline* was used to analyse DNS traffic with the aim of detecting malicious traffic. This paper is intended to aid in the understanding of how this traffic can be detected, and to support the research carried out by the wider security community.

Keywords

DNS protocol, security, malicious traffic, detection, malware, botnet, machine learning, whitelist, blacklist, pipeline.

Avant-propos

Ce mémoire représente certes l'aboutissement de mon projet en vue de l'obtention du grade de Master en Sciences Informatiques, mais il est aussi le témoin de mon intérêt pour un espace toujours plus informatisé et connecté qui est de fait la proie de hackers rarement bien intentionnés.
Les pirates en tout genre qui voient le jour sur la toile se doivent d'être mis en lumière et combattus ; pour ce faire, ils doivent avant tout être détectés. C'est dans cette réalité que le présent travail trouve tout son sens.
Ce travail ardu et complexe a pu se faire grâce au soutien et avec l'aide de plusieurs personnes que je tiens ici à nommer et à remercier.

Tout d'abord, je souhaite remercier ma compagne, mes beaux-parents et mes parents pour leur soutien, leur intérêt sincère et leurs relectures qui m'ont mené, entre autres, à l'élaboration de ce beau projet.

Ensuite, je remercie mon promoteur, Monsieur Colin, pour ses conseils et les discussions qui m'ont inspiré tout au long du développement de ma recherche. Il m'a guidé en me questionnant et en me proposant des pistes d'évolution. Je le remercie également pour ses relectures et le coaching qu'il m'a apporté pour appréhender au mieux le thème choisi.

Je tiens également à exprimer ma reconnaissance envers Geoffroy Herbin, Nassim Benoussaid, Laurent Miny pour leurs relectures, leurs réflexions, le partage sur les aspects plus techniques.

Je tiens aussi à remercier mon responsable hiérarchique et mes collègues, qui m'ont soutenu tout au long de ces années d'étude.

Merci aux professeurs de la faculté d'informatique de l'UNamur de m'avoir transmis leur savoir et leur passion durant cette année de Master. De fait, ils m'ont fait découvrir des facettes d'un monde informatique que je soupçonnais mais dont j'ignorais les horizons

Finalement, je vous remercie, cher lecteur, pour l'attention que vous portez à ce mémoire.

Table des matières

Acronymes **17**

Introduction **19**

1 Le protocole DNS **21**
1.1 Système de noms de domaine . . . 21
1.2 Architecture . . . 22
1.2.1 Hiérarchie . . . 22
1.2.2 Modèle client-serveur . . . 23
1.2.3 Résolution . . . 24
1.2.4 Couche Transport . . . 25
1.3 Propriétés . . . 25
1.3.1 Noms de Domaine . . . 25
1.3.2 Zones . . . 26
1.3.3 Resource record . . . 27
1.3.4 in-addr.arpa . . . 28
1.3.5 Cache . . . 29
1.4 Messages . . . 29
1.4.1 Structure . . . 29
1.4.2 En-tête . . . 30
1.4.3 Question . . . 31
1.4.4 Réponse . . . 31
1.5 Services DNS . . . 33
1.5.1 Round-Robin DNS - RRDNS . . . 33
1.5.2 Content Delivery Network - CDN . . . 33

2 DNS & Sécurité **35**
2.1 Malware . . . 35
2.2 Les Botnets . . . 36
2.2.1 Définition . . . 37
2.2.2 Usages du botnet . . . 37
2.2.3 Composants . . . 37
2.2.4 Topologies . . . 39
2.2.5 Usages du DNS . . . 39
2.3 Exploitation du DNS . . . 41
2.4 Attaques serveurs DNS . . . 42
2.5 Attaques utilisateurs . . . 43
2.6 Limitations du protocole DNS . . . 43
2.6.1 DNS Security Extensions - DNSSEC . . . 44
2.6.2 DNS over HTTPS - DoH . . . 45

2.6.3 DNS over TLS - DoT . . . 45

3 Études existantes **47**
3.1 Définition du trafic malicieux . . . 47
3.2 Défis de la détection du trafic malicieux . . . 47
3.3 Le passive DNS - pDNS . . . 48
3.4 Les études . . . 49

4 Outils **57**
4.1 Programmation . . . 57
4.2 Serveur . . . 58
4.3 IDS Bro . . . 58
4.4 Datasets . . . 58
4.4.1 Datasets non-labélisés . . . 58
4.4.2 Datasets labélisés . . . 58
4.4.3 Gestion de datasets en ligne . . . 60
4.5 Synthèse . . . 61

5 Le machine learning **63**
5.1 Supervisé et non supervisé . . . 63
5.2 Algorithmes supervisés de classification . . . 64
5.3 Évaluation . . . 65
5.3.1 Apprentissage et test . . . 65
5.3.2 Matrice de confusion . . . 65
5.3.3 Métriques . . . 66
5.4 Améliorations . . . 67
5.4.1 Cross Validation . . . 67
5.4.2 Grid search . . . 68
5.4.3 Équilibrage de données . . . 68

6 Expérimentations **69**
6.1 Objectif . . . 69
6.2 Pipeline . . . 70
6.3 Informations du dataset de l'UNamur . . . 71
6.4 Partie 1 : Labélisation à l'aide de listes de références . . . 73
6.4.1 Labélisation d'un maximum de requêtes . . . 73
6.4.2 Conclusion . . . 76
6.4.3 Labélisation de requêtes avec un minimum de FP et FN . . . 77
6.4.4 Discussion . . . 80
6.4.5 Conclusion . . . 81
6.5 Partie 2 : Classification à l'aide de machine learning . . . 82
6.5.1 Le pipeline . . . 82
6.5.2 Labélisation de datasets de références . . . 82
6.5.3 Création de features . . . 83
6.5.4 Extraction et préparation des données d'apprentissage . . . 85
6.5.5 Validation des modèles . . . 86
6.5.6 Résultats des tests des modèles . . . 87
6.5.7 Détail du modèle Random Forest . . . 87
6.5.8 Analyse du dataset de l'UNamur . . . 90
6.5.9 Conclusion . . . 92

Conclusion **93**

Annexes **101**
A.1 Bro détails . . . 101
A.2 Code - labéliser la capture . . . 102
A.3 Extrait du fichier unlabelled_data.csv . . . 103
A.4 Visualisation d'une requête DNS avec wireshark . . . 104
A.5 Machine virtuelle . . . 106
A.6 Code - création, mélange et équilibrage des données . . . 107
A.7 Code - validation de modèle . . . 107
A.8 Code - matrice de confusion . . . 108
A.9 Code - recherche des meilleurs hyperparamètres . . . 108
A.10 Code - classification dataset UNamur . . . 109

Table des figures

1.1 Niveaux des domaines dans l'espace de noms . . . 23
1.2 Fonctionnement DNS - Résolution de noms de domaine . . . 24
1.3 Domaines et noeuds . . . 26
1.4 Zones . . . 27
1.5 Configuration de serveur de noms pour la zone ROOT [37] . . . 28
1.6 Format d'un message [38] . . . 30
1.7 Structure de l'en-tête d'un message DNS [38] . . . 30
1.8 Structure d'une question [38] . . . 31
1.9 Structure d'un *resource record* [38] . . . 31
1.10 Illustration d'une réponse DNS . . . 32

2.1 Cisco Cybersecurity February 2019 Threat Report [12] . . . 35
2.2 Top 10 des malwares représentant 52% du trafic total de malware pour le mois de Janvier 2019 [22] . . . 36
2.3 Topologie standard d'un Botnet [31] . . . 38
2.4 Différence entre le *Single-Flux* et le *Double-Flux* [32] . . . 40
2.5 Limitations en terme d'intégrité et de privatisation des données du protocole DNS. 44

3.1 Exemple de caractéristiques ou *features* . . . 48
3.2 Liste de features utilisées dans *Exposure* [45] . . . 50
3.3 Pipeline d'*Exposure* [45] . . . 51
3.4 Labélisation de jeux de données par *Exposure* . . . 51
3.5 Features du protocole DNS [49] . . . 55

4.1 Liste de blacklists . . . 60
4.2 Résultat d'une résolution pour le nom de domaine malicieux *pagaldaily.com* . . . 61

5.1 Matrice de confusion utilisée dans les expériences de ce projet [27] . . . 66
5.2 Mécanisme du cross validation [55] . . . 68

6.1 Pipeline général . . . 70
6.2 Aperçu du dataset *1384844* de 8Gb . . . 72
6.3 Liste des blacklists . . . 74
6.4 Intersection des noms de domaine de la whitelist et la blacklist . . . 74
6.5 Résultats d'une première tentative de labélisation . . . 75
6.6 Nom de domaine *.googleapis.com* dans les blacklists et la whitelist Alexa . . . 76
6.7 Résultats de labélisation à l'aide des blacklists et de la whitelist . . . 78
6.8 Affichage du top 20 des noms de domaine les plus populaires considérés légitimes et malicieux . . . 78
6.9 Confirmation des requêtes malicieuses . . . 79
6.10 Détail de l'extension SNI . . . 80
6.11 Nombre de requêtes de botnets par rapport aux requêtes normales . . . 83

6.12 Dataset d'apprentissage et de test 86
6.13 Scores moyens des métriques pour chaque modèle 87
6.14 Matrice de confusion du modèle *Random Forest* (voir code en annexe A.8) . . . 88
6.15 Importance des features dans le modèle *Random Forest* 89
6.16 Classification en tant que botnet des requêtes de l'UNamur (voir code A.10) . . 91
6.17 Top 20 de la classification en tant que botnet des requêtes de l'UNamur (voir code A.10) 91
18 Bro Logs dns.log 101
19 Configuration de la machine virtuelle 106

Liste des tableaux

6.1 Croisement des blacklists et whitelists sans les URLs. 77
6.2 Liste de features de *Fast-Flux* 84
6.3 Liste de features de *Domain-Flux* 85
6.4 Résultats du *cross validation* des différents modèles (voir code annexe A.7) . . . 86
6.5 Résultats de la classification du *test set* sur les différents modèles 87

Acronymes

3LD Third Level Domains. 22

ccTLD country code Top Level Domains. 22
CIS Center for Internet Security. 36

DGA Domain generation algorithms. 41, 42
DKIM DomainKeys Identified Mail. 28
DNS Domain Name System. 21, 38, 41, 47, 49, 55, 57, 60
DoH DNS over HTTPS. 45
DoT DNS over TLS. 45

eTLD effective Top Level Domains. 22

FQDN Fully Qualified Domain Name. 25, 26, 41
FTP File Transfer Protocol. 21, 42

gTLD generic Top Level Domains. 22

HTTP Hypertext Transfer Protocol. 38, 41, 42
HTTPS Hypertext Transfer Protocol Secure. 54

IANA Internet Assigned Numbers Authority. 29, 31
ICANN Internet Corporation for Assigned Names and Numbers. 22, 44
IDS Intrusion Detection System. 42, 47, 53
IP Internet Protocol. 21
IPS Intrusion Prevention System. 42
IRC Internet Relay Chat. 38, 41

NIC Network Information Center. 21

ODP Open Directory Project. 52
OSI Open Systems Interconnection. 22

P2P Peer-2-Peer. 38, 41
pcap Packet Capture. 58, 73
pDNS passive DNS. 47–50

RFC Request For Comments. 21, 22, 35
RR Ressource Record. 27, 40, 42

RRDNS Round-Robin DNS. 40

SLD Second Level Domains. 22
SMB Server Message Block. 36
SNI Server Name Indication. 78
SPF Sender Policy Framework. 28
SSL Secure Sockets Layer. 54
SVM Support Vector Machine. 54

TCP Transmission Control Protocol. 25
TLD Top Level Domains. 22
TLS Transport Layer Security. 45, 55, 79
TTL Time To Live. 40

UDP User Datagram Protocol. 25

VPN Virtual Private Network. 42

Introduction

Le monde de l'internet et son espace métaphorisé induit dans les esprits un état de conscience déformé. Bon nombre d'utilisateurs se sentent en sécurité en utilisant chaque jour leurs outils connectés ; ils n'ont pas toujours conscience des dangers qui les guettent. Autant les océans étaient les lieux privilégiés des pirates à l'époque des premières chevauchées maritimes, autant l'espace du Net est celui des hackers en tout genre qui sévissent dans l'ombre des logiciels et des réseaux. Ils sont bien souvent insaisissables pour le commun des mortels et souvent anonymes pour ceux qui les traquent. Conscient que cette guerre souterraine peut nuire aux libertés et au bien-être de notre société, il m'est apparu opportun et pertinent d'étudier la possibilité de démasquer ces pirates contemporains. C'est dans ce contexte que le présent travail trouve sa place.

Notre démarche s'inscrit donc dans la recherche d'outils informatiques qui permettent la détection de trafics malicieux, c'est-à-dire toute activité réseau étant en relation avec du *phishing*, *spamming*, propagation de *malware*, *botnet* ou tout autre trafic dérivé de ces dernières. Les études existantes et relatives à ces comportements illégaux sur Internet sont de plus en plus nombreuses, mais le phénomène observé est relativement récent. C'est sur base des études récentes que nous tenterons de mettre au point un
outil scientifique qui pourra être mis en oeuvre en vue d'identifier les trafics sur Internet.

Dans le but d'appréhender au mieux la problématique du trafic malicieux, il nous est apparu utile de présenter de manière détaillée le protocole DNS, et ce dans le but d'en fixer les bases et d'en définir ses caractéristiques. Nous passerons ainsi en revue, son architecture, ses propriétés ainsi que son modèle de communication. Nous mettrons également en lumière ses avantages, ses faiblesses et les techniques mises en oeuvre reposant sur ce protocole. Pour compléter ce descriptif nous passerons en revue ses différents degrés de sécurité et un état de l'art des *botnets* ainsi que leurs modi operandi seront largement analysés. Nous soulignerons particulièrement les techniques d'attaques qui s'appuient sur le protocole DNS pour échapper à tout contrôle. En effet, ces techniques d'attaques sont régulièrement utilisées par les pirates informatiques : étudier le protocole DNS est semblable à étudier les océans d'autrefois.

L'architecture du DNS étant mieux précisée, il nous paraît essentiel d'identifier et de caractériser ce que l'on appelle communément le trafic malicieux. En effet, sur base de ses propriétés et de ses modes de fonctionnement, nous proposerons de construire un modèle de détection. Bien que cette tâche relève du défi, nous l'aborderons en nous appuyant sur les récentes études scientifiques relatives à la détection de trafic malicieux orienté vers le protocole DNS. Grâce à ces dernières,
nous pourrons investiguer la construction d'un dispositif de détection capable de mettre en évidence les trafics identifiés comme malicieux.

Le dispositif comportera différents éléments dont un ordinateur puissant, des canevas de travail, des *datasets* que nous analyserons, entre autres, au gré des objectifs poursuivis.

Les méthodes préconisées dans la recherche du trafic malicieux étant relativement complexes, nous en proposerons un cahier des charges afin de permettre à tous de bien suivre notre démarche. C'est pour cette raison qu'il nous paraît opportun de nous attarder sur les concepts de *machine learning* que nous mettrons en oeuvre. Les outils d'analyse, d'amélioration et de mesure seront également clarifiés. Cette transparence dans le développement des outils guidera le lecteur sur les options de travail que nous avons choisies tout au long de ce projet.

La dynamique des tests que nous réaliserons à partir des jeux de données de l'Université et du dispositif de détection que nous proposons (le *pipeline*) sera scindée en deux parties. La première nous conduira vers une identification des trafics analysés ; ce qui nous permettra de labéliser les données issues du *dataset* à l'aide de listes de références nommées *blacklist* et *whitelist*. Après analyse des résultats obtenus, nous procéderons à la seconde partie des tests qui résidera en une classification des requêtes DNS à l'aide de *machine learning*. Durant cette phase des tests, nous serons particulièrement attentifs à la précision des données récoltées. En effet, nous prendrons comme éléments de référence les critères repris dans les études scientifiques récentes pour choisir le modèle le plus performant. Afin de pouvoir préciser la démarche nous terminerons par une mise en perspective des actions entreprises ; cette façon de faire consolidera notre approche qui se veut transparente.

Chapitre 1

Le protocole DNS

Le protocole Domain Name System (DNS) est l'un des acteurs centraux dans le fonctionnement d'Internet. De fait, pour un humain, il est plus aisé de se souvenir d'un nom de site Internet *"www.unamur.be"* plutôt qu'une série de nombres *"138.48.4.201"*. Ce premier chapitre est consacré au concept du protocole DNS, à son architecture, à ses propriétés et met en lumière le mécanisme de communication Client-Serveur.

1.1 Système de noms de domaine

L'interaction de l'homme avec la machine joue un rôle central dans le monde numérique. Tout utilisateur connecté à Internet est constamment à la recherche de contenus, stockés sur des sites web. D'un point de vue réseau, chaque machine ou ressource est joignable au moyen de son adresse unique (Internet Protocol (IP)) [1] intégrée dans un espace hiérarchique. Cependant, comme introduit ci-dessus, l'humain n'est pas habilité à mémoriser une suite d'adresses IP, mais peut en revanche se souvenir d'appellations. Pour pallier ce problème, l'intégration d'un système de noms permet l'association d'une adresse IP avec un nom de ressource. L'organisation Network Information Center (NIC) publie en Décembre 1973 la première liste officielle reprenant les 90 ressources existantes sur le réseau [2]. Le système mis en place à l'origine était composé d'un fichier central HOSTS.TXT contenant une table de conversion reprenant les noms de ressources liés à leur adresse IP. Le NIC était chargé de maintenir à jour la liste de ressources dans laquelle tout changement était communiqué par e-mail. L'ensemble des clients connectés au réseau pouvaient dès lors télécharger périodiquement ce fichier par File Transfer Protocol (FTP) pour en stocker une copie localement [3]. C'est alors que ce fichier était sollicité localement afin de joindre toutes les ressources réseaux à partir d'un nom et non plus d'une adresse IP. Toutefois, ce système a rapidement montré ses limites. D'une part, le fichier HOSTS.TXT atteignait une taille trop volumineuse pour être téléchargé régulièrement ce qui surchargeait les réseaux. D'autre part, la gestion des ressources était une tâche centrale et contraignante au vu de l'expansion d'Internet et par conséquent de l'accroissement des hôtes. C'est pourquoi, en 1981 ce système est devenu obsolète et a été revu à travers plusieurs Request For Comments (RFC)s [4] clés : RFC799, RFC805, RFC819, RFC830, RFC882, RFC883. Finalement, en 1987, Paul Mockapetris publie deux RFCs (RFC1034, RFC1035) étant la base du standard DNS, toujours utilisés aujourd'hui... [32]

1. Les adresses IP version 4 sont les plus utilisées de nos jours. Une adresse *IPv4* est formée sur 4 octets chacun séparé d'un point (par exemple : 192.0.2.53) *https ://www.iana.org/numbers*
2. https ://tools.ietf.org/html/rfc597
3. https ://tools.ietf.org/html/rfc608
4. "Request For Comments" document contenant la spécification d'Internet

Le DNS est un système fondé sur un espace de noms hiérarchique (section 1.2.1) garantissant l'unicité des noms du réseau. Il respecte une structure solide reflétant l'ensemble des organisations existantes sur Internet. Il marque les différences de niveaux à l'aide du caractère *"."* comme *"opac.unamur.be."*. De plus, il répartit la charge des requêtes dans une structure de serveurs, ceux-ci sont alors moins surchargés (section 1.2.2). Le système mis en place permet de réduire le coût de mises à jour des ressources et a l'avantage de confier à chaque organisation la gestion de son propre domaine (section 1.3). Aujourd'hui, de nombreux systèmes sont mis en place afin d'améliorer la performance et la disponibilité d'Internet (section 1.5). Enfin le DNS, principalement décrit dans les deux RFCs 1034 et 1035, est amélioré par d'autres RFCs décrivant les solutions de sécurité, d'implémentation, d'administration, ... [47]

1.2 Architecture

Dans cette partie, nous abordons la structure hiérarchique du DNS ainsi que les deux éléments clés qui le composent pour ensuite décrire brièvement la couche Open Systems Interconnection (OSI) sur laquelle celui-ci repose.

1.2.1 Hiérarchie

La base de données du DNS est fondée sur une structure hiérarchique (voir figure 1.1) dans laquelle sont représentés les noms de domaine à l'instar de *vaxa.isi.edu.* (voir section 1.3.1). Cette structure nommée espace de noms de domaine est constituée de plusieurs niveaux. Au sommet de la hiérarchie, on retrouve le niveau dominant appelé : la racine. Gérée par l'Internet Corporation for Assigned Names and Numbers (ICANN), ce niveau est représenté par le *"point" ou "."* le plus à droite du nom de domaine. Les fils descendant de la racine sont appelés Top Level Domains (TLD) (*.edu*), leurs noms sont gérés par l'ICANN. Ces derniers, considérés comme la partie la plus à droite du nom de domaine, sont divisés en deux grandes catégories :

- generic Top Level Domains (gTLD) : il spécifie le domaine d'activité tel que *.com* pour les activités commerciales, *.edu* pour les ressources concernant l'éducation,... ; [39]
- country code Top Level Domains (ccTLD) : il précise la localisation géographique du domaine comme par exemple, *.be* pour la Belgique, *.ca* pour le Canada. Chaque pays possède ainsi son propre code. [5] [60]

Pour chaque TLD, une compagnie que l'on appelle *Registry* est chargée de maintenir l'accessibilité aux domaines se trouvant sous sa responsabilité. La plupart des TLDs sont formés d'un label. Cependant, il existe des exceptions telles que *.co.uk*, *pvt.k12.ma.us* ou encore *googleapis.com*. Celles-ci sont appelées effective Top Level Domains (eTLD) et sont listées dans une base de données publique. [24]

Le niveau suivant, le Second Level Domains (SLD) (*.isi*), est quant à lui un sous-domaine de la zone TLD. Considéré comme la partie de gauche du nom de domaine , il est vendu et enregistré par un *Registrar*. Il cible le nom du système en ligne. Cela peut être tant un serveur web, qu'un serveur d'e-mails,... [35]

Enfin, le Third Level Domains (3LD) (*vaxa*) ainsi que les niveaux suivants sont gérés par un *Registrant*. Ce dernier est une organisation ou une personne possédant les droits d'utilisation et de gestion d'un SLD. Un registrant peut alors créer autant de sous niveaux qu'il le souhaite tout en respectant les limites décrites à la section 1.3.1.

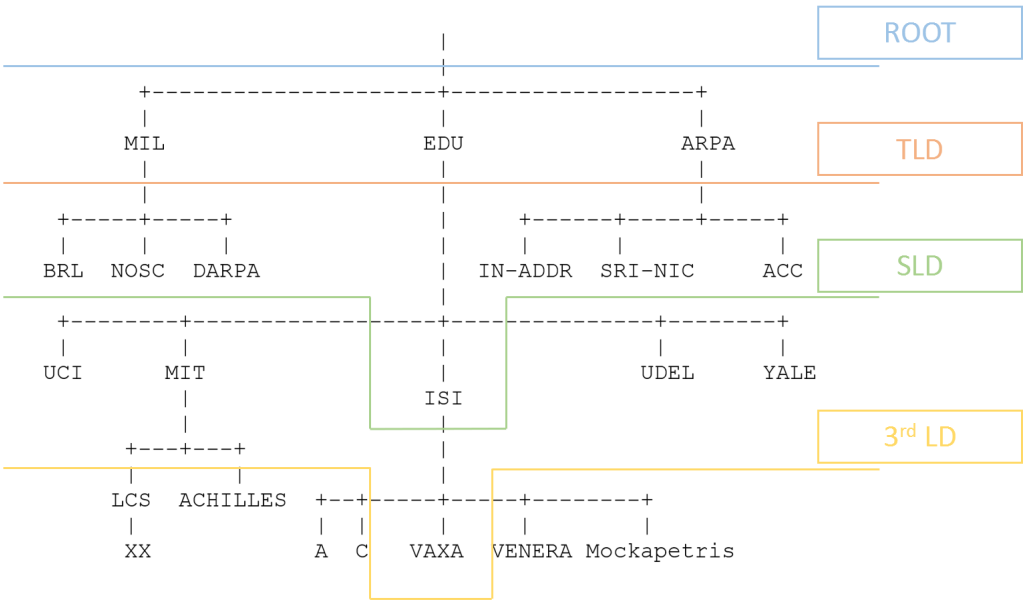

FIGURE 1.1 – Niveaux des domaines dans l'espace de noms

1.2.2 Modèle client-serveur

Pour toutes les communications DNS, on distingue 2 éléments principaux : les serveurs de noms et les résolveurs.

Les serveurs de noms

Les serveurs de noms sont des systèmes stockant les informations correspondant à leur sous-domaine dans l'espace de noms. Ce sous-domaine, appelé "zone", est décrit dans la section 1.3.2. Les informations qu'elle contient sont stockées sous forme de collections de *resource records*, détaillée à la section 1.3.3, dans leur base de données. De manière générale, chaque zone est composée d'un serveur de noms primaires et sa base de données. Dans certains cas, les organismes propriétaires des serveurs peuvent opter pour une solution redondante dans laquelle la base de données est dupliquée sur plusieurs serveurs distants. On parlera alors de serveurs de noms secondaires.[13] Un serveur de noms répondant aux requêtes clients pour sa zone est dit AUTORITAIRE.[37] Le DNS étant un système distribué, la configuration des serveurs est maintenue par l'organisme responsable du nom de domaine. [47]

Les résolveurs

Le résolveur ou *resolver* est un programme système situé sur la machine client (appellé *stub resolver*) opérant comme interface entre d'une part, les applications nécessitant une résolution de noms de domaine (voir 1.2.3) et d'autre part, les serveurs de noms. Ainsi, c'est au résolveur que revient la tâche d'initier, de gérer et d'interpréter les requêtes vers les serveurs distants. Les réponses possibles d'un serveur peuvent être soit la réponse attendue ou une erreur. En plus de son rôle principal, un serveur de noms peut également opérer en tant que résolveur récursif ou *recursive resolver*[5] comme le montre la figure 1.2. De manière générale, les résolveurs choisissent un des deux types d'approche dans le processus de résolution d'un nom de domaine, à savoir l'approche récursive ou itérative. Nous expliquons plus en détails ces deux modes dans le point suivant 1.2.3. [13]

5. ou *caching resolver* ou *full resolver* ou *recursive name server*, défini dans le RFC7626

1.2.3 Résolution

Comme nous l'avons vu précédemment, le résolveur initie une requête vers un serveur de noms à la demande d'un programme local. Dans le meilleur des cas, ce serveur est autoritaire sur la zone et renverra l'information. En revanche, dans le cas contraire, c'est lui qui se chargera de rapatrier l'information en interrogeant d'autres serveurs de noms dans l'espace de noms. Ce serveur reprenant le rôle de résolveur est appelé *recursive resolver*. [47] L'ensemble de ce processus est appelé la "résolution de noms". Par conséquent, lors d'une résolution, le résolveur peut initier un des deux types d'approche dans sa requête soit récursive, soit itérative.

L'approche récursive

Dans cette première approche, le résolveur spécifie dans sa requête (voir section 1.4.2) qu'il désire placer la charge de résolution sur le serveur de noms. Ainsi, le serveur de noms recevant la requête est responsable de la gestion de celle-ci jusqu'à l'obtention d'une réponse complète (voir figure 1.2). Une fois la réponse obtenue, il communiquera celle-ci au client. Cette méthode a la particularité d'initier très peu de trafic entre le résolveur et le serveur de noms. De fait, supposons que les clients se situent dans un réseau de moyenne entreprise et qu'ils travaillent avec un serveur de type d'approche récursive, ceux-ci pourraient faire bénéficier à l'ensemble du réseau local une nouvelle réponse. Comme nous le verrons plus tard (voir section 1.3.5), les serveurs stockent les informations obtenues dans un cache, pour une période limitée. L'approche récursive est de loin l'approche la plus utilisée dans un réseau local au contraire d'Internet. [47]

L'approche itérative

Dans cette deuxième approche, le serveur de noms interrogé réfère au serveur itératif (*recursive resolver*) le serveur successeur afin que celui-ci poursuive sa requête. À la suite d'une séquence de requêtes, le *recursive resolver* reçoit la réponse complète du serveur autoritaire. Cette méthode est la plus utilisée sur internet comme l'illustre la figure 1.2. Cette méthode reprend les avantages énoncés dans la deuxième partie de la section 1.1.

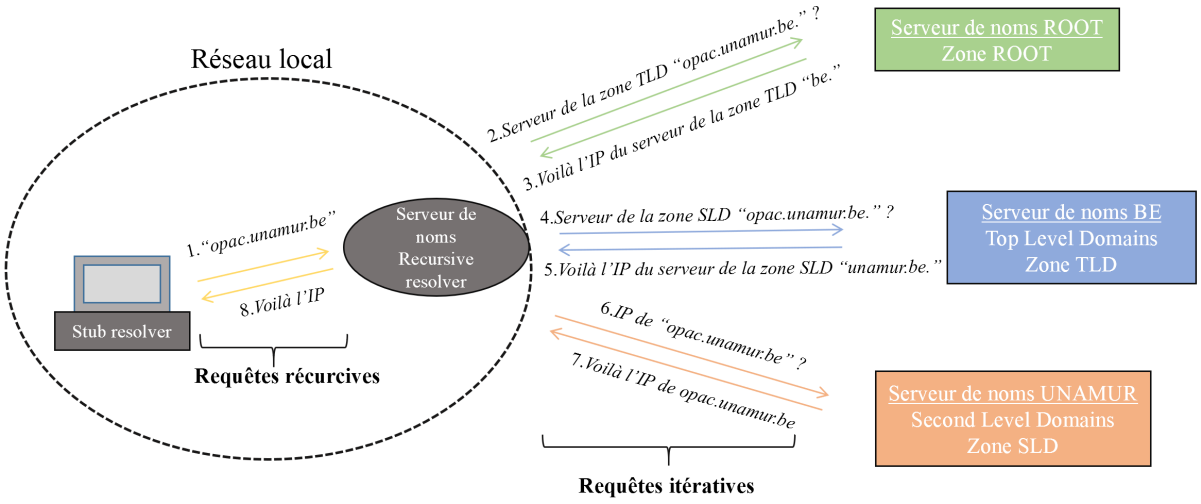

FIGURE 1.2 – Fonctionnement DNS - Résolution de noms de domaine

Échec

Les approches ci-dessus considèrent la résolution comme réussie au vu de la présence de l'information dans la configuration d'un serveur de noms autoritaire. Cependant, le DNS est capable de gérer toutes sortes d'erreurs en informant le client à l'aide d'un code adéquat (RCODE 1.4.2). En se référant à la figure 1.2, dans le cas où le nom de domaine *unamur.be* n'existerait pas, le serveur de noms de la zone TLD renverrait un message d'erreur explicite au client. La résolution serait alors considérée comme un échec.

1.2.4 Couche Transport

Le DNS est basé sur l'architecture connue Client-Server et est un protocole *sans état*.[20] Pour communiquer, il repose sur deux protocoles de la quatrième couche (Transport) du modèle OSI.[56] Le premier est l'User Datagram Protocol (UDP), le protocole le plus utilisé pour les requêtes standard car il est moins coûteux et obtient de meilleures performances.[40] Le deuxième est Transmission Control Protocol (TCP) ; celui-ci est privilégié pour les activités plus sensibles comme la mise à jour des zones (requête de type AXFR voir section 1.4.3).

UDP

Le serveur DNS écoute sur le port 53. La taille des messages est limitée à 512 bytes [6] (sans compter l'entête IP ou UDP). Par conséquent, les messages d'une taille supérieure à 512 bytes sont tronqués et signalés en configurant le bit TC (voir section 1.4.2). L'UDP n'offre aucune garantie que le message arrive à destination, ce qui induit une stratégie de retransmission (2-5 secondes).[38]

TCP

TCP est utilisé pour des transferts exigeant une connexion fiable. Les serveurs DNS écoutent sur le port 53 tout comme l'UDP. Les messages envoyés contiennent un préfixe de deux octets signalant la longueur du message, excluant les deux octets de ce préfixe. Cette information permet de rassembler le message complet à l'arrivée et de procéder à son parsing.[38]

1.3 Propriétés

1.3.1 Noms de Domaine

La figure 1.3 est la représentation partielle de l'espace de noms de domaine. Chaque nœud, père ou fils de l'arborescence, est appelé nom de domaine. Son domaine est alors la concaténation des nœuds, ou labels, pères jusqu'au label *root*. De manière générale, les organismes choisiront les labels les plus explicites reflétant leur fonction. Un nom de domaine possède plusieurs caractéristiques :

- c'est une séquence de labels séparés par des points "." ;
- le nombre maximum de labels autorisés est 127 ;[36]
- les labels ont une longueur limitée à 63 octets ;
- les frères d'un nœud du même niveau ne peuvent avoir le même label ;
- il n'est pas sensible aux majuscules et aux minuscules, mais pourrait l'être un jour ;
- sa représentation complète reprend tous ses nœuds. Par exemple : *vaxa.isi.edu* est appelé nom absolu ou Fully Qualified Domain Name (FQDN) ;

6. 1 Byte = 1 octet = 8 bits

— la longueur du FQDN est limitée à 255 octets ;
— sa représentation partielle, appelée nom relatif, nécessite d'être complétée par un programme local. Par exemple : le nom relatif *vaxa* dans le domaine *isi.edu* ;[37]
— un nom de domaine peut se trouver dans plusieurs domaines. Par exemple : *vaxa.isi.edu* appartient au domaine *isi.edu* mais aussi au domaine *edu*.

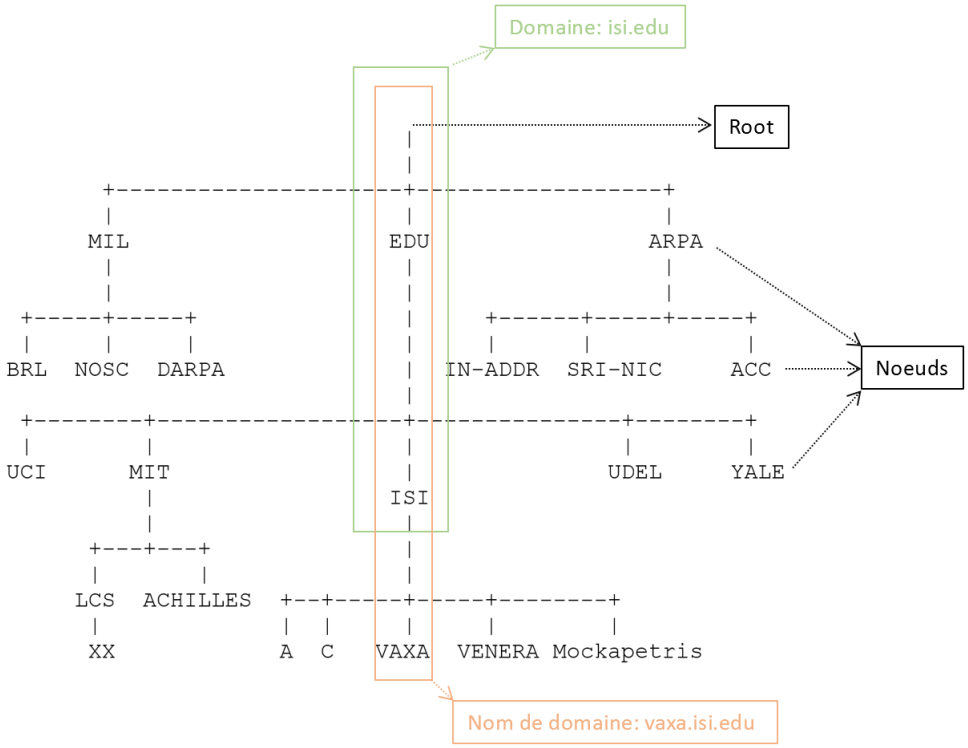

FIGURE 1.3 – Domaines et noeuds

1.3.2 Zones

Une zone est une partie délimitée dans l'espace de noms de domaine. Elle est composée de nœuds adjacents dont, inéluctablement, un nœud le plus haut souvent utilisé pour désigner la zone. Une zone possède un ou plusieurs serveurs de noms autoritaires ayant comme mission de répondre aux requêtes relatives à la zone (section 1.2.2). Dans le cas où la requête n'est pas incluse dans la zone du serveur autoritaire, ce dernier délègue la requête au serveur autoritaire de la zone le plus appropriée. Chaque organisme doit gérer ses serveurs de noms et doit veiller à maintenir à jour leur base de données (voir 1.3.3) [13]

À titre d'exemple (figure 1.4), la zone *isi.edu* délègue l'autorité de la zone *vaxa.isi.edu* au serveur autoritaire *vaxa*. Si une requête interroge *isi.edu* à propos d'un hôte dans la zone *vaxa.isi.edu* alors *isi.edu* redirigera la requête vers le serveur de noms *vaxa.idi.edu*.

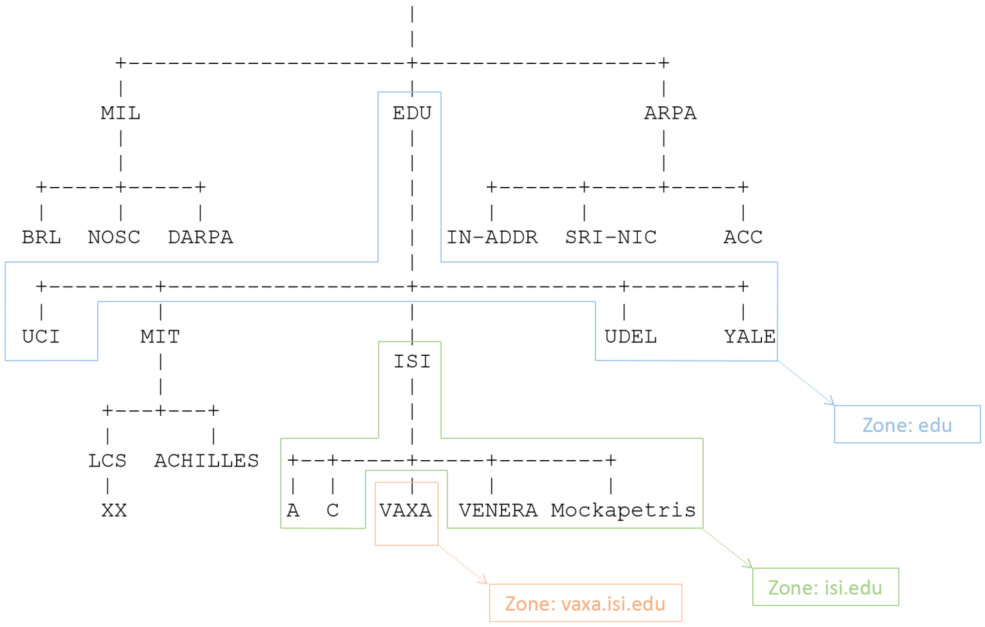

FIGURE 1.4 – Zones

1.3.3 Resource record

La base de données d'un serveur de noms est constituée des *resource records* ou RRs. Les RRs décrivent les informations de la zone dont, l'ensemble des hôtes, les références de délégation vers les autres zones ou encore la transparence d'informations pour l'authenticité des serveurs e-mails. Un Ressource Record (RR) permet, par exemple, de répondre à une requête de traduction d'un hôte vers l'adresse IP ou vice versa.[38] La figure 1.5 montre un exemple de fichier de configuration.

Une entrée RR dans un fichier de configuration comporte les cinq champs suivants :

1. OWNER (32 bits) : le nom de domaine dans lequel se situe le RR.
2. TYPE (16 bits) : spécifie le type de la ressource du RR :
   - A : indique l'adresse IP d'un hôte. Dans ce cas le OWNER sera le nom de l'hôte et RDATA l'adresse IP ;
   - AAAA : indique l'adresse IPv6 d'un hôte. Dans ce cas le OWNER sera le nom de l'hôte et RDATA l'adresse IPv6 ;
   - CNAME : spécifie un alias. L'OWNER sera l'alias et RDATA le nom réel du serveur/hôte ;
   - MX (Mail Exchange) : renseigne le ou les serveur(s) e-mails du domaine. OWNER sera le nom de domaine et RDATA le nom du serveur e-mails de ce nom de domaine ;
   - PTR : pointe vers une autre partie de l'espace de noms de domaine. Il est utilisé pour la résolution inverse ;
   - NS : ce type indique le serveur de noms ayant autorité sur le domaine ;
   - SOA : indique le début d'une zone autoritaire ;
   - HINFO : indique la description de l'hôte, son matériel, son système d'exploitation, ... ;
   - TXT : permet d'ajouter du texte additionnel pour un nom de domaine.
3. CLASS (16 bits) : identifie la famille du protocole. La plupart du temps "IN", pour *Internet System*, est utilisé.
4. TTL (32 bits) : durée de vie en secondes de l'entité. Il est utilisé par les résolveurs afin de rafraîchir leurs informations stockées localement. Il est souvent configuré en terme de jours ; cependant il peut être raccourci temporairement afin d'anticiper de nombreux changements.

5. RDATA : dépendant du type, RDATA peut prendre l'une des valeurs suivantes :
   — A : pour la classe IN, RDATA sera de 32 bits pour stocker une adresse IPv4 ;
   — CNAME : nom de domaine ;
   — MX : gestion d'e-mails, RDATA sera de 16 bits pour stocker le nom de domaine ;
   — NS : un nom d'un hôte ;
   — PTR : un nom de domaine ;
   — SOA : plusieurs champs, comme l'adresse e-mail du responsable de la zone ;
   — TXT : des informations pour toute ressource externe au domaine. TXT est notamment utilisé pour la vérification de l'authenticité et de l'intégrité d'e-mails. Sender Policy Framework (SPF) [7] ou encore DomainKeys Identified Mail (DKIM) [8] sont deux exemples utilisant un resource record de type TXT.

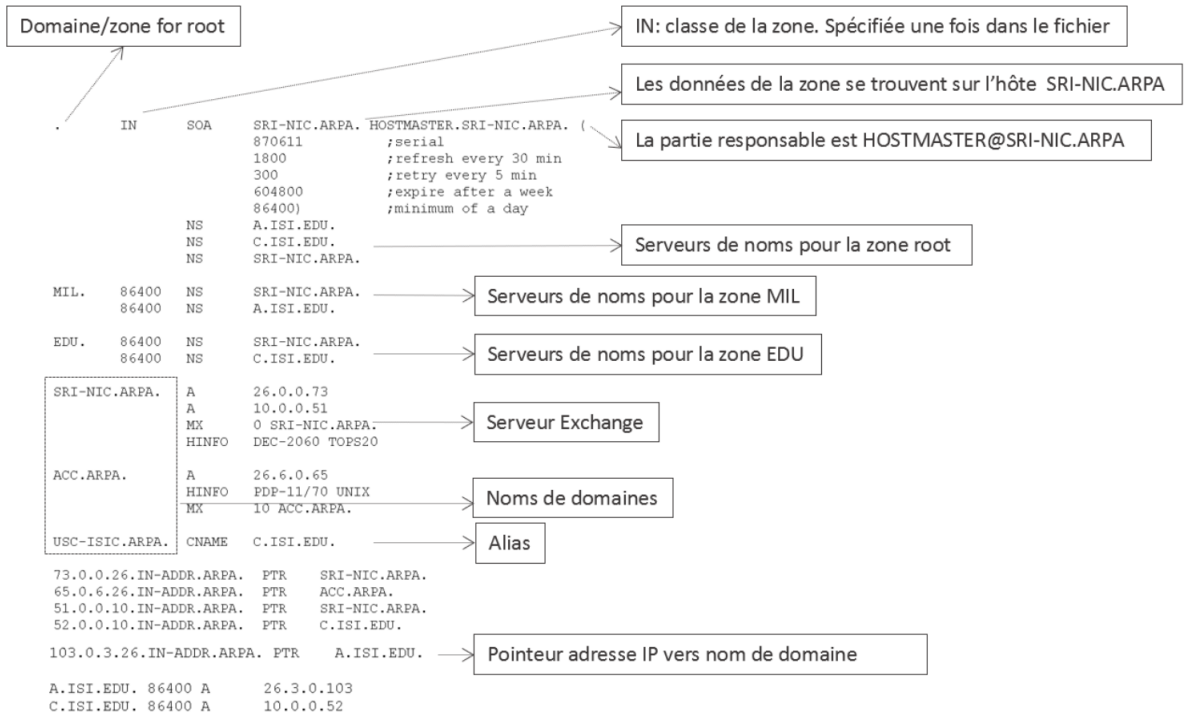

FIGURE 1.5 – Configuration de serveur de noms pour la zone ROOT [37]

1.3.4 in-addr.arpa

De manière générale, le DNS a pour tâche de traduire un nom de domaine en adresse. Cependant, dans certains cas, il peut s'avérer utile de traduire une adresse en nom de domaine (appelé : reverse DNS ou encore rDNS). Or, l'espace de noms de domaine est indexé sur les noms, ce qui rend facile la résolution d'un nom mais pas d'une adresse. Pour remédier à ce problème, l'espace des noms de domaine possède le domaine *in-addr.arpa* dont le but est de convertir une adresse en nom de domaine. Ce domaine contient jusqu'à 6LD (voir section 1.2.1), les deux premiers étant *in-add.arpa* et les 4 derniers les 4 octets inversés. Dans ce cas, une adresse IPv4 *192.168.1.20* aura comme nom de domaine *20.1.168.192.in-addr.arpa*. Il est évident mais utile

7. Le protocole SPF met en place un mécanisme, donnant la possibilité à un serveur d'e-mails de refuser la réception d'un e-mail si l'émetteur n'appartient pas à la liste des émetteurs autorisés. Cette liste est obtenue via un RR TXT. RFC4408

8. Le protocole DKIM fournit au serveur d'e-mails de réception un mécanisme de vérification d'authenticité et d'intégrité d'un e-mail RFC6376

de rappeler qu'il n'y a pas de résolution d'adresse IP. Concrètement, si un utilisateur entre *138.48.4.201* dans son web browser, aucune requête DNS n'est générée. [13] [37]

1.3.5 Cache

Généralement, les résolveurs (voir 1.2.2) stockent ou mettent en cache toutes les données reçues au sein d'une réponse. Si une donnée s'avère être déjà présente dans le cache, le résolveur choisira d'en garder une des deux. Cependant, plusieurs types de données devraient être proscrits du cache, à savoir :

- si les données sont déjà présentes dans le fichier de configuration de la zone autoritaire ;
- si c'est le résultat de requêtes inversées ;
- si dans le résultat de requêtes, le QNAME contient un label de type masque ou "*" ;
- si le résultat provient d'un requête douteuse, si le résolveur reçoit une requête non sollicitée, un RR qui n'a pas été demandé ;
- si un ensemble de RRs de même type est déjà disponible, le résolveur doit choisir entre tout stocker en cache ou tout ignorer.

Un serveur de noms pourrait mettre en cache une structure ou un ensemble d'informations sur une partie du domaine. Lors de la mise en cache, l'information est accompagnée d'un TTL (1.3.3) après quoi l'information est supprimée du cache. Le TTL peut être volontairement court afin de limiter la mise en cache d'informations et peut être également configuré à zéro pour désactiver la mise en cache. Cependant, la RFC1912 [41] recommande une valeur minimum de 1 à 5 jours.
Couplé à l'approche récursive (voir section 1.2.3), le cache devient une arme indispensable pour réduire la quantité de trafic dans un réseau et réduire le temps de résolution d'une requête. À titre d'exemple, si l'utilisateur était amené à initier une deuxième requête DNS vers *opac.unamur.be* le résolveur se référait, après avoir effectué une première requête DNS où le serveur autoritaire aurait répondu, à son cache (voir figure 1.2). Cependant le revers de la médaille en fait un élément particulièrement sensible aux attaques, comme expliqué dans la section 2.4.

1.4 Messages

1.4.1 Structure

Toutes les communications entre les clients et les serveurs sont réalisées par une structure nommée message et décrite dans la RFC1035.[38] Les messages contiennent des paramètres maintenus par l'Internet Assigned Numbers Authority (IANA) [33] et sont divisés en cinq sections (figure 1.6) :

- HEADER OU EN-TÊTE : toujours présent dans un message. Il contient plusieurs champs mis à jour par le client et le serveur. L'en-tête est détaillée dans le point 1.4.2 ;
- QUESTION : contient les champs décrivant la question à poser au serveur de noms. La structure d'une question est détaillée dans le point 1.4.3 ;
- ANSWER : contient les *resource records* répondant à la question, ce point est décrit dans la section 1.4.4 ;
- AUTHORITY : fournit la liste des *resource records* pointant vers un ou plusieurs serveur(s) de noms autoritaire ;
- ADDITIONAL : contient les *resource records* additionnels liés à la question mais n'y répondant pas pour autant.

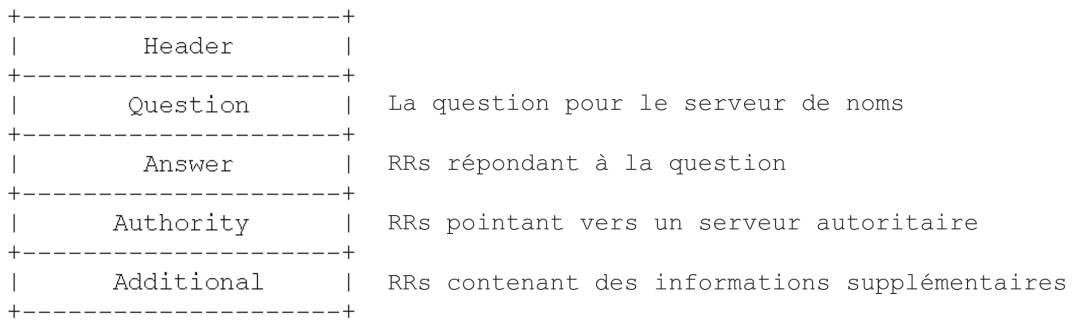

FIGURE 1.6 – Format d'un message [38]

1.4.2 En-tête

La première section d'un message est l'en-tête et est toujours présente. Les champs qu'elle contient caractérisent le type de message et résument la quantité d'informations qu'il véhicule. Ces champs sont (figure 1.7) :

- IDENTIFIER : est l'identifiant assigné par le programme générant la requête ;
- QR : est configuré à 0 pour une question et à 1 pour une réponse ;
- OPCODE : spécifie la nature de la question. Configuré à 0 pour une requête standard, et à 2 pour une requête serveur ;
- AA : est configuré à 1 pour signifier que la réponse provient d'un serveur autoritaire ;
- TC : est configuré à 1 pour signaler qu'un message a été tronqué ;
- RD : indique au serveur de noms de poursuivre la requête en approche récursive (voir 1.2.3) ;
- RA : informe la machine initiant la requête si l'approche récursive est disponible sur le serveur de noms ;
- Z : n'est pas utilisé mais doit être à 0 ;
- RCODE : *Response Code*. Codé sur 4 bits, il peut prendre plusieurs valeurs. Les plus importantes sont :
  - 0 : Pas d'erreur ;
  - 1 : Erreur de format estimé par le serveur de noms ;
  - 2 : Problème rencontré sur le serveur de noms ;
  - 3 : Venant d'un serveur de noms autoritaire, le nom de domaine n'existe pas.
- QDCOUNT : spécifie le nombre d'entrée dans la section question du message ;
- ANCOUNT : spécifie le nombre de *resource records* dans la section réponse du message ;
- NSCOUNT : spécifie le nombre de *resource records* dans la section autorité du message ;
- ARCOUNT : spécifie le nombre de *resource records* dans la section additionnelle du message.

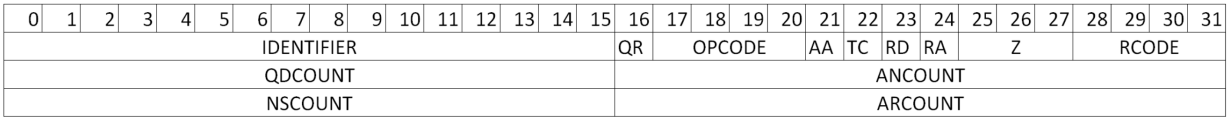

FIGURE 1.7 – Structure de l'en-tête d'un message DNS [38]

1.4.3 Question

Dans un message, la deuxième section correspond à la question. Celle-ci est générée par le client et comporte les trois champs suivants :

— QNAME : contient le nom de domaine (nom absolu ou FQDN) à résoudre ;
— QTYPE : définit le type de la requête. Les plus utilisés sont A, AAAA, MX, AXFR, TXT de la liste maintenue à jour par l'IANA ;[33]
— QCLASS : définit la classe de la requête. La plus utilisée est "IN". [33]

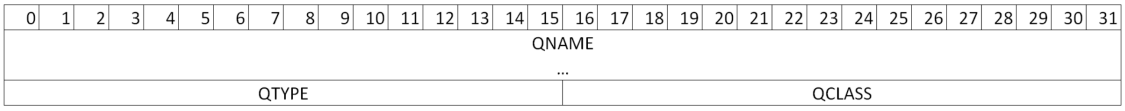

FIGURE 1.8 – Structure d'une question [38]

1.4.4 Réponse

Lors du traitement de la requête par le serveur de noms, celui-ci consulte sa base de données et recherche les informations en relation avec la requête. Une fois toutes les informations utiles collectées, elles sont ajoutées dans les différentes sections de la réponse. Or, nous avons vu au point 1.3.3 que la base de données est une liste de *resource records* du même format. C'est pourquoi les trois dernières sections d'un message ont la même structure, c'est à dire une collection de *resource records*. On notera la présence d'un champ supplémentaire RDLENGTH, celui-ci étant la longueur de la valeur du champ RDATA en octets. La structure d'un *resource record* est représentée comme suit :

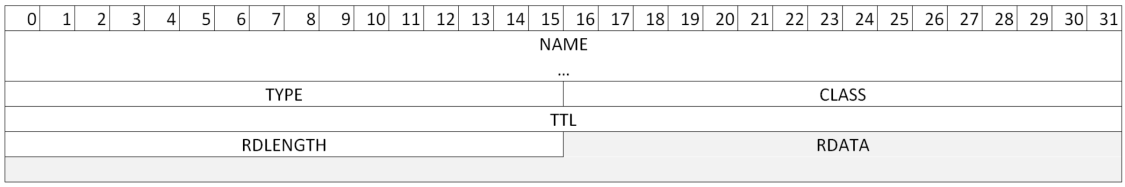

FIGURE 1.9 – Structure d'un *resource record* [38]

Exemple de réponse d'un serveur de noms

La majorité des requêtes DNS venant des logiciels sont à l'affût d'informations ciblées. Cependant, dans le cas de problèmes réseaux persistants, il peut s'avérer utile de formuler des requêtes sur mesure. C'est pourquoi, il existe une palette d'outils permettant d'interroger les serveurs de noms. Un des outils les plus connus est la commande *dig* (*domain information groper*). En plus d'être flexible, son utilisation est facile et le résultat qu'elle produit est clair. [3]

Dans le listing 1.1, la commande *dig* retourne la résolution du nom de domaine *opac.unamur.be.*. Celle-ci liste, entre autres, un alias pointant vers le nom de domaine *bib.sipr.ucl.ac.be.*.

```
rebaudin ~ $ dig opac.unamur.be

; <<>> DiG 9.10.3-P4-Ubuntu <<>> opac.unamur.be
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 60519
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 6, ADDITIONAL: 13

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1280
;; QUESTION SECTION:
;opac.unamur.be.                        IN      A

;; ANSWER SECTION:
opac.unamur.be.         66      IN      CNAME   bib.sipr.ucl.ac.be.
bib.sipr.ucl.ac.be.     86315   IN      A       130.104.5.68

;; AUTHORITY SECTION:
be.                     7660    IN      NS      a.ns.dns.be.
be.                     7660    IN      NS      b.ns.dns.be.
be.                     7660    IN      NS      d.ns.dns.be.
be.                     7660    IN      NS      c.ns.dns.be.
be.                     7660    IN      NS      x.ns.dns.be.
be.                     7660    IN      NS      y.ns.dns.be.

;; ADDITIONAL SECTION:
a.ns.dns.be.            54560   IN      A       194.0.6.1
b.ns.dns.be.            73373   IN      A       194.0.37.1
c.ns.dns.be.            31329   IN      A       194.0.43.1
d.ns.dns.be.            52415   IN      A       194.0.44.1
x.ns.dns.be.            54560   IN      A       194.0.1.10
y.ns.dns.be.            52415   IN      A       120.29.253.8
a.ns.dns.be.            54560   IN      AAAA    2001:678:9::1
b.ns.dns.be.            53170   IN      AAAA    2001:678:64::1
c.ns.dns.be.            53170   IN      AAAA    2001:678:68::1
d.ns.dns.be.            52415   IN      AAAA    2001:678:6c::1
x.ns.dns.be.            54560   IN      AAAA    2001:678:4::a
y.ns.dns.be.            52415   IN      AAAA    2001:dcd:7::8

;; Query time: 17 msec
;; SERVER: 144.254.71.184#53(144.254.71.184)
;; WHEN: Sat Jan 26 17:17:01 CET 2019
;; MSG SIZE  rcvd: 456
```

Listing 1.1 – Exemple de réponse d'un serveur de noms via la commande *dig*

À titre d'exemple, une partie de la réponse obtenue ci-dessus est représentée dans la figure 1.10.

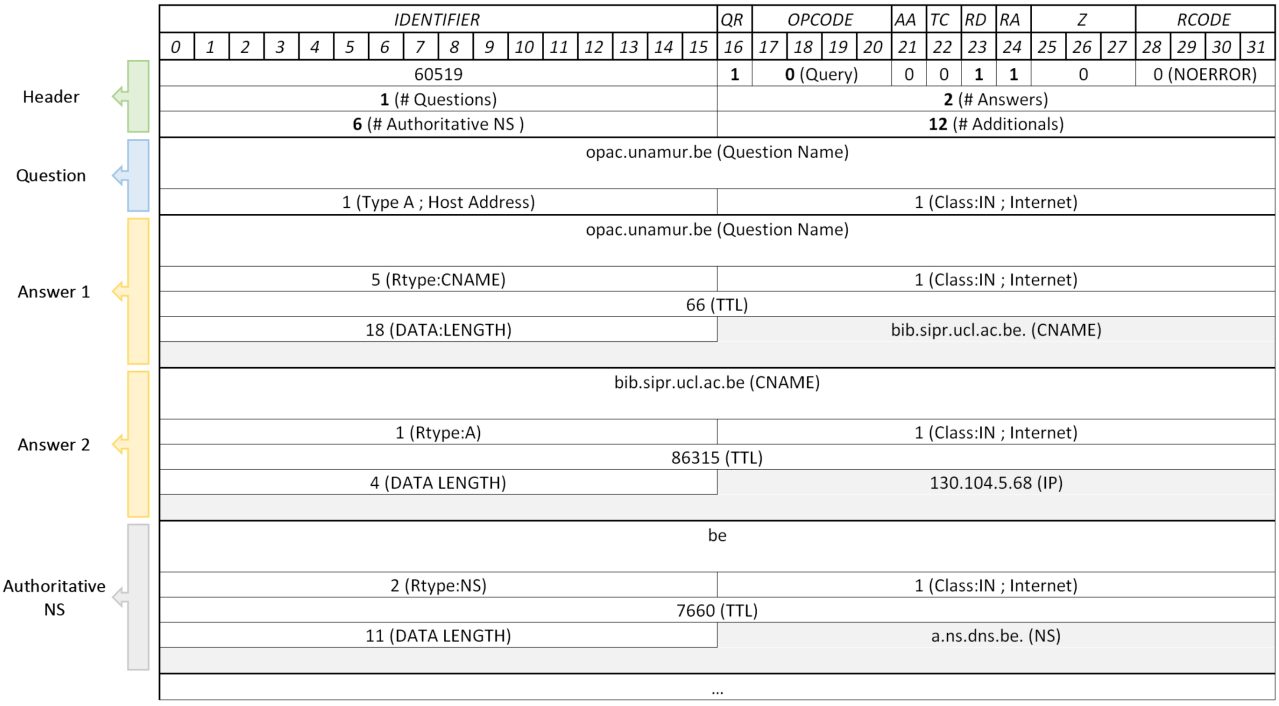

FIGURE 1.10 – Illustration d'une réponse DNS

1.5 Services DNS

En plus du rôle essentiel de traduction de noms de domaine en adresses IP, le DNS a permis de développer de nombreuses techniques afin d'améliorer la performance et la disponibilité des services légitimes d'Internet. Parmi les techniques, on retrouve le Round-Robin DNS et le Content Delivery Network. [16]

1.5.1 Round-Robin DNS - RRDNS

Les sites web ayant une taille relativement importante et traitant des millions de requêtes quotidiennes ne se limitent pas à un unique serveur Web. De fait, ces organisations maintiennent un groupe de serveurs dans un centre de données situé dans un lieu géographique. Les requêtes sont alors distribuées à travers ces serveurs, ce qui augmente la disponibilité en cas d'attaque ou de problème d'un serveur du groupe. Dans la pratique le serveur de noms possède une liste de RRs de type A pour le même nom de domaine. Lors d'une requête pour ce même domaine, le serveur renverra une sélection aléatoire de RRs de type A. [30]

1.5.2 Content Delivery Network - CDN

Le réseau de livraison de contenu ou "Content Delivery Network" est un service supplémentaire ajouté au Round-Robin DNS. L'idée, pour un serveur de services (serveur DNS, site web,...) n'est plus de limiter la distribution de charge de requêtes à travers un seul centre de données géographiques mais plutôt d'étendre la distribution à travers des centres géographiquement dispersés. Cette technique couplée au service DNS permet d'accéder, à partir d'un nom de domaine, au serveur géographiquement le plus proche. Les avantages essentiels sont de limiter la communication sur de longues distances pour améliorer le temps de réponse, diminuer l'impact d'une attaque en multipliant les centres de données et ainsi augmenter la disponibilité. [30]

Chapitre 2

DNS & Sécurité

Dans le chapitre précédent, nous avons décrit le rôle essentiel du protocole DNS au sein de l'architecture d'Internet. Pour le représenter, une série de RFC (RFC1034, RFC1035,...) ont été créées. Or, le trafic DNS est loin d'être un trafic inoffensif. En effet, nombreux sont les logiciels exploitant ses capacités pour entretenir leurs activités et cela les hackers l'ont bien compris.
Ce chapitre définit tout d'abord un *Malware*, ses modèles de propagations et de comportements. Ensuite, nous déclinons un malware plus particulier. De fait, dans leur rapport (voir figure 2.1), Cisco fournit le top 5 des menaces réseaux les plus récurrentes. Et c'est à hauteur de 58% que le botnet domine le classement. C'est pourquoi un état de l'art des botnets est réalisé. Enfin, nous évoquons les différentes attaques visant les serveurs DNS ainsi que les utilisateurs. Pour finir, nous mentionnons les limites du protocole DNS en terme de securité et évoquons quelques pistes pour y remédier.

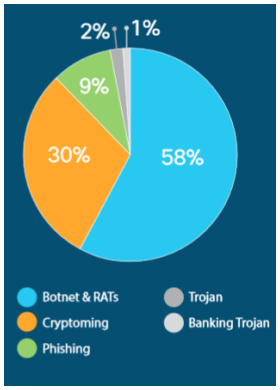

FIGURE 2.1 – Cisco Cybersecurity February 2019 Threat Report [12]

2.1 Malware

Un problème majeur aujourd'hui est l'infection des systèmes par l'intermédiaire de fichiers téléchargés sur des sites web infectés. Autrement dit, il n'y a pas d'interaction directe entre l'attaquant et la victime. L'attaquant transfert le fichier infecté sur un serveur et attend que la victime le télécharge et l'exécute. Un travail est effectué par la plupart des moteurs de recherche afin de bloquer (ou *blacklister*) les sites web infectés. Cependant, les moteurs de recherche sont très prudents et prennent donc du temps à *blacklister* un site web pour éviter les False Negatives [1]. De plus, les utilisateurs ignorent les avertissements mis en place ou encore trouvent une alternative qui est rarement la solution. [23] C'est alors à l'organisation dans

1. C'est à dire d'empêcher un contenu alors qu'il est sain.

laquelle se trouve la victime de se protéger de ces fichiers infectés.
Le *malware* (Malicious Software) est un terme générique défini comme logiciel malicieux et reprenant un ensemble de logiciels comme les virus, trojans, worms et tous les logiciels utilisés par les hackers dans le but d'accéder aux données sensibles de la victime. [25] Il existe plusieurs manières de catégoriser un malware :

1. La propagation du malware :
   — Virus : se propage de système en système par l'intermédiaire d'un programme. Lorsque le programme est exécuté par l'utilisateur, le virus contamine l'ensemble du système ;
   — Trojan : ne se réplique pas lui-même. Un fichier contenant un trojan contient du code malicieux qui, une fois exécuté, contamine le système. Le fichier est souvent distribué via un spam ;
   — Worm : se propage lui même de système en système sans avoir besoin d'être intégré à un fichier.
2. L'objectif du malware :
   — Rootkit : est un programme contenant un ensemble d'outils donnant accès à distance à un système permettant à l'attaquant d'exécuter des commandes ;
   — Ransomware : est un programme qui chiffre les données de l'ordinateur de la victime et demande une rançon ;
   — D'autres objectifs existent comme *Adware*, *Cryptojacking*.[25]

À titre illustratif et d'après le Center for Internet Security (CIS) [22], parmi les malwares les plus connus (voir figure 2.2), Wannacry est l'un des derniers ransomwares cryptoworm utilisant comme type de propagation le vecteur réseau en abusant du protocole Server Message Block (SMB). Le CIS pointe également que le malspam (malware via le spam) est le vecteur d'infection principal en Janvier 2019.

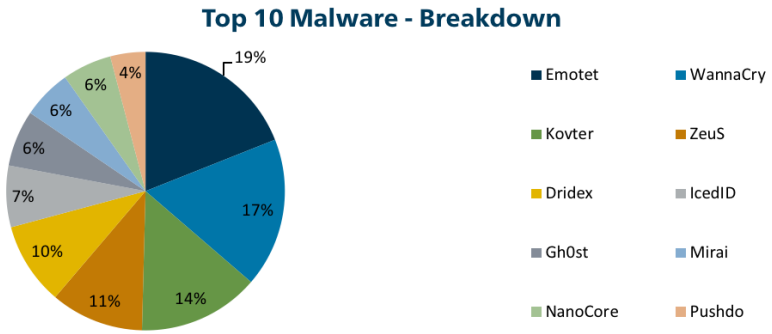

FIGURE 2.2 – Top 10 des malwares représentant 52% du trafic total de malware pour le mois de Janvier 2019 [22]

2.2 Les Botnets

Dans cette section, nous commençons par définir ce qu'est un *botnet* et son usage. Ensuite, nous décrivons ses composants et les différentes topologies. Enfin, nous présentons leurs moyens de communication. Finalement, nous voyons les techniques des botnets afin d'échapper aux moyens de détection.

2.2.1 Définition

Un *botnet* (l'abréviation de bot network) est une collection d'objets corrompus (la plupart du temps des ordinateurs) connectés à Internet ayant pour but d'exécuter des opérations inopinées à l'insu de leur propriétaire. Ces objets, souvent mal protégés, sont infectés par un malware ouvrant un port de communication vers des serveurs de commandes et de contrôles ou serveurs *C&C* ou encore *C2* (voir section 2.2.3). [2] Une taille moyenne de botnet, 1000 ou 2000 objets, est suffisante pour nuire à un service réseau.[30]

2.2.2 Usages du botnet

Les créateurs de bot mettent en place un panel d'outils valorisant au maximum leur *botnet* (ordinateur infecté voir point 2.2.3). Ainsi, un système potentiellement corrompu a plusieurs valeurs telles que les ressources de la machine, sa bande passante, mais aussi les données de l'utilisateur. Les usages d'un botnet sont multiples :

- *DDoS* : profite du nombre de bots composant le botnet pour générer un nombre abondant de requêtes vers un service réseau dans le but de le ralentir ou de l'interrompre ;
- *Exploit scanning* : est une évolution du malware dans laquelle celui-ci est capable de scanner le réseau, de lister des cibles et d'installer une copie du malware pour ensuite l'exécuter ;
- *Download and installation* : est une fonctionnalité installée par défaut dans la plupart des bots. Elle permet de pouvoir effectuer des téléchargements et d'exécuter des fichiers ;
- *Click fraud* : est un moyen mis en place par le malware pour afficher des publicités à la victime dans le but de simuler des clics sur celles-ci ;
- *Spyware* : est la capacité pour un bot à récupérer des informations personnelles de la victime, comme des captures d'écran, du *keylogging*[2] ou encore du phising. Les données peuvent alors servir à l'attaquant ou peuvent être revendues ;
- Les botnets peuvent être également loués à d'autres organisations de hackers dans un but bien précis ; [1]
- D'autres usages sont expliqués dans *Botnets as a Vehicle for Online Crime* [50].

2.2.3 Composants

Quels que soient l'architecture, la taille et le protocole de communication, les botnets sont composés de quatre éléments : le *botmaster*, le *bot*, le(s) serveur(s) C&C et leurs canaux de communication (voir figure 2.3). De plus en fonction de leur topologie (voir section 2.2.4), les botnets peuvent avoir une ou plusieurs couche(s) de *proxies* entre les serveurs C&C et les *bots*. [32] Voici les quatre éléments composant le botnet :

1. *Botmaster* : est l'entité criminelle, une personne ou un groupe, qui contrôle la manière dont le botnet opère. C'est lui qui décide quelle opération et à quel moment celle-ci doit être envoyée aux bots pour exécution.
2. *Bot* : est un ordinateur infecté par un malware transformant celui-ci en zombie et faisant partie du botnet. Les techniques d'attaques/infections de l'utilisateur sont décrites à la section 2.5. Le terme *bot* désigne l'ordinateur infecté et le programme malicieux, à savoir le malware. L'ensemble des bots, contrôlés par le botmaster, sont soumis aux commandes d'activités malicieuses. Ainsi, la dangerosité d'un botnet est définie par sa taille ou le nombre de zombies. Techniquement, sur base d'instructions du botmaster ou périodiquement, chaque bot télécharge et met à jour la version de son code dans le but de renforcer sa résistance face aux techniques de détection, d'ajouter de nouvelles

2. Le keylogging est l'enregistrement de l'activité du clavier d'une victime.

méthodes d'attaques, d'améliorer les techniques d'exploitation, ... C'est grâce à cette technique que les botnets continuent d'évoluer une fois mis en place.

3. Serveurs C&C : sont les intermédiaires via lesquels le botmaster contrôle le botnet. Cela en fait un composant critique dans la lutte contre les botnets. De fait, si l'on trouve un C&C serveur, on trouve le botmaster. Par conséquent, une des techniques est d'instaurer plusieurs couches de proxies afin de brouiller les pistes des autorités et les empêcher de prendre le contrôle du Botnet pour le couper définitivement. Selon la topologie de botnet, les serveurs C&C peuvent stocker des malwares et les rendre téléchargeables pour les bots. La topologie définit également l'endroit où les serveurs C&C sont placés et la manière dont ils sont configurés. Par exemple, si le botnet utilise le protocole Internet Relay Chat (IRC) pour communiquer, le serveur C&C peut être installé sur un serveur IRC. Par contre si le botnet utilise le protocole Hypertext Transfer Protocol (HTTP), le serveur C&C est un serveur Web.
4. Canal C&C : utilise différents protocoles de communication pour distribuer les instructions des serveurs C&C vers les bots. Le canal de communication joue un rôle important dans l'existence du botnet et dans ses attaques. Il existe deux types de canaux :
   (a) *push* : dans ce mode, les bots attendent que les serveurs C&C les contactent avec des instructions ;
   (b) *pull* : dans ce mode, les bots contactent périodiquement un des serveurs C&C pour recevoir des instructions.

   Le protocole chiffré IRC était largement utilisé par les premiers botnets. Cependant, aujourd'hui IRC est de moins en moins utilisé dans les réseaux, ce qui en fait un protocole inhabituel. En conséquence, la tendance est à abandonner ce protocole pour des méthodes de communication agiles et plus sophistiquées comme HTTP, Peer-2-Peer (P2P) ou encore DNS. Dans le cadre de ce mémoire, nous nous focalisons sur le DNS pouvant être exploité comme décrit dans la section 2.3. De plus, les botnets peuvent également utiliser les réseaux sociaux pour cacher des instructions. Enfin, les botnets mettent en place des clés publiques pour identifier les C&C et ainsi empêcher d'autres entités d'en prendre le contrôle.
5. *Proxies* : est une couche intermédiaire entre des bots et des serveurs C&C ayant pour objectif de cacher l'identité des serveurs C&C. Pour augmenter la complexité, plusieurs couches peuvent être configurées. Bien qu'elles soient optionnelles, ces couches permettent à un botnet d'échapper à l'identification des serveurs C&C et ainsi à leur neutralisation.

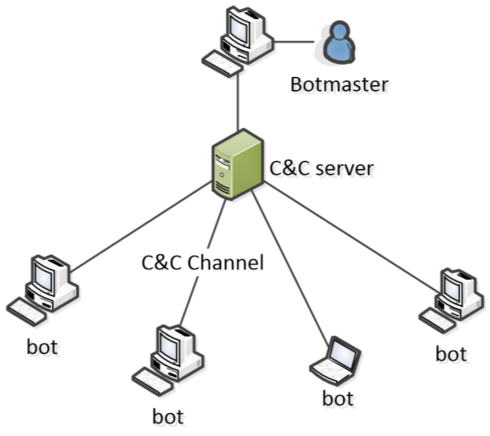

FIGURE 2.3 – Topologie standard d'un Botnet [31]

2.2.4 Topologies

Un botnet de taille moyenne contient de 1000 à 2000 bots (2.2.1). Pour assurer le contrôle et la sécurité du botnet, son infrastructure doit être robuste. C'est pourquoi les topologies ont été optimisées avec le temps. La sélection de celle-ci pour un botnet dépendra des risques perçus et de son modèle business malicieux. Les différentes topologies rencontrées sont typiquement : *star*, *multi-server*, *hierarchical*, *random*.

Star

Dans une topologie en *star* ou étoile, les bots communiquent de manière directe avec un unique C&C centralisé. Dès lors, lorsqu'une nouvelle victime est infectée, la première instruction pré-configurée dans le code du bot est de joindre le C&C. Une fois qu'elle a rejoint le botnet, elle attend les prochaines instructions. Cette topologie apporte une connexion rapide entre le serveur C&C et ses bots, mais peut être rapidement neutralisée si l'unique serveur est détecté. [17]

Multi-Server

La topologie multi-serveur est une extension de la topologie en étoile dans laquelle plusieurs serveurs C&C fournissent les instructions aux bots. Les serveurs communiquent entre eux et fournissent une solution de secours dans le cas où un des serveurs est supprimé. Cette topologie a pour avantage de disperser géographiquement ses serveurs C&C ce qui a pour effet d'augmenter la vitesse de communication avec ses composants. [17]

Hierarchical

Une topologie hiérarchique reflète les niveaux de propagation du botnet au travers des bots. Les bots ont la capacité de transmettre les instructions C&C via les niveaux supérieurs de leur hiérarchie de bots. Par conséquent, la communication subit des problèmes de latence et rend difficile son utilisation en temps réel. Ce modèle augmente la complexité en cas de détection des serveurs C&C mais aussi pour estimer la taille d'un botnet.[17]

Random

Il n'y a pas d'infrastructure C&C dans une topologie *random* ou aléatoire. En revanche, les commandes sont envoyées au botnet via un bot. Ce dernier reconnaît les commandes comme autoritaires car elles sont signées. Le bot sait alors qu'il doit propager la commande aux autres bots. Ce type de botnet est résistant aux techniques de détection et de coupure. Cependant, il est facile d'identifier un membre du botnet en surveillant le trafic d'un hôte infecté. [17]

2.2.5 Usages du DNS

Comme nous l'avons vu plus haut, le DNS est défini sur base des RFCs (voir chapitre 1). Pour échapper aux techniques de détection, les botnets manipulent cette infrastructure robuste et distribuée. En utilisant les noms de domaine, ils ont la capacité de se réorganiser facilement et rapidement. De fait, les serveurs C&C peuvent ainsi être migrés régulièrement d'une machine à une autre. Ils restent joignables en mettant à jour leur nom de domaine à l'aide de différentes techniques décrites ci-dessous. [17]

Fast-Flux

Aussi appelé *IP-Flux*, le *Fast-Flux* est une technique qui consiste, pour un nom de domaine particulier, à changer fréquemment d'adresse IP. L'idée est, pour les botmasters, de profiter de la flexibilité du DNS pour lier une liste d'adresses IP à un nom de domaine. Ensuite cette liste est constamment mise à jour afin d'échapper à toute détection. [17]
Lorsque le *Fast-Flux* est combiné aux *proxies* cela en fait une architecture résistante aux détections. Il existe deux types de *Fast-Flux* :

1. **Single-Flux** : enregistre et retire dynamiquement des centaines, voire des milliers d'adresses IP associées à un nom de domaine (à l'aide de RR de type A, voir section 1.3.3). Cette technique utilise l'association de deux méthodes : la distribution d'adresse IP par Round-Robin et l'utilisation de faibles valeurs Time To Live (TTL). Une fois de plus, ces techniques sont bien connues du DNS car elles sont utilisées par des services légitimes comme le Round-Robin DNS (RRDNS) et le CDN (voir section 1.5). La figure 2.4 (a) représente la résolution du nom de domaine *flux.example.com*. Le TLD *.com* renvoie la référence vers le serveur autoritaire **.example.com* souvent hébergé chez un Registrar ne respectant pas la charte d'Internet. Ce dernier répond au client avec une adresse IP de sa liste.
2. **Double-Flux** : est une évolution du *Single-Flux* dans laquelle les adresses IP des serveurs de noms (RR de type NS, voir section 1.3.3) sont aussi mises à jour régulièrement et pointent vers un bot. La figure 2.4 (b) montre le serveur de noms *ns.example.com* (un bot ou zombie) recevant une résolution. Ce bot est alors en communication avec un *backend* serveur (C&C serveurs) afin de résoudre le nom de domaine *flux.example.com* et renvoyer la réponse au client.

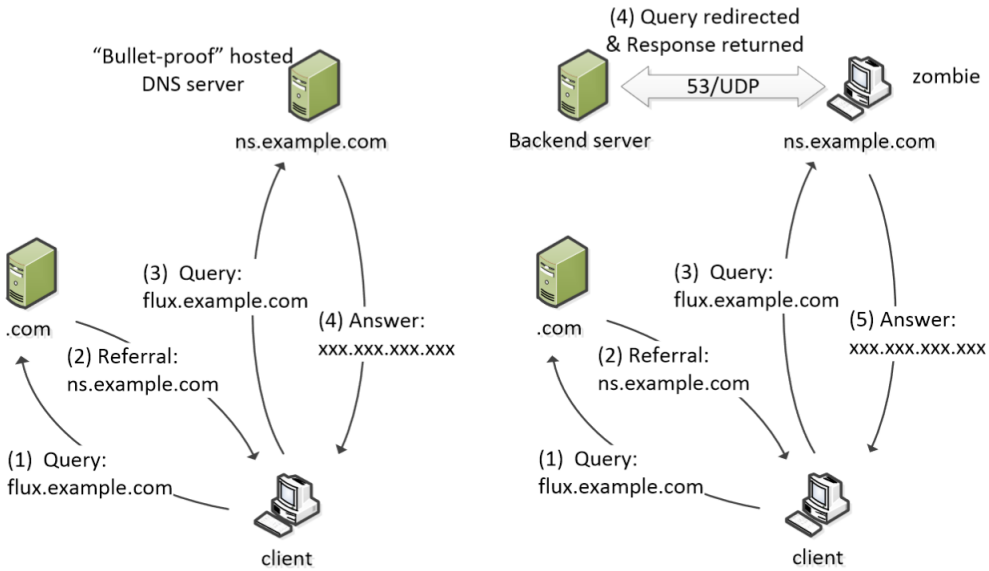

FIGURE 2.4 – Différence entre le *Single-Flux* et le *Double-Flux* [32]

Domain-flux

La technique du *Domain-Flux* consiste à allouer une liste de FQDN à une adresse IP ou une infrastructure C&C. Cette liste est alors constamment mise à jour afin d'empêcher toute détection de suspicion. [17] Typiquement, le bot peut soit posséder une liste de FQDN soit générer cette liste. Le botmaster possède la même liste et n'enregistre qu'une partie des noms de domaine. L'idée pour le bot est, de résoudre un des noms de domaine qu'il possède et ainsi contacter le serveur C&C. [32]
Il existe deux techniques de *Domain-flux* :

1. **Domain Wildcarding** : utilise le *masque* ou *wildcard* étant une fonctionnalité du DNS. De fait, le DNS permet de généraliser les sous-domaines d'un nom de domaine comme par exemple **.unamur.be*. Tout sous-domaine, tel que qwrirtg.unamur.be ou akljtfe.unamur.be, pointerait alors vers la même adresse IP, celle configurée pour **.unamur.be*.
2. **Domain generation algorithms (DGA)** : génère une liste dynamique de FQDN. Ces derniers sont alors parcourus par le bot qui essaye de les résoudre. Si la résolution réussit pour un FQDN, alors le bot obtient l'adresse IP de l'infrastructure C&C. La liste générée est de courte durée (24 heures), ce qui rend impossible le blocage de tels domaines.

Cependant, ces techniques ont des caractéristiques communes qui peuvent être utilisées dans notre recherche :

— Le nombre de noms de domaine généré doit être suffisamment grand pour sortir de l'ensemble des blacklists. (voir section 4.4.2) ;
— Les noms de domaine générés ne doivent pas exister dans le DNS ;
— Le bot génère des requêtes de résolution de noms n'étant pas dans le DNS, ce qui peut être douteux dans un réseau ;
— Les noms de domaine générés sont souvent imprononçables.

URL-Flux

La technique du *URL-Flux* utilise les services des sites Web 2.0 [3] comme moyen de communication. Un peu comme le domaine-flux, le bot génère un liste de noms d'utilisateurs. Pour récupérer une instruction du botmaster, le bot essaye de se connecter à un site web 2.0 en s'authentifiant avec une clé publique. Si l'authentification réussit avec l'utilisateur, alors le bot peut récupérer l'instruction. Cette technique fait place à une recherche hors du sujet de ce mémoire car sa détection ne se base pas sur du trafic DNS mais plutôt sur du trafic HTTP. [14]

2.3 Exploitation du DNS

Depuis leurs apparitions, les malwares utilisent différents stratagèmes pour entrer en contact avec leur infrastructure C&C. Comme nous l'avons vu, plusieurs protocoles sont utilisés : IRC, HTTP, P2P et le protocole DNS sur lequel nous nous concentrons dans ce travail. À l'aide du DNS, il existe deux techniques clés pour exfiltrer des données : le DNS Signaling et le DNS Tunneling.

3. Les services du Web 2.0 sont par exemple les blogs, les microblogs, *Google App Engine*, ... Un des microblogs les plus connu est *Twitter*

DNS Signaling

Dans cette technique, un malware logé sur un système envoie des requêtes DNS de type TXT ou génère des requêtes pour résoudre un nom d'hôte lié à un domaine C&C (un domaine malicieux). Le domaine C&C peut être codé en brut ou alors généré via un DGA, par exemple *49203498-randomstuff-12aklsdf23a9809u.evil-c2-domain.com.* Ce type de requête permet à un malware d'effectuer plusieurs tâches :

— envoyer des mises à jour ;
— réaliser une demande particulière ;
— exfiltrer des données de la victime.

Ces requêtes apparaissent comme anodines mais sont gérées et redirigées vers le serveur de noms autoritaire du domaine malicieux *evil-c2-domain.com.*. Le propriétaire de ce dernier peut alors interpréter la requête pour ensuite formuler une réponse ou encore un RR TXT qui informe le malware de ce qu'il doit faire. Le malware est indépendant sur un système, il peut configurer son propre serveur de noms et ne plus passer par le serveur de noms du système local. En conséquence, la victime est déconnectée du serveur DNS de son organisation. [54]

DNS Tunneling

Cette deuxième technique permet d'encapsuler un autre protocole (HTTP, FTP,...) dans des requêtes DNS. Elle repose sur le même principe qu'une session Virtual Private Network (VPN). L'avantage pour un botnet d'utiliser cette technique est d'échapper aux signatures d'Intrusion Detection System (IDS) [4] ou d'Intrusion Prevention System (IPS) [5] et autres systèmes de monitoring. Le trafic malicieux est ainsi véhiculé dans un protocole inoffensif vers des serveurs légitimes du monde entier. De plus avec IPv6, DNSSEC, et d'autres extensions du DNS, les requêtes DNS sont basées sur le protocole TCP et sont de plus en plus volumineuses. Par conséquent, le volume d'une requête DNS n'est plus un critère adéquat pour détecter une requête malicieuse. [54]

2.4 Attaques serveurs DNS

Quelle que soit la technique d'attaque, l'attaquant poursuit toujours les mêmes objectifs. Les plus connus sont : mettre la victime en confiance alors qu'elle visite un site frauduleux, installer un malware, voler des données, ... en d'autres termes, prendre le contrôle de sa machine à son insu.

Une des attaques les plus répandues est le *DNS Spoofing*. Elle consiste à modifier un RR se trouvant sur un serveur DNS. Après quoi, la résolution du nom de domaine corrompu redirigera la victime vers un site frauduleux. [43]

Il existe plusieurs types d'attaques :

— le DNS *cache poisoning* : est la méthode la plus utilisée. Elle consiste à ajouter un RR sur le serveur DNS autoritaire en utilisant une technique de prédiction de l'Identificant de transaction en conjonction avec le port source UDP de la requête ;[11]
— la compromission de serveur DNS : est une attaque dans laquelle les attaquants prennent le contrôle du serveur DNS Autoritaire. Les attaquants peuvent alors modifier les RRs, ce qui redirige les utilisateurs vers des sites frauduleux ; [53]
— une attaque *Man-in-the-middle* : arrive lorsque l'attaquant réussit à s'intercaler dans une communication entre, la plupart du temps, un client et un serveur. Le trafic est alors intercepté par l'attaquant qui modifie ou injecte des informations. Le client est

4. Détecte des évènements pouvant révéler des intrusions sur un réseau.
5. Agit comme un pare-feu. L'IPS prend la décision de bloquer du trafic si celui-ci représente une menace.

ainsi redirigé vers un serveur web malicieux à son insu. [48]

L'attaque des serveurs DNS sort du contexte de ce mémoire. Cependant, afin de comprendre la présence de trafic malicieux dans un réseau, il s'avère utile de situer cette étape dans le processus d'attaque.

2.5 Attaques utilisateurs

Dans le cadre de détection de trafic malicieux, il est judicieux d'analyser une capture réseau afin de repérer des connexions à partir de clients locaux vers des serveurs pouvant stocker des malwares. Pour encourager les victimes à installer des malwares, les attaquants utilisent les techniques suivantes : le phishing, le pharming, l'exploitation des vulnérabilités du système.

Le Phishing

Le principe du phishing est l'envoi d'e-mails reprenant un look similaire aux grandes organisations connues et comprenant un lien guidant la victime de l'e-mail vers un site frauduleux. Le but de l'attaquant est de voler les données privées de la victime. [52]

Le Pharming

Le *pharming* est une cyber-attaque durant laquelle une personne malintentionnée oriente le trafic destiné à un serveur web vers un serveur malicieux. Invisible pour l'utilisateur qui entre l'URL correcte, cette pratique est indétectable par les antivirus et les logiciels de détection d'espion. Il existe deux manières de mettre en place l'attaque. La première est par le *DNS cache poisoning* (détaillée dans le point 2.4) et la seconde est par l'*URL hijacking*, typiquement mise en place par un trojan. Cette dernière modifie les marques pages du navigateur web ou intercepte les requêtes DNS et les modifie. Le but du hacker est de voler les données privées de l'utilisateur. Dans la plupart des cas, cette pratique cible principalement les PC car ceux-ci manquent de sécurité et d'administration. [7]

Exploitation des vulnérabilités du système

Une autre technique d'attaque est l'exploitation des vulnérabilités des systèmes. Les malwares se propagent au travers des réseaux via les vulnérabilités des systèmes d'exploitation. Comme nous avons vu dans la section 2.1, le worm scanne constamment le réseau afin de contaminer les autres systèmes. [32]

2.6 Limitations du protocole DNS

Comme nous le mentionnons dans le chapitre 1, le protocole DNS a été défini dans deux RFCs principales (RFC1034 et RFC1035), puis a subi des améliorations avec le temps. À l'origine en 1980, le protocole DNS a été conçu pour la résolution de noms de domaine sans tenir compte des aspects de sécurité. Cependant, avec l'évolution des technologies, le respect de la vie privée et les dernières techniques d'attaques (voir 2.4 et 2.5), deux facteurs importants sont à considérer.

Le premier est l'intégrité des données transmises entre les éléments du protocole DNS ; celle-ci est représentée par un cadenas ouvert sur les serveurs de noms dans la figure 2.5.

Le deuxième facteur est la protection des données privées transmises par l'utilisateur, représenté dans la figure 2.5 par un cadenas sur chaque communication entre les éléments du modèle.

De plus, comme le montre la figure 2.5, le DNS comprend deux types d'interactions. D'une part, l'interaction entre le *Stub Resolver* et le *Recursive Resolver* et, d'autre part l'interaction entre le *Recursive Resolver* et les *Serveurs Autoritaires*.

Malgré cette complexité trois protocoles répondent à cette problématique.

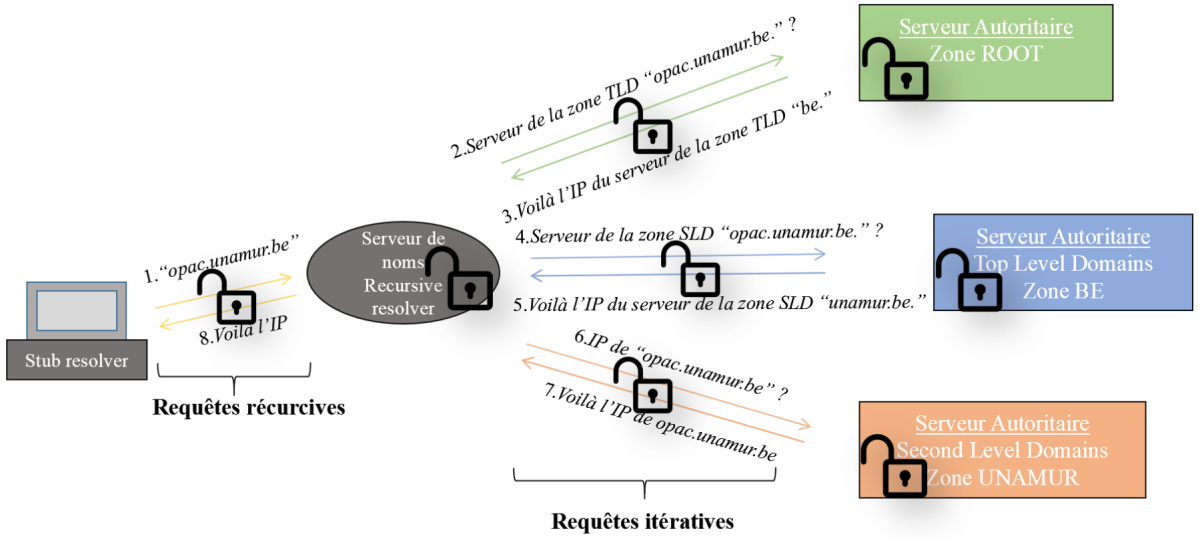

FIGURE 2.5 – Limitations en terme d'intégrité et de privatisation des données du protocole DNS.

2.6.1 DNS Security Extensions - DNSSEC

Nous ne pouvons pas mentionner la vulnérabilité du protocole DNS sans faire une parenthèse sur ses extensions de sécurité. Avec le temps, le protocole DNS a vu ses limites faire surface et des adaptions ont du être apportées. C'est pourquoi un ensemble de RFCs (RFC4034, RFC4025, RFC2535, RFC2930, RFC2230, RFC4255) sont venus compléter les RFCs standards, ce qui a permis d'adapter le protocole face à ses vulnérabilités comme le DNS spoofing. L'intérêt du DNSSEC n'est pas de chiffrer les données mais plutôt de vérifier la source des données reçues. De cette manière les éléments du modèle client-serveur recevant des informations DNS peuvent garantir deux réalités. La première est l'identité de l'émetteur grâce à sa signature [6]. La deuxième est l'intégrité des données grâce à la signature [7] des données par l'émetteur. L'application du DNSSEC est facultative mais l'ICANN espère voir de plus en plus de *resolvers* et de serveurs autoritaires adopter ce protocole.[34]

6. Une signature est le même principe qu'une signature d'un document par une personne. Elle est unique pour chaque personne et peut être vérifiée

7. L'émetteur possède deux clés. La première est une clé privée, lui permettant de signer les données et de générer une signature digitale. La deuxième est une clé publique envoyée au destinataire et sert à la validation de la signature digitale. Si elle est valide le destinataire accepte l'information, sinon il l'ignore

2.6.2 DNS over HTTPS - DoH

DNS over HTTPS (DoH) est un protocole défini dans le RFC8484, permettant au protocole DNS de communiquer par l'intermédiaire de la couche HTTPS (RFC2818) et donc du protocole Transport Layer Security (TLS). Chaque requête/réponse est liée à un échange HTTP, et bénéficie des caractéristiques de ce dernier (cache, redirection, proxy, authentification, compression définis dans RFC7230, RFC7237). DoH vise à répondre à la nécessité de chiffrer le trafic DNS circulant sur les réseaux entre les clients (*stub resolver*) et les résolveurs récursifs (voir figure 2.5). De fait, DoH peut être intégré de différentes manières [8]. Cependant, le protocole vise principalement les *web browsers*, vu la facilité d'intégration côté client. [62]

2.6.3 DNS over TLS - DoT

DNS over TLS (DoT), défini dans le RFC7858, est un protocole permettant de créer une communication chiffrée avec un résolveur récursif. De fait, dans la section DoH (2.6.2), nous expliquons le canal de communication chiffré entre le *stub resolver* et le serveur récursif incluant le protocole HTTPS. Cependant, les serveurs autoritaires ne possèdent pas le protocole HTTPS dans leur stack vu la nature de leur rôle. C'est pourquoi, DoT apporte une solution plus légère reposant uniquement sur le protocole TLS. Récemment, plusieurs mises en application ont été réalisées par de grandes organisations pour montrer la stabilité du protocole [9]. Il est à noter que cette solution convient également pour la communication entre le *Stub Resolver* et le résolveur récursif. [62]

8. https ://github.com/curl/curl/wiki/DNS-over-HTTPS#doh-tools
9. DNS over TLS : Encrypting DNS end-to-end https ://code.fb.com/security/dns-over-tls/

Chapitre 3

Études existantes

Dans les deux chapitres qui précédent, nous venons de parcourir l'existant en termes de protocole DNS et de trafic malicieux. Il paraît judicieux de compléter cette approche par une synthèse des études existantes dans ce domaine. Le but est d'illustrer notre propos plutôt qu'inventorier l'ensemble des analyses faites. L'orientation choisie vise à éclaircir les pistes d'études en cours.
L'analyse du DNS devient clairement un des facteurs principaux dans la détection de trafic malicieux au sein des réseaux. Nous commençons par évoquer la définition du trafic malicieux, ensuite nous mentionnons les défis auxquels nous sommes confrontés dans sa détection pour en présenter un type d'approche. Puis, nous décrivons comment le passive DNS (pDNS) peut répondre au manque d'historique dans le DNS, une de ses limites. Enfin, il existe de multiples recherches disponibles sur Internet consacrées à l'étude de la détection de trafic malicieux. Nous évoquons celles qui nous permettent de mieux comprendre ce qui est considéré comme malicieux, les techniques utilisées, les outils mis en place et les résultats obtenus. Nous tâchons de reprendre les recherches impliquant les botnets et leurs modi operandi comme décrit au chapitre 2. Enfin, les différents outils utilisés sont mis en évidence comme le *pDNS*, le machine learning, les blacklists 4.4.2, ... ceux-ci sont également détaillés dans le chapitre 4.

3.1 Définition du trafic malicieux

La définition du trafic malicieux prend tout son sens au vu de la diversité du trafic que l'on peut trouver dans un réseau. Dans le cadre de nos expériences, il est important de le définir pour éviter toute confusion. Ainsi, le trafic malicieux est défini comme toute activité réseau étant en relation avec du phishing, spamming, propagation de malware, botnet, ou tout autre trafic dérivé de ces dernières (voir section 2.5). [46] Par contre, nous ne considérons pas comme trafic malicieux les sites avec du contenu illégal, tels que les sites à caractère "pédophile", de "radicalisation", "pédo-pornographique", ... mais aussi les sites restreints à une certaine tranche d'âge, tel que le contenu "pornographique". Dans notre recherche, l'activité est centralisée uniquement sur le protocole DNS.

3.2 Défis de la détection du trafic malicieux

Comme nous l'avons vu dans le chapitre précédent, un botnet est un réseau de bots restant aléatoirement en interaction. Cependant, il doit s'adapter aux contraintes des nouvelles technologies de détection d'intrusions comme les IDS ou encore les anti-virus. Pour contourner ces systèmes de détection, les botnets mettent en place des techniques astucieuses en tirant profit de protocoles dont le protocole DNS. Ces techniques, comme le *Fast-Flux* ou *Domaine-Flux*

listées dans la section 2.2.5, représentent des défis majeurs pour les organismes/départements de sécurité. Pour contrer ces techniques, une multitude d'études sont disponibles, proposant des solutions différentes. Parmi ces études, ce qui est principalement utilisé est une capture réseau, des listes de références (*blacklist, whitelist* section 4.4.2) et du *machine learning* (chapitre 5). L'idée principale est tout d'abord de labéliser les noms de domaine ou adresses IP de la capture répertoriés comme malicieux ou légitime, à partir de listes de références. Ensuite, une analyse des requêtes et réponses des techniques des botnets est réalisée. Sur base de cette analyse, une liste de leurs caractéristiques ou *features* qui les discriminent du trafic légitime est construite. À titre d'exemple, la figure 3.1 présente une liste de features (voir section 6.5.3). Nous y retrouvons, entre autres, le *nombre d'adresses IP uniques dans la réponse*, nous rappelant les techniques de *Fast-Flux*. Ou encore les statistiques sur les noms de domaine, comme le *nombre de caractères numériques*, *la longueur d'un nom de domaine*, ... caractérisant les noms de domaine générés (DGA). Enfin, l'ensemble des caractéristiques est calculé sur les données contenues dans la capture pour entraîner des algorithmes de machine learning. Finalement, ces algorithmes sont testés et évalués afin de permettre la détection de nouvelles requêtes malicieuses.

Caractéristiques ou features
nombre d'adresses IP uniques dans la réponse
moyenne du TTL
minimum du TTL
maximum du TTL
une réponse NXDOMAIN indique que le nom domaine n'existe pas
longueur du FQDN
longueur du nom de domaine
nombre de caractères numériques dans le nom de domaine
nombre de caractères non alphanumériques dans le nom de domaine
nombre de sous-domaines
nombre de traits d'union dans le FQDN
longueur du plus long sous domaine
nombre de voyelles dans le FQDN
nombre de différents caractères dans le FQDN
nombre de consonnes dans le FQDN

FIGURE 3.1 – Exemple de caractéristiques ou *features*

3.3 Le passive DNS - pDNS

Les serveurs DNS ne retiennent aucune information excepté celles stockées dans leur cache, mais leur existence est limitée par le TTL (voir 1.3.5). De fait, sur un serveur de noms, si le TTL d'un resource record (RR) arrive à expiration, ce RR est supprimé définitivement du serveur. Si à la prochaine requête ce RR est modifié par son serveur autoritaire, alors le serveur de noms rapatriant ce RR contiendra uniquement les nouvelles informations. Ce système apporte une limitation et pose problème en cas de nécessité d'analyse de l'historique. Pour répondre à cette problématique, Florian Weimer [61] présente une technique nommée pDNS. Le principe est simple : toutes les requêtes et réponses (appelées flux) DNS sont enregistrées et stockées dans une base de données. Ces enregistrements représentent alors un historique de toutes les informations DNS passées ce qui en fait une source importante dans l'analyse et la détection d'activités malicieuses. Cela permet, entre autres, de retrouver toutes les adresses IP associées à un nom de domaine. Dans le cas où cette liste serait excessive, elle pourrait indiquer la présence de Fast-Flux. Le pDNS donne naissance à des features utiles dans la discrimination du trafic malicieux par rapport au trafic légitime. Dans la figure 3.1, la feature *moyenne du TTL* est typiquement créée à partir de pDNS.[61] De plus, le pDNS permet d'analyser le trafic malicieux sans être repéré par le propriétaire du domaine suspect. De fait, cette technique n'émet aucune requête vers les noms de domaine malicieux. Au contraire, les sites hébergeant des *blacklists*

(section 4.4.2) émettent régulièrement des requêtes de manière à tenir leur base de données à jour.[46]

3.4 Les études

Building a Dynamic Reputation System for DNS

M. Antonakakis et al. [46] proposent un système, *Notos*, ayant pour objectif de classer à l'aide d'algorithmes de machine learning un nom de domaine malicieux ou légitime. Le système assigne à chaque nom de domaine d'une requête DNS un score de réputation indicateur de sa classe. Ainsi, plus le score est faible plus la requête est suspectée d'être classée malicieuse. Dans leur recherche, ils commencent par évoquer une des caractéristiques du célèbre *worm Conficker*[1]. Après l'infection de la machine, le *bot* essaye de contacter son infrastructure C&C en sélectionnant une série de noms de domaine dans une liste de plus de 50 000 entrées créée dynamiquement (voir *domain-flux* et plus particulièrement la technique du DGA 2.2.5). Ce type de botnet, en utilisant cette technique, vise clairement à contrer l'efficacité des blacklists (voir 4.4.2). De fait, les blacklists sont des listes statiques régulièrement mises à jour. Cependant avec l'apparition des techniques comme le *Fast-Flux*, le *Domain-Flux*, *URL-Flux* cela devient de plus en plus difficile de les maintenir à jour. À cette fin, le système *Notos* est construit sur plusieurs étapes :

1. l'utilisation de jeux de requêtes pDNS ;
2. pour chaque nom de domaine *d*, le système collecte deux listes. La première est la liste des IPs ayant appartenu à *d*. La deuxième est la liste des noms de domaine ayant appartenu aux IPs ;
3. à partir de ces deux listes, trois groupes de features [2] sont calculés : réseaux (18 features), zone (17 features), d'indication (6 features) ;
4. la collection de ces features est alors utilisée pour entraîner un algorithme de classification ;
5. enfin, l'algorithme attribue un score de réputation pour tout nouveau nom de domaine.

Finalement, le système est capable de classer un nouveau nom de domaine avec un taux de False Positives (FP%) [3] de 0.38% et un taux de True Positive (TP%) [4] de 96,8%. Leur configuration est un arbre de décision utilisé avec le top 500 d'Alexa (voir *whitelist* 4.4.2), 9530 noms de domaine malicieux connus, un cross validation configuré à 10 et un seuil de détection de score configuré à 0.5. Les auteurs montrent qu'ils obtiennent de meilleurs résultats en sélectionnant un nombre plus restreint de noms de domaine d'Alexa. Cela est probablement dû au fait que le top 100 000 de la whitelist (Alexa) comprend 5% de noms de domaine malicieux repris dans des blacklists. Enfin, le système réussit à assigner un mauvais score aux noms de domaine en relation avec des malwares et cela avant qu'ils n'apparaissent dans les blacklist.[46]

En conclusion, sur base d'une quantité importante de données DNS (pDNS) l'idée est de construire un historique des résolutions des noms de domaine. Ensuite, le système discrimine, le trafic malicieux du légitime, à l'aide de caractéristiques et de listes de références (blacklists et whitelists). Enfin, la dernière étape applique des algorithmes de machine learning afin de

1. Conficker est un *worm* célèbre apparu dans les années 2000
2. feature est le terme Anglais traduit par caractéristique. Le terme feature est largement utilisé dans le monde du Machine Learning
3. Un domaine malicieux est classé comme domaine légitime par le système
4. Un domaine malicieux est classé comme domaine malicieux par le système

classer un nom de domaine suivant qu'il soit malicieux ou non. Finalement, les auteurs soulignent les limitations de *Notos* dont les deux suivantes. La première est que le système a besoin d'énormément de données pour être précis. De fait, un site hébergé chez un fournisseur hébergeant des sites connus comme malicieux sera pénalisé, et donc classifié comme potentiellement malicieux. La deuxième concerne le manque de la notion de temps minimum requis, pour avoir une précision acceptable.

EXPOSURE : Finding Malicious Domains Using Passive DNS Analysis

Exposure [45] est un système de détection de noms de domaine malicieux semblable au système *Notos*. À partir de données pDNS, L. Bilge et al. [45] utilisent 15 features reparties en 4 catégories (figure 3.2) afin de caractériser les différences entre les noms de domaine mais aussi la manière dont ils sont utilisés (ex : paramètre de requête, temps de vie d'un Resource Record). À la différence de *Notos*, le système *Exposure* requiert moins de données pDNS. De plus, il détecte les noms de domaine associés à une nouvelle adresse IP pour une courte durée et qui ne sera jamais plus réutilisée. Ensuite, sur base de *blacklists* et *whitelists* (décrites ci-dessous), le système labélise un maximum de requêtes DNS. Les données labélisées sont alors utilisées pour entraîner un algorithme de classification (un arbre de décision) tandis que les données non-labélisées utilisent l'algorithme pour être classées (figure 3.3). Finalement, c'est sur base d'une collection de deux mois et demi de données représentée par 300 000 noms de domaine distincts, que *Exposure* classe 17686 domaines comme malicieux ; soit 5.895%. Cette collection, de 300 000 noms de domaine, dérive en réalité de la capture à l'origine de 4.8 millions de noms de domaine distincts. Cependant, 4.5 millions de noms de domaine ont été retirés car le nombre de requêtes totales pour chacun d'entre eux était inférieur à 20 sur les deux mois et demi de capture. Après analyse, les auteurs obtiennent un FP de 7.9% [5] soit 1408 domaines. Il est à noter que les données DNS reçues par les auteurs sont privatisées dans le sens où les informations de la source initiant la requête ne sont pas dévoilées. [45]

Feature Set	#	Feature Name
Time-Based Features	1	Short life
	2	Daily similarity
	3	Repeating patterns
	4	Access ratio
DNS Answer-Based Features	5	Number of distinct IP addresses
	6	Number of distinct countries
	7	Number of domains share the IP with
	8	Reverse DNS query results
TTL Value-Based Features	9	Average TTL
	10	Standard Deviation of TTL
	11	Number of distinct TTL values
	12	Number of TTL change
	13	Percentage usage of specific TTL ranges
Domain Name-Based Features	14	% of numerical characters
	15	% of the length of the LMS

FIGURE 3.2 – Liste de features utilisées dans *Exposure* [45]

5. Un domaine légitime est classé comme domaine malicieux par le système

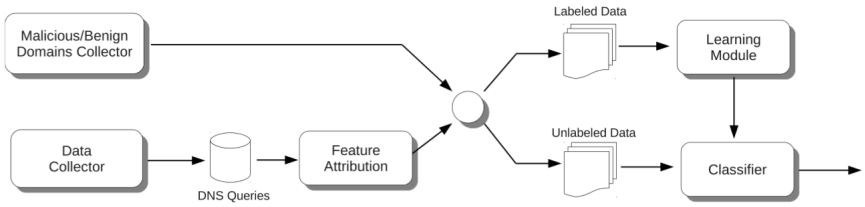

FIGURE 3.3 – Pipeline d'*Exposure* [45]

Blacklist

Pour construire la *blacklist* (voir 4.4.2), ces mêmes auteurs[45] utilisent des données d'activités malicieuses comme C&C, sites de téléchargement, des pages de phishing, des sites d'escroqueries.
Sources de domaines malicieux :

- *Malware Domains List*[6] ;
- *The Zeus Block List* ;[7]
- *Malware Domains List* ;[8]
- *Anubis reports.*

Sources d'URLs malicieuses :

- *Wepawet*[9] ;
- *Phishtank*[10].

Listes de domaines générées par la technique de botnet DGA :

- *Conficker* ;
- *Mebroot.*

Vu la présence de 3LD (*"serveur1.exemple.com"*) dans les sources d'URLs malicieuses, il est nécessaire d'effectuer une vérification avant de définir le SLD (*"exemple.com"*) comme malicieux. De fait, si l'on considère un domaine d rapporté comme malicieux sur base des sources URLs *Wepawet* ou *Phishtank* ; alors si D_{tot} et D_{mal} où, D_{tot} est le total des 3LDs de d recueillis par *Exposure* et D_{mal} est le nombre de 3LDs malicieux sur base des données d'*Exposure* (voir figure 3.4), alors il considère d comme malicieux si $D_{mal}/D_{tot} > 0.75$ où plus de 75% des 3LDs sont malicieux. Ils obtiennent ainsi un jeu de données de 3500 domaines malicieux pouvant être injectés dans le système *Exposure*.

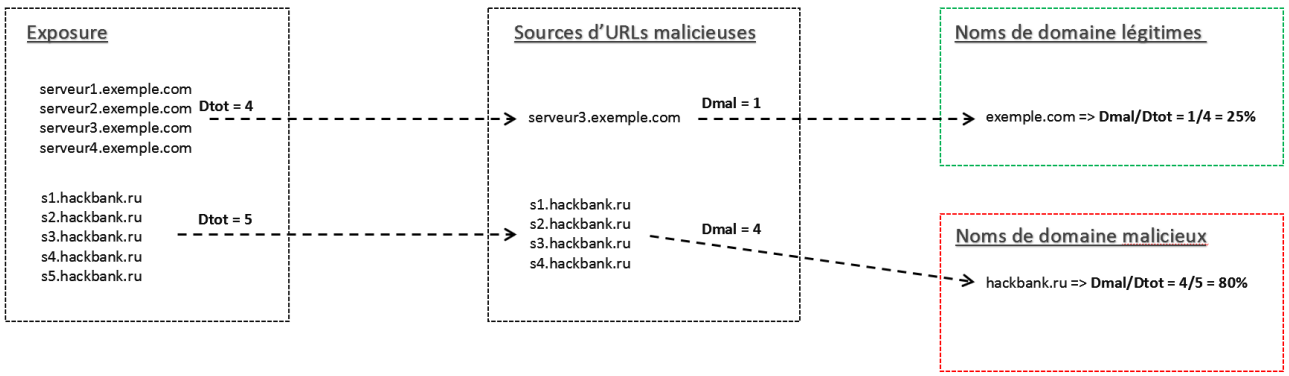

FIGURE 3.4 – Labélisation de jeux de données par *Exposure*

6. http ://www.malwaredomains.com/
7. https ://zeustracker.abuse.ch/
8. http ://www.malwaredomainlist.com/
9. http ://wepawet.iseclab.org/
10. http ://www.phishtank.com/

Whitelist

La construction de la *whitelist* (voir 4.4.2) est composée du top 1000 Alexa (voir *whitelist* 4.4.2) et d'une liste regroupant les noms de domaine créés depuis plus d'un an. Une fois la liste obtenue, les auteurs effectuent deux vérifications. La première est la comparaison de la *whitelist* avec la *blacklist* construite mais également avec des outils de tests en ligne comme *McAffee Site Advisor*[11] et *Norton Safe Web*[12]. La deuxième est le croisement des données avec un annuaire du site web *Open Directory Project (ODP)* géré entièrement par une vaste communauté. Aujourd'hui ce projet n'existe plus mais est remplacé par le projet Curlie, "le plus grand répertoire du Web édité par des humains. Il est construit et entretenu par une communauté mondiale passionnée de rédacteurs bénévoles."[15]. La *whitelist* obtenue contient 3000 domaines pouvant être injectés dans le système *Exposure*.

En conclusion, *Exposure* est un système semblable à *Notos* mais détectant davantage de noms de domaine malicieux grâce à l'utilisation des caractéristiques du temps (*Time based features* figure 3.2). De plus, comparé à *Notos*, *Exposure* utilise d'autres features et n'a pas besoin de retenir les adresses IP et les noms de domaine dans une base de données. Ils soulignent également le mécanisme de gestion d'URLs malicieux mis en place pour recourir à la limitation d'informations dans le protocole DNS. Ensuite, les auteurs de *Exposure* signalent que comparé à *Notos*, *Exposure* a besoin d'une période fixe de sept jours pour être entraîné. Finalement, les auteurs soulignent les limitations de *Exposure* dont les deux suivantes : la première est la présence obligatoire de trafic malicieux dans le jeu de données pDNS afin de permettre à l'algorithme de détecter ce type de trafic. La deuxième est dans le cas où l'attaquant analyse le code source ou le fonctionnement de *Exposure* ; il pourrait alors contourner les techniques de détection mises en place.

Detecting Malware Domains at the Upper DNS Hierarchy

Dans leur recherche M. Antonakakis et al. [42] développent un système de détection de noms de domaine, *Kopis*, en relation avec les malwares. Ils comparent les recherches de *Notos* [46] et *Exposures* [45] dans lesquelles les auteurs utilisent du trafic capturé à partir d'un nombre limité de serveurs de noms récursifs. Or, la limitation du nombre de serveurs considérés réduit la diversité du trafic. De fait, cela ne représente qu'une petite partie du trafic DNS appartenant à un serveur de noms récursif. C'est pourquoi, *Kopis* capture le trafic dans les couches supérieures de la hiérarchie du DNS, au niveau TLD (section 1.2.1) dans ce cas. En travaillant dans les couches supérieures, les auteurs capturent un plus grand nombre de requêtes et construisent un système capable de détecter les noms de domaine malicieux avant qu'ils ne se propagent dans le reste des couches inférieures. Les features de *Kopis* sont différentes de celles des deux études précédentes. De fait, intégrant un environnement différent dans la hiérarchie du DNS, le système calcule, en outre, des features sur les *recursive resolvers* en fonction de leur taille, celle-ci pouvant varier du particulier au *recursive resolver* d'une grande entreprise. De plus, *Kopis* a l'avantage d'être moins dépendant de la réputation d'adresse IP, ce qui n'est pas le cas de *Notos* et *Exposure*. Pour les auteurs, cet avantage est non négligeable au vu du déploiement futur de l'IPv6 et, par conséquent de l'accès à un plus grand nombre d'adresses IPs pour les attaquants. Dans leur conclusion, ils réussissent à obtenir un taux de False Positives [13] de 0.3%

11. Évalue les sites sur plusieurs attributs spécifiques `https://kc.mcafee.com/corporate/index?page=content&id=KB53369`
12. http ://safeweb.norton.com/
13. Un domaine malicieux est classé comme domaine légitime par le système

et de True Positive [14] de 98.4% pour les noms de domaine existant. La meilleure configuration est l'utilisation du modèle *Random Forest* avec le top 30 d'Alexa et de blacklists publics. Enfin, ils soulignent la capacité de *Kopis* de détecter des noms de domaine malware avant qu'ils n'apparaissent dans des blacklists. [42]

En conclusion, l'analyse de trafic DNS dans les parties supérieures de sa hiérarchie est judicieux afin de stopper la propagation de noms de domaine malicieux. De nouvelles features sont créées et adaptées à l'environnement dans lequel réside la détection. De plus, au vu du déploiement d'IPv6, les auteurs soulignent les faiblesses des systèmes de détection travaillant sur les réputations d'IPs comme *Notos* et *Exposure*. Dans leur système, les auteurs obtiennent les meilleurs résultats à l'aide du *Random Forest*. Finalement, les auteurs évoquent les limitations du système *Kopis* : étant donné le rôle du cache dans les couches inférieures le système ne voit pas régulièrement les requêtes de certaines délégations. Le système pourrait ne pas détecter un domaine malicieux si la résolution du domaine provient de différents *recursive resolvers* ou *stub resolvers* (souvent des machines infectées).

Botnet Detection Based On Machine Learning Techniques Using DNS Query Data

XD Hoang et QC Nguyen pointent l'évolution constante des botnets en terme de taille et de complexité. Les outils permettant leurs détections par signatures comme les IDS ne sont plus adaptés. De fait, lors de nouveaux botnets, leur base de données de signatures doit être mise à jour, ce qui donne aux malwares le temps de se propager. Cependant, plusieurs études montrent que les méthodes de détection d'anomalies dans un réseau sont de plus en plus efficaces. Elles ne nécessitent pas de base de données statiques, ce qui a l'avantage d'accélérer la détection de nouveaux botnets et ainsi de réduire le temps de réaction afin de les contrer. À cet égard, les auteurs proposent un modèle de détection de botnet par machine learning. Dans leur article, ils décrivent l'activité des botnets et soulignent l'utilisation de techniques comme le DGA ou le *Fast-Flux* (voir section 2.2.5). Ces techniques permettent, à l'aide du DNS, de changer constamment d'adresse IP ainsi que de nom de domaine afin d'échapper à tout outil de sécurité. Dans leurs analyses, les auteurs orientent leur recherche sur la structure lexicale des noms de domaine pour permettre de distinguer un nom de domaine malicieux d'un nom de domaine légitime. À cette fin, ils utilisent deux types de features : le premier se focalise sur des statistiques de caractères adjacents (nommé n-gram, où n est le nombre de caractères utilisés) et le deuxième concerne la distribution de voyelles. Une fois la liste de features réalisée, ceux-ci sont appliqués sur les données de la capture réseau. Ensuite, les auteurs utilisent quatre algorithmes de machine learning supervisés : *k-nearest neighbors (kNN)*, *decision tree*, *random forest* et *Naive Bayes*. Le but est alors d'entraîner les algorithmes, de les tester et comparer les résultats afin de mettre en avant la technique la plus précise. Dans le cas de la recherche de XD Hoang et QC Nguyen, l'algorithme *random forest* donne les meilleurs résultats avec un taux de False Postives [15] de 4.80%. [29]

En conclusion, les IDS deviennent obsolètes au vu du délai de diffusion des mises à jour de leur base de données. Les auteurs proposent un mécanisme de détection de noms de domaine malicieux par l'analyse de leur structure lexicale. L'algorithme *random forest* donne les meilleurs résultats avec un FP de 4.80%.

14. Un domaine malicieux est classé comme domaine malicieux par le système
15. Un domaine malicieux est classé comme domaine légitime par le système

Botnet Detection Using Passive DNS

À l'aide d'un classificateur, Pedro Marques de Luz [16] tente de répondre à une question : "Est-il possible de détecter des noms de domaine utilisés par des botnets ?". Au début de son travail, il décrit le DNS ainsi que les techniques de Round-Robin et CDN. Bien que ces techniques mettent en place un système DNS distribué et robuste, elles bénéficient malencontreusement au monde cyber-criminel. Les botnets font partie des systèmes exploitant ces techniques afin d'améliorer leur disponibilité et leur agilité. Tout d'abord, Pedro Marques de Luz choisit de se focaliser sur l'utilisation de *Fast-Flux* et le DGA par les botnets. À cet égard, il commence par utiliser les données du trafic DNS capturées pendant une semaine afin d'étudier leurs comportements. Ensuite, il compare le trafic malicieux et légitime pour mettre en évidence leurs caractéristiques et les répartit en deux groupes : les caractéristiques lexicales et les caractéristiques réseaux. À l'aide du machine learning, il met en place un classificateur sur base de 36 caractéristiques. Puis, pour entraîner les classificateurs, il construit différents *trainings set*[16] labélisés (malicieux, légitime) reposant sur une semaine de données pDNS. Afin de déterminer si un domaine est malicieux ou non, il utilise des blacklists et des whitelists comme référence. L'auteur souligne que pour récupérer un plus grand nombre de données de référence, il choisit de retourner, pour chaque SLD présent dans une blacklist ou whitelist, tous les 3LDs y correspondant. Enfin, à l'aide d'un *test set*[17] de deux semaines, il conclut qu'il est possible de classer un nom de domaine malicieux ou légitime à l'aide d'un classificateur *random forest* avec un taux de False Positives[18] et également de False Negatives[19] de 3%.

En conclusion, par rapport à l'étude précédente, Pedro Marques de Luz utilise des caractéristiques lexicales et des caractéristiques réseaux pour détecter les requêtes malicieuses. Les 36 features utilisées par l'auteur sont une bonne base sur laquelle des tests peuvent être réalisés. Ensuite, dans la construction du training set, il précise que du bruit peut y être inséré au vu de la généralisation des 3LDs en SLD. Finalement, l'auteur évoque les limites de sa recherche : 3% de FP restent un volume de requêtes assez volumineux pour être traité ; il conseille également d'ajouter de nouvelles features.[16]

Detection of HTTPS Malware Traffic

D'après František Střasák, le nombre de malware utilisant le protocole chiffré Hypertext Transfer Protocol Secure (HTTPS) croît ... [58] À l'aide de machine learning, l'idée du travail est de trouver des caractéristiques et des méthodes permettant de détecter les malwares sans déchiffrer le trafic. Pour réaliser ses tests, l'auteur utilise les datasets déja labélisés du projet *Stratosphere IPS* [51] détaillés dans la section 4.4.2. De plus, par manque de données, il crée et labélise également ses propres datasets. Les données de ces datasets sont ensuite agrégées en deux étapes : la première utilise l'outil *IDS Bro* (voir 4.3) qui permet d'extraire les données et de les combiner par requêtes et réponses (un flux) ; la deuxième étape agrège les données par 4-uplet (IP source, IP de destination, port de destination et protocole). František analyse le trafic HTTPS et définit 3 types de features pour caractériser les requêtes : les informations de la connexion, du protocole Secure Sockets Layer (SSL) et de certificats. Au total, 28 features sont calculées pour chaque 4-uplet. František utilise plusieurs algorithmes de machine learning dont : *Support Vector Machine (SVM)*, les réseaux de neurones, *random forest* et *XGBoost1.* L'auteur obtient les meilleurs résultats avec *XGBoost1* qui détecte les requêtes malicieuses avec

16. Jeu de données d'entraînement
17. Jeu de tests
18. Un domaine légitime est classé comme domaine malicieux par le système
19. Un domaine malicieux est classé comme domaine légitime par le système

une précision (5.3.3) de 96.64% mais un FP [20] de 1.89%. Il conclut également que de meilleurs résultats pourraient être possibles avec plus de datasets labélisés.[58]

En conclusion, l'auteur montre qu'il est possible de garder la privatisation des données tout en détectant le trafic malicieux. Il pointe également une référence de datasets disponibles et prêts à l'emploi. František met aussi à disposition son code source, ce qui permet d'avoir une première approche pertinente.

Botnet detection in encrypted traffic - a machine learning approach

Dans sa thèse [49], Laurent Miny se base sur les recherches de František Střasák, *Detection of HTTPS Malware Traffic* [58], et poursuit l'objectif d'améliorer la détection de botnets dans des jeux de données chiffrés. Pour cela, il reproduit dans un premier temps les tests du travail de František Střasák dans le but d'obtenir des résultats similaires. Une fois le but atteint, il ajoute dans un deuxième temps des features DNS et une feature supplémentaire du protocole TLS pouvant améliorer le score de détection de botnet. La liste des caractéristiques définies (par exemple DNS, voir figure 3.5) et à l'aide de plusieurs algorithmes de machine learning, il réalise plusieurs tests et les compare. L'auteur obtient d'excellents résultats grâce à la feature supplémentaire du protocol TLS. De fait, l'algorithme XGBoost fournit un taux de détection de 99,7% (5.3.3) avec un FP [21] de 0.1%. Il souligne enfin que l'ensemble de ces résultats doivent être mis en perspective avec les datasets (du projet *Stratosphere IPS* [51]) qui peuvent être biaisés.[49]

En conclusion, L. Miny a amélioré les résultats du travail de František Střasák [58] en ajoutant des features DNS. Dans ses recherches, il a également apporté une nouvelle feature ayant un impact positif sur ses résultats. Finalement, l'auteur souligne que les résultats peuvent être biaisés dûs aux datasets récupérés du projet *Stratosphere IPS*.[51]

IDs	Features
F41	in the alexa top100
F42	in the alexa top1k
F43	in the alexa top10k
F44	in the alexa top100k
F45	in the alexa top1m
F46	not in alexa
F47	FQDN length
F48	domain name length
F49	number of numerical chars
F50	number of non alphanumeric chars
F51	number of unique IP addresses in response
F52	number of subdomains
F53	average ttls
F54	std ttls
F55	min ttls
F56	max ttls
F57	number of hyphens in fqdn
F58	length of longest subdomain name
F59	number of voyels in FQDN
F60	number of different chars in FQDN
F61	number of consonants in FQDN
F62	shanon entropy 2ld
F63	shanon entropy 3ld

FIGURE 3.5 – Features du protocole DNS [49]

20. Un domaine légitime est classé comme domaine malicieux par le système
21. Un domaine légitime est classé comme domaine malicieux par le système

Synthèse

Nous synthétisons ce chapitre en soulignant l'évolution tout au long des recherches des techniques de détection de trafic malicieux à l'aide de l'intelligence artificielle.

À cet égard, un canevas des approches des différentes études peut être inféré de la manière suivante. Tout d'abord, il est nécessaire de se procurer une capture du trafic DNS du réseau à analyser. Sur base de cette capture, plusieurs analyses sont réalisées afin de construire une liste de caractéristiques permettant de différencier le trafic malicieux du trafic légitime. En parallèle, deux bases de connaissances sont construites. La première reprend un ensemble de noms de domaine connu comme malicieux (*blacklist*), tandis que la deuxième contient les noms de domaine légitimes (*whitelist*). Ensuite, à partir de chaque flux (requête/réponse DNS), ces caractéristiques sont calculées et le flux est labélisé malicieux ou légitime. Le tout donnant un dataset labélisé. Puis, à l'aide des algorithmes de machine learning, un modèle général est entraîné et testé. Les résultats sont alors analysés et, s'ils s'avèrent prometteurs, le modèle est utilisé pour classer de nouveaux flux.
De plus, au vu des contenus des différentes études citées ci-dessus nous déduisons les points suivants :

- *Notos* et *Exposure* reposent sur les caractéristiques d'adresses IP dans leur système de détection, ce qui apporte une limitation en terme d'évolution future. De fait, le déploiement d'IPv6 fournira une plage d'adresses IP plus vaste ce qui induit une gestion plus difficile des réputations de celles-ci ;
- les features sont créées en fonction des données disponibles et de l'environnement dans lequel a été collecté le jeu de données pDNS ;
- les algorithmes ont besoin de beaucoup de données pour être entraînés avec précision ;
- l'utilisation de sources externes de données déjà labélisées est non-négligeable ;
- la présence de deux types de trafics (malicieux, légitime) dans les jeux de données pDNS est obligatoire ;
- deux types de groupes de features se dessinent : réseau, lexical ;
- lors de la mise en place d'une solution proposée ci-dessus, les auteurs soulignent les impacts pour un botnet. Ces impacts forcent le *botmaster* à trouver de nouvelles techniques d'évasion ou à modifier les paramètres de leurs techniques actuelles. Ces modifications sont néfastes pour l'activité de leur botnet.

Ces études scientifiques sont sans conteste des sources d'inspiration de travail pour mener à bien des tests de cette nature.

Chapitre 4

Outils

Le chapitre 3 nous indique différentes approches possibles en terme de détection de trafic malicieux. Encore faut-il utiliser les outils adéquats pour que ces méthodes soient optimisées. Le présent chapitre vise à présenter et expliquer notre démarche dans la construction d'un outil de détection du trafic malicieux. L'approche empirique dont il est l'objet est décrite, tant dans son environnement que par rapport aux datasets utilisés, ainsi que par la démarche heuristique. Dans le cadre de ce mémoire, la plupart des outils cités ont fait partie d'une formation complète. Nous décrivons tout d'abord l'environnement dans lequel nous avons travaillé. Ensuite, l'outil que nous utilisons pour créer nos datasets. Enfin, nous comparons les différents types de datasets utilisés lors des tests. Finalement, nous évoquons le fonctionnement d'un service de vérification de noms de domaine en ligne.

4.1 Programmation

Dans un premier temps, nous nous sommes focalisés sur la compréhension de la structure des paquets DNS. À cet égard, nous utilisons *Wireshark*[1], un outil de capture et de visualisation de trafics réseaux (un exemple de réponse se trouve en annexe A.4). Dans un second temps, nous avons sélectionné un ensemble d'outils afin d'exploiter les datasets via des scripts. Ces scripts sont réalisés en langage *Python*[2] dans l'éditeur *anaconda*[3] prenant en charge l'affichage des graphes. Le choix s'est porté sur *Python* au vu de son apprentissage convivial, de la large panoplie de librairies disponibles, de sa gratuité ainsi que de sa communauté active. La partie machine learning est exploitée à l'aide de la librairie gratuite *scikit-learn*[4], qui fournit une collection d'algorithmes détaillée dans le chapitre 5. Elle apporte également un ensemble de facteurs permettant de mesurer les performances des systèmes. Finalement, en présence de big data, il est avantageux d'utiliser des graphes. L'affichage de ceux-ci est réalisé par la librairie *pandas*[5] qui encapsule en réalité des appels à la librairie *matplotlib*[6]. Il est à noter que le code est passé par plusieurs représentations et plusieurs phases d'ajustement tout au long du projet. Ainsi, pour garder un versioning cohérent, l'outil *github*[7] a largement été utilisé à chaque itération du travail.

1. https ://www.wireshark.org/
2. https ://www.python.org/
3. https ://www.anaconda.com/
4. https ://www.anaconda.com/
5. https ://pandas.pydata.org/
6. https ://matplotlib.org/
7. Github est une plateforme de développement mettant à disposition plusieurs outils dont le stockage de codes, le versioning, la gestion de projet, ... *https ://github.com/*

4.2 Serveur

Le nombre de données étant conséquent et les algorithmes demandeurs de ressources, une des premières difficulté est de trouver une machine capable de répondre à ces besoins. Il s'est avéré utile d'installer un serveur avec des ressources plus évoluées que celles que l'on retrouve dans un ordinateur portable ou un PC. C'est pourquoi un Linux de la distribution Ubuntu 16.04.2 LTS a été installé dans une machine virtuelle sur un environnement VMware [8]. La configuration de celle-ci est composée de 32 CPUs, 64 GB RAM et 400 GB d'espace de stockage. Une illustration de sa configuration se trouve en annexe A.5.

4.3 IDS Bro

Tournant sur ce serveur, IDS Bro est une plateforme pour l'analyse de trafic réseau passif développé et supporté par une communauté active depuis plus de 20 ans. Il met à disposition des fonctionnalités puissantes permettant de détecter les activités suspectes d'un réseau. De manière générale, sur base des captures réseau, IDS Bro génère des fichiers de type **.log* de haut niveau classés par protocole. L'avantage de ces fichiers est qu'ils sont créés dans un format texte sur base d'une structure facile à la réutilisation. Dans le cadre de ce mémoire, IDS Bro répond au besoin de transformation du fichier pcap en fichier texte. Cette solution a été choisie comme solution "standard" et point de départ dans notre pipeline (voir pipeline 6.2). Une illustration d'un fichier **.log*, à savoir *dns.log*, se trouve au listing 6.1. [4]

4.4 Datasets

Au cours du travail, plusieurs types de datasets ont été utilisés dont ceux générés par IDS Bro. Un dataset est avant tout une collection de données similaires ou ayant des relations intrinsèques. Ces données sont destinées à être préservées dans une structure de fichier indépendamment d'une application et ce dans le but de faciliter son exploitation au cours de futures recherches. [18] Nous évoquons dans cette partie les datasets non-labélisés, labélisés, et en ligne.

4.4.1 Datasets non-labélisés

C'est au sein du campus de l'UNamur qu'une capture réseau a été prélevée. Cette capture, fournie en format Packet Capture (pcap), contient uniquement le trafic DNS. Une des premières étapes est de convertir la capture en dataset afin de manipuler les données. Ce dernier est à la fois la base et l'objectif de notre travail. En effet, le dataset permet d'analyser le trafic circulant au sein d'une Université tout en essayant de différencier le trafic légitime du trafic malicieux. Et c'est là tout l'enjeu de ce travail ; nous tâchons progressivement de passer d'un dataset comportant des données non-labélisées en données labélisées.

4.4.2 Datasets labélisés

Dans le cadre de la première partie de ce travail, nous recherchons dans le dataset du campus, des éventuelles traces de trafic malicieux et légitime. Pour nous aider et effectuer cette distinction, deux types de listes sont utilisées : les listes noires (*blacklists*) et les listes blanches (*whitelists*). De toute évidence, nous privilégions des datasets gratuits.

8. La machine virtuelle est configurée sur un environnement ESXi https ://www.vmware.com/products/esxi-and-esx.html

Whitelist

Les listes blanches ou *whitelists* contiennent le classement des noms de domaine les plus populaires d'Internet. Ces listes se différencient par leur construction en utilisant différentes méthodes et métriques. Nous retrouvons les plus réputées comme Alexa [9], Cisco Umbrella [10], Majestic [11] et Quantcast [12]. Toutefois, nous soulignons qu'elles ne sont pas à l'abri d'inclure des noms de domaine malicieux (0,1% d'après [44]). Ce nombre étant relativement bas, elles restent tout de même une bonne référence dans le classement du trafic. Dans notre recherche, nous avons sélectionné la liste la plus connue Alexa. De fait, d'après plusieurs recherches, l'utilisation du top 1000 d'Alexa apporte une amélioration dans les résultats obtenus.[45] Cela est dû au fait que des noms de domaine malicieux deviennent populaires, mais pas assez pour entrer dans le top 1000.

Blacklist

De nombreux projets mettent à disposition des listes noires ou *blacklists* contenant les noms de domaine, adresses IPs, URLs réputés comme malicieux. Nous en déduisons que les trois types de listes proviennent des techniques des botnets vues à la section 2.2.5. À l'égard du nombre de listes disponibles sur internet, nous privilégions les listes décrites avec un taux de False Positives faible [13]. Lors de nos recherches, nous avons rencontré plusieurs difficultés :

- une diversité des structures des fichiers au travers de différents projets ;
- le manque d'informations quant à la fréquence des mises à jour ;
- la mixité de 2LD et 3LD, voir plus, dans les listes de noms de domaine ;
- ces fichiers sont de plus en plus utilisés et sont donc régulièrement téléchargés. Pour éviter d'engorger les serveurs, les zones de téléchargements nécessitent une authentification ;
- au cours du projet, la blacklist *https ://urlhaus.abuse.ch/api/#retrieve* a changé de format. Le téléchargement des nouvelles versions n'est plus adapté à notre code.

De manière évidente, nous privilégions les listes régulièrement mises à jour. Pour ce projet, nous avons sélectionné des listes de chaque type (noms de domaine, adresses IPs, URLs) représentées dans la figure 4.1

9. https ://www.alexa.com/tops
10. https ://umbrella.cisco.com/blog/2016/12/14/cisco-umbrella-1-million
11. http ://downloads.majestic.com/majestic_million.csv
12. https ://www.quantcast.com/top-sites/
13. Le pourcentage de noms de domaine légitimes présent dans la liste de noms de domaine malicieux est relativement bas

Site web	Description	Type	Mise à jour	URL	
abuse.ch	**IPs du Botnet Feodo**	**IPs**	**oui**	**https://feodotracker.abuse.ch/downloads/ipblocklist.txt**	**V**
	Domaines de Ransomwares	**Domaines**	**oui**	**https://ransomwaretracker.abuse.ch/downloads/RW_DOMBL.txt**	**V**
	URLs de Ransomwares	**Mixte**	**oui**	**https://ransomwaretracker.abuse.ch/downloads/RW_URLBL.txt**	**V**
	IPs de Ransomwares	**IPs**	**oui**	**https://ransomwaretracker.abuse.ch/downloads/RW_IPBL.txt**	**V**
	IPs connexions SSL de C&C	**IPs**	**oui**	**https://sslbl.abuse.ch/blacklist/sslipblacklist.txt**	**V**
	URLs de malwares	**Mixte**	**oui**	**https://urlhaus.abuse.ch/downloads/csv/**	**V**
	Liste de domaines du botnet Zeus	Domaines	nc	https://zeustracker.abuse.ch/blocklist.php?download=baddomains	
	IPs du botnet Zeus	IPs	nc	https://zeustracker.abuse.ch/blocklist.php?download=badIPs	
osint.bambenekconsulting.com	**C&C domaines**	**Domaines**	**oui**	**http://osint.bambenekconsulting.com/feeds/c2-dommasterlist.txt**	**V**
	Liste de domaines DGA	**Domaines**	**oui**	**http://osint.bambenekconsulting.com/feeds/dga-feed-high.csv**	**V**
malwaredomainlist.com	Liste d'hôtes	Domaines	nc	http://www.malwaredomainlist.com/hostslist/hosts.txt	
	IPs de traffics malicieux	IPs	nc	https://www.malwaredomainlist.com/hostslist/ip.txt	
	Liste de domaines	Domaines	nc	https://www.malwaredomainlist.com/hostslist/delisted.txt	
malwaredomains.com	**Liste de domaines suspects**	**Domaines**	**oui**	**http://malc0de.com/bl/ZONES**	**V**
	Liste de domaines suspects	Domaines	nc	http://mirror1.malwaredomains.com/files/justdomains	
dshield.org	**Liste de domaines suspects**	**Domaines**	**oui**	**https://www.dshield.org/feeds/suspiciousdomains_High.txt**	**V**
	Top 20 des classes C de sources d'attaques	IPs class c	oui	https://www.dshield.org/block.txt	
phishtank.com	URLs de phishings	URLs	oui	https://data.phishtank.com/data/<id-key>/online-valid.csv	
talosintelligence.com	**IPs de traffics malicieux**	**IPs**	**oui**	**https://talosintelligence.com/documents/ip-blacklist**	**V**
malc0de.com/bl/	**Liste de domaines suspects**	**Domaines**	**oui**	**http://malc0de.com/bl/BOOT**	**V**
	IPs de traffics malicieux	**IPs**	**oui**	**http://malc0de.com/bl/IP_Blacklist.txt**	**V**

FIGURE 4.1 – Liste de blacklists

Datasets de recherche

Lors de la deuxième partie de ce travail, une recherche à l'aide de machine learning tente de classer les données qui n'ont pas été labélisées à la fin de la phase précédente. Toutefois, si la quantité de donnée labélisées dans la phase précédente n'est pas suffisante, le machine learning a besoin de données supplémentaires pour entraîner un modèle, ensuite le tester et enfin l'utiliser pour classer des données. Sur Internet on peut trouver toutes sortes de jeux de données, mais celles-ci manquent souvent de documentations et de labels. Or, pour entraîner notre modèle, les données doivent être labélisées. Lors de nos recherches, nous nous sommes arrêtés au projet nommé *Malware Capture Facility Project* ou MCFP.[6]

Le but de ce projet est de capturer, analyser et publier des captures de trafics réels et de longues durées dans un environnement contrôlé. L'avantage des datasets fournis est qu'ils sont complètement documentés, ce qui permet de les labéliser à l'aide de scripts.
Dans la liste des datasets fournis par MCFP, nous nous focalisons sur le trafic Botnet et le trafic normal. Le premier est constitué de 390 répertoires, nommés "CTU-Malware-Capture-Botnet-*" et le deuxième reprend 25 répertoires, nommés "CTU-Normal-*". Dans chaque répertoire, on retrouve plusieurs fichiers. Pour nos tests, concernant le protocole DNS et pour chaque répertoires, nous nous intéressons à deux fichiers. Le premier est le fichier *dns.log* du sous-répertoire */Bro* ; ce dernier contient les données des requêtes DNS. Le second est le fichier *readme*, pourvu des informations nécessaires pour labéliser le ficher *dns.log*.

4.4.3 Gestion de datasets en ligne

À travers nos recherches sur les blacklists et whitelists (voir 4.4.2), nous avons découvert plusieurs possibilités de gestion de datasets. En dehors de l'importance de la mise à jour régulière et de la gratuité de ceux-ci, il existe deux types de gestion des datasets :

— le téléchargement d'un fichier texte : celui-ci est stocké sur un serveur distant. Une fois le fichier manuellement ou automatiquement téléchargé, celui-ci est chargé dans un service/programme que l'on peut alors exploiter. L'inconvénient majeur est qu'il est lié à un téléchargement régulier pour le maintenir à jour localement. Un autre inconvénient est que si le format du fichier venait à changer, nos scripts devraient être réécrits ;

— le *DNS service lookup* : ce service permet de vérifier la réputation d'une ressource au travers de datasets distants. Le principe est simple, à partir d'un nom de domaine (ex : *pagaldaily.com*) que l'on veut vérifier, on soumet une résolution de domaine en ajoutant le suffixe du nom de domaine du dataset distant (ex : *.multi.surbl.org*). La résolution

renverra une adresse IP codée. Dans le cas de la figure 4.2, l'adresse IP renvoie est 127.0.0.8 signifiant l'implication du nom de domaine dans du Phishing.[59] D'autres listes sont disponibles comme ZeusTracker [8], SORBS [57],...

```
rebaudin ~ $ dig +short pagaldaily.com.multi.surbl.org TXT pagaldaily.com.multi.surbl.org A
"Blocked, pagaldaily.com on lists [ph], See: http://www.surbl.org/lists"
127.0.0.8
```

FIGURE 4.2 – Résultat d'une résolution pour le nom de domaine malicieux *pagaldaily.com*

Comme première approche, ce travail utilise la première technique explicitée. En effet, celle-ci nous permet d'accélérer nos phases de tests et d'exclure toutes dépendances vers tout service externe.

4.5 Synthèse

Ce chapitre enveloppe toutes les informations nécessaires concernant l'environnement dans lequel nous travaillons. Nous commençons par y décrire le langage utilisé. De fait, l'apprentissage du langage *python* est une étape indispensable, cela dans le but d'utiliser les librairies *scikit-learn* et *pandas*. Ensuite, nous y évoquons le logiciel *IDS Bro* qui nous permet d'obtenir des fichiers standards, respectant une structure stricte et maniable. Enfin, nous listons la sélection de six *blacklists* et d'une *whitelist*, servant de référence dans nos recherches de trafics malicieux. Finalement, le projet *MCFP* fournit des datasets documentés ; une aubaine pour entraîner nos algorithmes de machine learning décrits dans le chapitre suivant (chapitre 5).

Chapitre 5

Le machine learning

Dans ce chapitre, nous allons aborder les concepts essentiels du machine learning utilisés dans nos expérimentations. Nous commençons par la distinction entre l'apprentissage supervisé et non-supervisé. Ensuite nous évoquons les algorithmes utilisés au sein des expérimentations ainsi que leurs caractéristiques. Puis, nous parcourons les éléments permettant d'évaluer ces algorithmes. Enfin, les outils d'amélioration des algorithmes sont abordés.

5.1 Supervisé et non supervisé

Supervisé

Le but de l'apprentissage supervisé est défini en deux étapes. La première est d'apprendre à un modèle [1] à prédire une réponse (ou *label*) à partir d'un échantillon de données entrées. La deuxième étape consiste à prédire la réponse pour toute nouvelle donnée inconnue en entrée. Le désavantage du supervisé est qu'il est limité à des données labélisées. Ce nombre de données devra être important, au quel cas la variation dans l'espace des données sera pauvre. Les données labélisées sont parfois difficiles à obtenir au vu des compétences que cette tâche peut requérir. De fait, dans le cas de ce mémoire, il faut trouver des personnes expertes dans la sécurité, plus précisément dans le domaine des malwares et du protocole DNS, pour labéliser des données et ainsi construire un dataset labélisé.
Les deux tâches les plus recherchées dans l'apprentissage supervisé sont : la classification et la régression. La première vise principalement à prédire une classe, une catégorie ou un label parmi un ensemble prédéfini. La deuxième a pour but de prédire une valeur continue non définie.[27]

Non Supervisé

Dans l'apprentissage non supervisé, un échantillon de données sans *label* est utilisé afin d'apprendre à un modèle à créer des groupes homogènes. [27] L'idée du non supervisé est de visualiser des données pour éventuellement observer des segments partageant les mêmes propriétés. Les datasets utilisés pour le non supervisé demandent moins de travail car ils n'ont pas besoin d'être labélisés. Dans le cadre de ce travail, nous utilisons l'apprentissage supervisé.

1. un modèle est une représentation généralisée d'un ensemble de données nommé échantillonnage

5.2 Algorithmes supervisés de classification

Dans le cadre de ce projet, nous avons sélectionné un ensemble d'algorithmes sur base des bons résultats fournis dans les études existantes.

Régression logistique

Logistic regression ou régression logistique, est un modèle simple et utilisé principalement pour la classification binaire (deux classes ; par exemple : "botnet" et "normal"). Il a l'avantage de ne pas être gourmand en terme de puissance de calculs et ne nécessite pas de réglages. Son inconvénient est qu'il n'est pas capable de prendre en charge un grand nombre de features. De plus, le modèle est connu pour être vulnérable à l'overfitting [2].[19]

K-Nearest Neighbors - KNN

Le *K-Nearest Neighbors* ou les "K plus proches voisins", est un algorithme de classification simple mais puissant et d'une grande utilité. Dans la réalité, il est largement utilisé au travers d'une variété d'applications. Son principe est simple : dans un espace, les données d'apprentissage sont tout d'abord placées. Ensuite, pour toute donnée de tests celle-ci sera placée dans l'espace et associée à la classe des k plus proches voisins. La valeur de k sera, la plupart du temps, un nombre impair pour éviter l'égalité lors de l'évaluation. Un des inconvénients du KNN est le fait de stocker les informations du training set en brut. En d'autres termes, il n'y a pas de travail de généralisation. Par conséquent, les prédictions deviennent très coûteuses à partir du moment où l'on travaille avec des datasets plus volumineux.[27]

Arbres de décision

Decision Tree ou arbres de décision est un simple modèle non linéaire de classification. Dans la pratique, ce modèle est plus rarement utilisé, mais il permet de mieux comprendre les autres modèles. De fait, les forêts de décision sont des sous-composants de modèles plus puissants. Il est représenté par une structure hiérarchique dans laquelle un noeud représente des règles de décisions inférées de données lors de l'apprentissage. Le parcours des noeuds suivant ces règles mène à une classification. Les arbres de décision ont l'avantage d'être simples à comprendre et à interpréter. Il demande une préparation des données, comme la normalisation de données ou encore la suppression des valeurs nulles [3]. Par contre, ils ont l'inconvénient de créer des arbres parfois trop complexes reflétant parfaitement le modèle de chaque donnée fournie lors de l'apprentissage. Cela a comme conséquence de diminuer la généralisation de données, ce que l'on appelle l'*overfitting*. De plus, les arbres de décision peuvent également être biaisés dans le cas où des classes en dominent d'autres. Pour éviter de telles situations, il est préférable d'équilibrer le dataset 5.4.3 avant l'apprentissage par l'algorithme.[27]

Forêts aléatoires

Random Forest ou Forêts aléatoires est un ensemble d'arbres de décision créés à partir de sous-ensembles des données d'apprentissage et des features. Ce modèle est souvent utilisé et comprend un paramètre principal étant le nombre d'arbres dans la forêt. Plus le nombre d'arbres augmente plus le modèle est amélioré, cependant cela a un coût sur la complexité de calcul et peut être plus lent lors de l'estimation d'une classe.[27]

2. L'*overfitting* désigne le sur-apprentissage d'un algorithme. On parle de sur-apprentissage lorsque le modèle apprend le bruit des données fournies.

3. une valeur nulle est dans ce cas une donnée absente

AdaBoost

AdaBoost est un algorithme de la famille *Boosting* faisant elle même partie des méthodes semblables aux Forêts aléatoires. Comme les Forêts aléatoires, *AdaBoost* crée un ensemble d'estimateurs pour réduire le risque d'un unique estimateur biaisé. [27] Il est facile à implémenter et n'a pas tendance à être en sur-apprentissage. L'inconvénient est qu'il est sensible aux bruits [4] dans les données. Par conséquent, les décisions seraient incorrectement fondées.

5.3 Évaluation

Il existe une variété de métriques pour évaluer et comparer les performances des modèles de classification. Ces métriques sont basées sur les prédictions correctes et incorrectes des labels par un modèle. Mais commençons d'abord par évoquer une bonne pratique d'apprentissage et de test.

5.3.1 Apprentissage et test

Dans l'apprentissage supervisé, entraîner un modèle puis le tester sur les mêmes données est une méthodologie incorrecte [5]. En effet, les prédictions sur ces données seraient parfaites alors que toutes nouvelles prédictions de données inconnues seraient incorrectes. On parle dans ce cas de sur-apprentissage ou *overfitting*. C'est pourquoi, une des méthodes recommandées est de scinder les données en deux ; d'une part les données destinées uniquement à l'apprentissage (*training set*) et d'autre part les données afin de tester le modèle (*testing set*). Il est de bonne pratique de diviser le dataset en 80%/20%, respectivement *training set*/*testing set*.

5.3.2 Matrice de confusion

La matrice de confusion, pour une classification binaire, est une matrice de dimension 2 * 2. Elle est utilisée afin d'évaluer les performances d'un modèle de classification. L'évaluation repose principalement sur les quatre concepts suivant : *True Positives ou TP*, *True Negatives ou TN*, *False Positives ou FP*, *False Negatives ou FN*. Les classes sont représentées par le positif et le négatif tandis que vraie ou faux signifie qu'une prédiction est respectivement correcte ou incorrecte. Tout à l'image de son appellation, la matrice de confusion peut parfois être difficile à interpréter. En effet, il revient à l'auteur de définir ce qui est positif, négatif, mais également de nommer les axes comme classe "réelle" et "prédiction". Ainsi, au début de nos recherches, la compréhension de celle-ci fut laborieuse. Dans la matrice de confusion ci-dessous, les lignes correspondent à la classe réelle, alors que les colonnes représentent les classes prédites par le classificateur.

4. lorsque les données contiennent des enregistrements erronés ou des aberrations
5. https ://scikit-learn.org/stable/modules/cross_validation.html#cross-validation

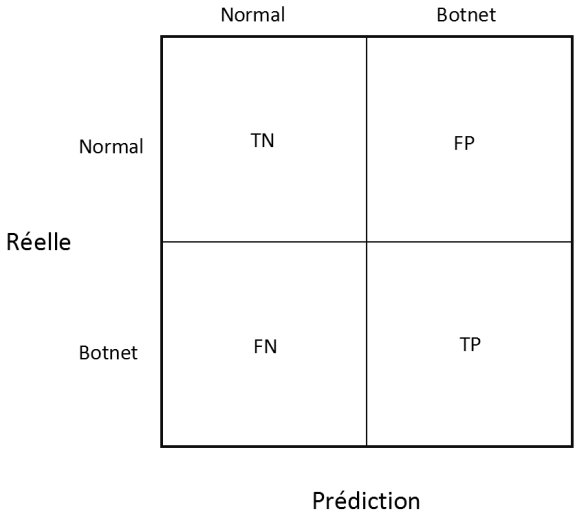

FIGURE 5.1 – Matrice de confusion utilisée dans les expériences de ce projet [27]

5.3.3 Métriques

Lors de nos expériences, nous mettons en concurrence les différents classificateurs cités ci-dessus de manière à mettre en évidence le plus performant. Pour les comparer, nous avons vu que la matrice de confusion fournit déjà une bonne indication sur le pourcentage de classes prédites correctement et incorrectement. Ces indications sont disponibles sous la forme de quatre variables : TP, TN, FP et le FN. Ces quatre variables sont la base d'une variété de métriques permettant d'évaluer les classificateurs.[27]
Ceux-ci sont présentés ci-dessous :

Accuracy

L'*accuracy* ou l'exactitude est une mesure indicative mais pas assez précise. De fait, celle-ci peut fournir de bons résultats mais peut également cacher un modèle ayant été biaisé en raison d'un échantillon de données non équilibré (voir 5.4.3). En conséquence, elle est utilisée avec d'autres métriques. Dans notre cas, celles-ci sont citées dans cette section.

$$Accurary = \frac{TN + TP}{TN + FP + FN + TP} \tag{5.1}$$

Precision

C'est la proportion de données prédites positives et qui sont réellement positives. Plus la précision est grande, moins il y a de false positives. En d'autres termes, plus la précision est grande, plus le modèle détecte de réelles requêtes de botnet.

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

Recall

Recall ou *True Positive Rate* c'est la capacité pour un classificateur de prédire toutes les données positives. C'est la proportion d'instance de données positives prédites comme positives. Plus le *recall* est grand moins il y a de False Negative. En d'autres termes, plus le *recall* est

grand, moins le risque de détecter des requêtes malicieuses comme légitimes est présent.

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

F1 score

Représente une mesure globale combinant la *precision* et le *recall*. Plus il est grand, plus la *precision* et le *recall* sont performants.

$$F1score = \frac{2 * precision * TPR}{precision + TPR} \tag{5.4}$$

Area Under the ROC curve - AUC

Area Under the ROC curve ou AUC est une métrique insensible aux datasets non-équilibrés. Elle représente les performances d'un classificateur en tenant compte des deux classes.[27] L'AUC est une valeur essentielle dans la classification [6].

5.4 Améliorations

5.4.1 Cross Validation

Dans la section *Apprentissage et test* 5.3.1, nous expliquons qu'il est indispensable de séparer le dataset labélisé fourni en deux parties pour éviter le sur-apprentissage. Cependant, dû aux hyperparamètres [7], le risque pour un classificateur d'être en sur-apprentissage existe toujours. Pour pallier ce problème, une possibilité est de réserver une partie des données du training set pour "une validation" avant la phase de test. Dès lors, après avoir entraîné le classificateur, il est validé sur ces données de validation et utilisé sur les données de test si les résultats sont concluants. L'inconvénient majeur de cette solution est la réduction de la quantité de données pour la phase d'apprentissage. C'est pour remédier à ce nouveau problème que le *cross validation* est utilisé (voir figure 5.2). D'abord, on définit le nombre d'ensembles k (*ici Fold*) que l'on veut créer (k prend souvent la valeur de 10 dans les études existantes), le modèle est alors entraîné sur k-1 données et validé sur la dernière partie. L'entraînement et sa validation sont alors répétés pour tout k-1 données du training set. La performance du classificateur est représentée par la moyenne des résultats des validations. Le *cross validation* est utilisé pour tester les hyperparamètres tout en évitant le sur-apprentissage et le sous-apprentissage.[55]

6. https ://developers.google.com/machine-learning/crash-course/classification/roc-and-auc ?hl=fr

7. Est un paramètre contrôlant l'apprentissage d'un algorithme. Il est commun d'effectuer plusieurs tests variant les valeurs des paramètres pour en trouver la meilleure valeur [27]

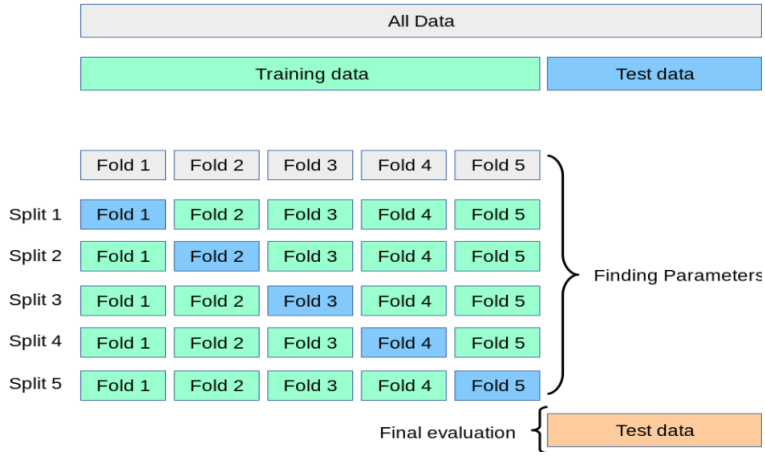

FIGURE 5.2 – Mécanisme du cross validation [55]

5.4.2 Grid search

Lors de la construction d'un modèle, les hyperparamètres par défaut sont souvent un bon début. Cependant, pour atteindre de meilleurs résultats et ainsi améliorer la performance du modèle, il faut tester un par un les hyperparamètres et évaluer les résultats. La méthode *Grid search* évite d'entrer manuellement les hyperparamètres. Pour chaque élément d'un ensemble de valeurs configurées, elle évalue le modèle. Son inconvénient est sa demande en terme de ressources de calculs lors de ses évaluations.

5.4.3 Équilibrage de données

Dans le contexte des algorithmes de classifications binaires (deux classes, "botnet" et "normal"), une intention particulière doit être apportée quant au ratio du nombre d'échantillons des deux classes définies. Si l'une des classes est dominante sur l'autre, cela peut mener à des métriques biaisées. C'est le cas lorsqu'un échantillon de données contient 90% de données d'une classe contre 10% de l'autre. Dès lors, si le classificateur classifie toutes les données en tant que première classe, alors l'*accuracy* sera de 90% ! Pour remédier à ce problème, des solutions existent [8]. Dans notre cas nous sélectionnons, pour les deux types de classes, la même quantité de requêtes.

8. https ://elitedatascience.com/imbalanced-classes

Chapitre 6

Expérimentations

Après les aspects méthodologiques et théoriques, il est pertinent d'aborder le volet des expérimentations. Cette partie essentielle au présent document fait l'objet de différents points que nous abordons de manière successive. Dans un premier temps, nous établissons les objectifs poursuivis et nous décrivons la méthode utilisée dans le cadre des traitements de données, à savoir le pipeline. Ce dernier n'a de sens que par rapport à son champ d'investigation que nous traitons dans les informations du dataset (la capture de l'université). La partie descriptive du travail d'analyse étant précisée, nous établissons la partie dynamique de l'analyse des données. Nous la divisons en deux ; d'une part, la labélisation à l'aide des listes de références comme présentée au chapitre 4 et d'autre part, la classification à l'aide de méthodes de machine learning vues au chapitre 5. Les résultats labélisés malicieux de la première partie des expérimentations sont vérifiés tandis que les résultats non-labélisés sont classés dans la deuxième partie. Enfin, les résultats sont listés et nuancés.

6.1 Objectif

Tenant compte des quantités de données à traiter et de l'objectif poursuivi, le big data est un environnement qui demande du temps. La mise en perspective des études existantes, ainsi que leurs données, ont permis de dégager des pistes de réflexions intéressantes pour la construction du pipeline (voir 6.2). Nous définissons dès lors l'objectif final comme étant la génération de deux fichiers. Le premier fichier *labelled_ data.csv* est le résultat de la première partie, la labélisation de requêtes malicieuses. Le deuxième fichier *classified.csv*, contient les résultats des requêtes classifiées comme malicieuses, le but étant de fournir une liste de requêtes malicieuses accompagnée de son taux d'erreur. La construction de notre pipeline est formée de deux parties. Cela a pour conséquence de détecter les requêtes malicieuses suivant deux considérations :

1. Dans la première partie, une requête est considérée comme malicieuse si une des deux assertions est vraie :
   — le FQDN inclus dans une requête DNS est connu dans une blacklist
   — l'adresse IP retournée du serveur de noms est connue dans une blacklist
   Un fichier *labelled_ data.csv* est généré.
2. Dans la deuxième partie, une requête est considérée comme malicieuse si le modèle construit sur base des caractéristiques d'un ensemble de données malicieuses et légitimes, juge la requête malicieuse.
   Un fichier *classified.csv* est généré.

6.2 Pipeline

L'approche inductive des multiples lectures (section 3) réalisées dans le cadre de ce travail ainsi que la mise en perspective des objectifs poursuivis nous permet de mettre en place un outil adapté qu'est le pipeline.

La "Partie 1" du pipeline de la figure 6.1 est liée au premier type de requête malicieux détaillé dans nos objectifs. C'est à l'aide des blacklists et des whitelists, décrites à la section 4.4.2, que l'on construit un mécanisme permettant d'analyser et ainsi labéliser toutes requêtes malicieuses dans la capture de l'UNamur.

La "Partie 2" du pipeline est fondée sur du machine learning dans lequel nous utilisons des données labélisées permettant de créer un modèle. Ce modèle est vu comme la généralisation d'une part, de requêtes malicieuses et d'autre part, de requêtes légitimes. Le modèle est d'abord testé afin de déterminer si sa méthode de classification des données introduites, malicieuses ou légitimes, est correcte. Ensuite, nous utilisons le modèle en vue de classer toute requête du deuxième type malicieux n'ayant pas été labélisée dans la partie 1.

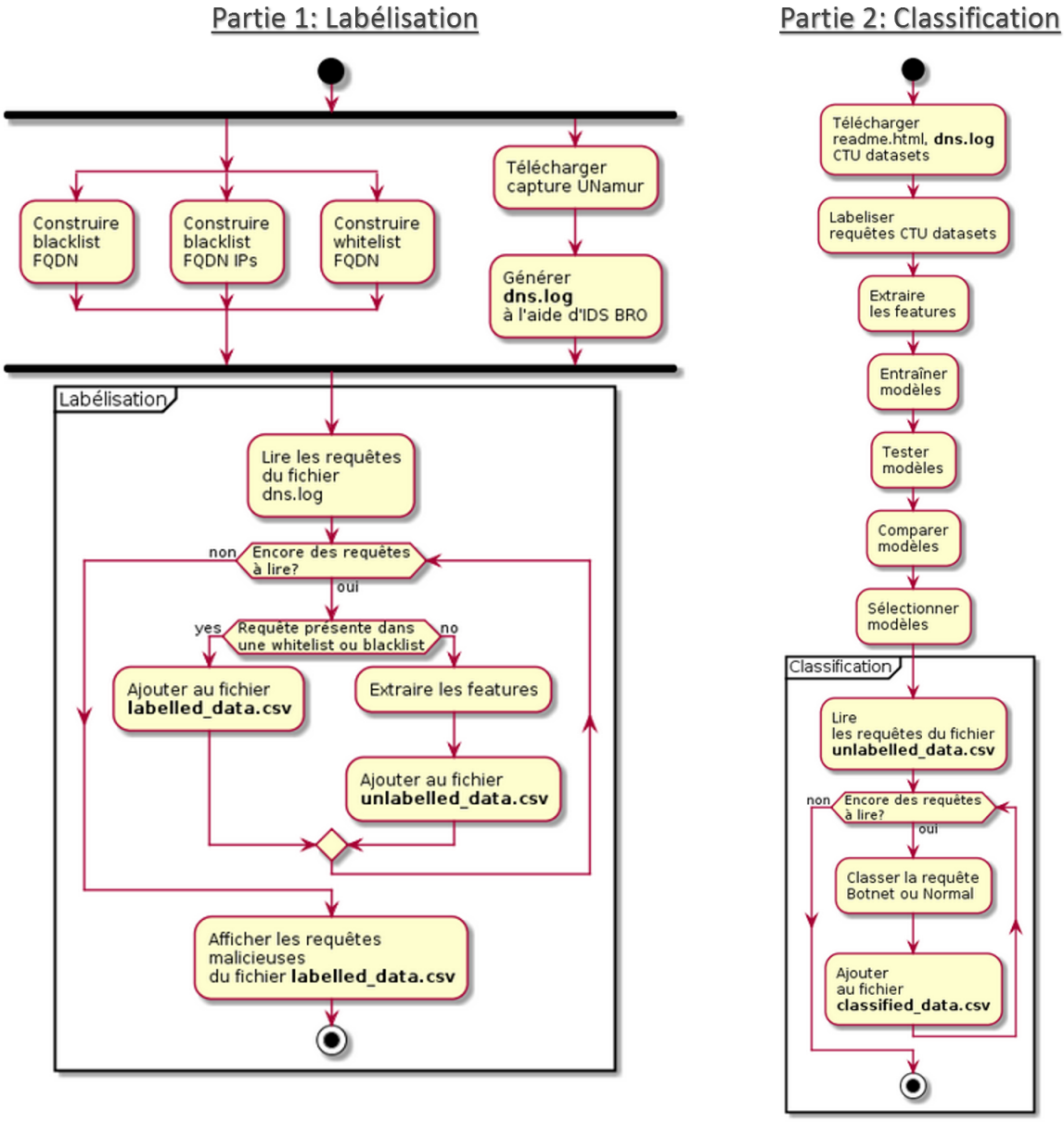

FIGURE 6.1 – Pipeline général

6.3 Informations du dataset de l'UNamur

Le dataset fourni dans le cadre des expériences a été réalisé et fourni par l'UNamur. Sa taille est de 8GB et nous le nommons *1384844*. Dans un premier temps, il nous a semblé judicieux d'analyser les données contenues dans celui-ci à l'aide d'outils (énoncés dans la section 4.1). Les résultats que nous avons pu analyser semblent pertinents et se rapprochent de ceux cités dans les études décrites au chapitre 3. Il se compose de 29856467 requêtes DNS dont les représentations sont mieux illustrées dans les graphiques de la figure 6.2. Nous décrivons chacun de ces graphes dans les points ci-dessous :

Type de requête

Nous constatons que les types de requêtes les plus utilisées sont, par ordre décroissant, le type "A", représentant une requête IPv4, le protocole le plus utilisé sur Internet de nos jours ; ensuite, ce sont les requêtes IPv6 (type AAAA). Celui-ci devrait, dans les années à venir, remplacer le protocole IPv4 qui est limité en terme de nombre d'adresses IP.

Protocole de communication

Concernant le protocole de communication, le plus utilisé est l'UDP. Cela semble logique étant donné l'utilisation unique de ce protocole pour la résolution de noms de domaine. De fait, le protocole TCP est quant à lui utilisé pour la mise à jour de zones ayant lieu moins régulièrement.

Code de réponse

Le troisième graphe du dessus montre le code des réponses le plus fréquent : "NOERROR". En d'autres termes, ce code précise qu'il n'y a pas d'erreur dans la résolution de domaine. Par contre, le code "NXDOMAIN" apparaît dans une mesure moindre et spécifie l'inexistence d'un nom de domaine. Cela peut être dû à une erreur de frappe dans un nom de domaine entré par l'utilisateur.

Fluctuation du trafic

Afin de mieux visualiser les résultats de la fluctuation du trafic obtenu, il paraît intéressant de les situer sur une ligne du temps. Sur un délai de 24 heures, la capture montre des pics apparaissant durant la journée et des creux en milieu de nuit.

Top 20 des sites les plus populaires

L'avant-dernier graphe fournit une vue du top 20 des requêtes DNS les plus populaires du campus. Le serveur LDAP [1] de l'UNamur occupe la première marche du podium, suivi des plus connus tels que *unamur.be*, *www.google.com* ou encore *www.facebook.com*. On peut également apercevoir des *reverse DNS* vu à la section 1.3.4. De manière générale, il apparaît clairement une diversité syntaxique dans les noms de domaine résolus.

1. LDAP est un protocole d'accès aux données, permettant d'interroger un annuaire. Un annuaire peut être présenté comme une base de données contenant toutes les informations d'utilisateurs ou systèmes du réseau de l'UNamur

Serveurs DNS

Finalement, le dernier graphe fournit le top 10 des serveurs DNS les plus utilisés. À ce sujet, le serveur lié à l'adresse IP *167.247.198.63* est probablement le serveur DNS de l'UNamur. D'un premier regard, il paraît étonnant de retrouver d'autres serveurs DNS. Cependant, comme nous l'avons vu, le protocole DNS est largement utilisé par d'autres services. Dans le dataset, une analyse plus détaillée relève des requêtes de type *1.98.246.173.sbl.spamhaus.org* vers les serveurs *107.189.79.82, 167.95.113.239 et 237.230.242.78*, ce qui nous rappelle le service décrit au point 4.4.3 permettant de vérifier à distance la légitimité d'un nom de domaine ou d'une adresse IP.

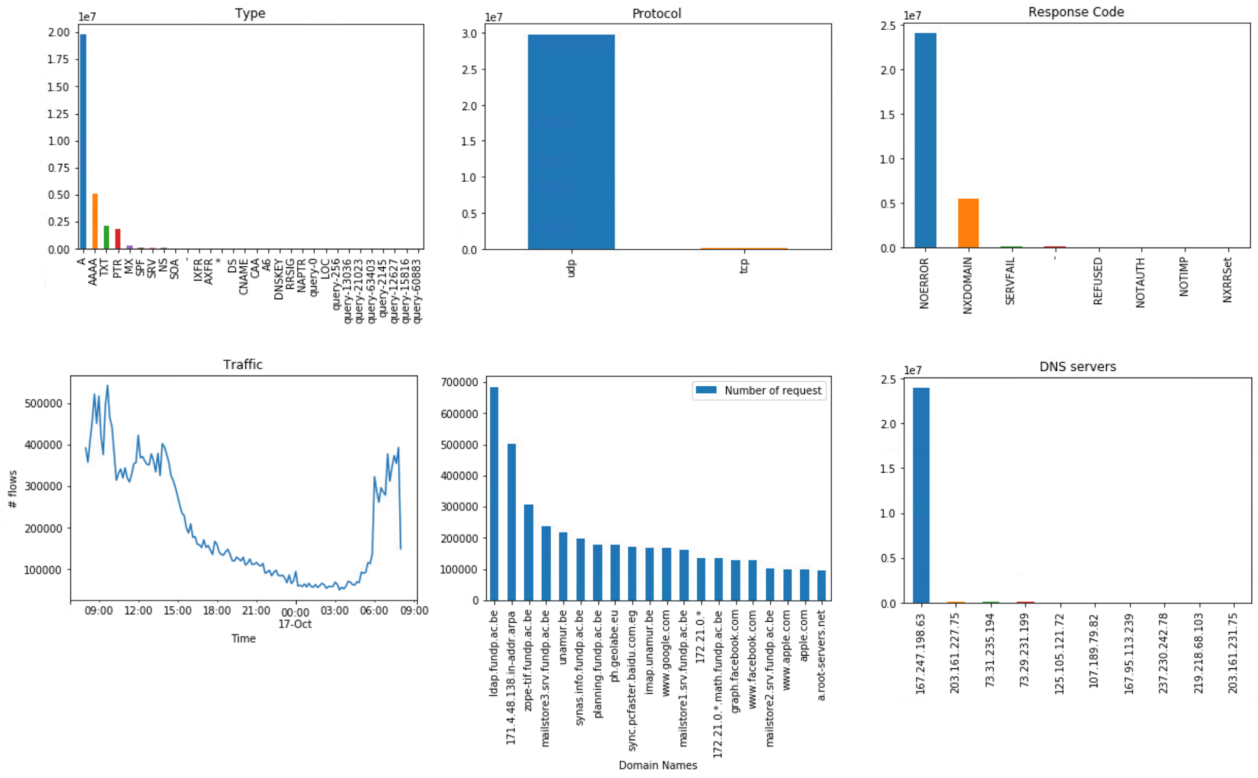

FIGURE 6.2 – Aperçu du dataset *1384844* de 8Gb

En synthèse

Le profil du dataset est composé de requêtes DNS de type *A* et *AAAA* reposant sur le protocole *UDP*. Les requêtes sont majoritairement résolues, ce qui peut être vérifié grâce au code de réponse *NOERROR*. Le pic des requêtes est localisé durant la matinée et l'ensemble des requêtes sont à destination du serveur DNS de l'UNamur *167.247.198.63*. Finalement, un top 20 des noms de domaine les plus populaires de l'UNamur présente le serveur *LDAP* comme le plus sollicité suivi des plus connus tels que *unamur.be*, *www.google.com* ou encore *www.facebook.com*. L'ensemble des requêtes définit le profil type de trafic d'un réseau universitaire.

6.4 Partie 1 : Labélisation à l'aide de listes de références

Étant donné la liste des noms de domaine contenue dans la capture de l'UNamur, cette première partie est consacrée à la labélisation de noms de domaine légitime ou malicieux au moyen de blacklists et whitelists (3.4). Deux expériences sont présentées ; la première a pour objectif de labéliser un maximum de requêtes DNS tandis que la deuxième a pour but de réduire le taux de False Positives et de False Negatives.

6.4.1 Labélisation d'un maximum de requêtes

Conversion de la capture

Le format de fichier fourni au départ de ce projet est de type pcap. Bien que celui-ci soit complet, il ne s'avère pas exploitable (voir 4.4) au travers de nos scripts. Pour y remédier, IDS BRO (section 4.3) est l'outil utilisé afin d'extraire les données vers un fichier *dns.log* structuré. Ce dernier contient les mêmes informations, mais agrège la paire de paquets requêtes/réponses par flux. Le fichier généré *dns.log* a une taille de 5.8Gb et contient 29856467 lignes représentant chacune une requête/réponse agrégée (un flux). La figure 6.1 montre en première ligne le nom des champs, puis le type des champs et enfin l'exemple d'un flux DNS. [63]

```
#fields ts uid id.orig_h id.orig_p id.resp_h id.resp_p proto trans_id rtt query qclass qclass_name qtype
    qtype_name rcode rcode_name AA TC RD RA Z answers TTLs rejected
#types time string addr port addr port enum count interval string count string count string count
    1547660857.305628 Ce4xfy3cHPlfvS2igj 10.0.2.15 40513 173.38.200.100 53 udp 63530 - www.unamur.bbc.com
     1 C_INTERNET 1 A 3 NXDOMAIN F F T F 0 - - F
1547660882.650835 C39rHx11k3d5N7IsTd 10.0.2.15 41635 173.38.200.100 53 udp 31786 0.044577 www.unamur.be 1
    C_INTERNET 1 A 0 NOERROR F F T T 0 138.48.4.201 5.000000 F
```

Listing 6.1 – Exemple du fichier généré IDS Bro dns.log (voir annexe A.1)

Construction de la blacklist

L'objectif de cette première partie est, sur base de listes de références blacklists (décrites dans la section 4.4.2), d'analyser les noms de domaine et les adresses IP de la capture *1384844* afin de labéliser les requêtes effectuées vers des sites connus comme malicieux. Les blacklists contiennent différents formats de données tels que des URLs, noms de domaine ou encore adresses IP. Pour remédier à cette divergence, nous décidons de créer deux collections. L'une est l'ensemble des adresses IP récoltées, l'autre est l'ensemble URLs et noms de domaine généralisés au format *SLD.TLD* (voir étude Pedro Marques da Luz 3.4). Au final, la première compte un nombre total de 6066 adresses IP et la seconde reprend 406286 noms de domaine connus comme malicieux (figure 6.3). L'approche est déduite de l'étude d'*Exposure* (voir étude 3.4) ; cependant nous n'implémentons pas le système de réputation d'une URL. De fait, notre échantillon contient 24h de trafic alors qu'*Exposure* possède des données DNS de deux mois et demi. En conséquence, nous ne possédons pas assez d'informations pour créer un système de réputation.

Il est à noter que, tout au long du développement des outils, une intention particulière s'est portée sur la performance des scripts. Un exemple dans la figure 6.3 montre un temps mesuré de 9,6 secondes pour charger et créer les blacklists.

```
Blacklist: RW_DOMBL.txt                entries: 1902
Blacklist: RW_IPBL.txt                 entries: 337
Blacklist: ipblocklist.txt             entries: 358
Blacklist: sslipblacklist.txt          entries: 116
Blacklist: ip-blacklist                entries: 1620
Blacklist: IP_Blacklist.txt            entries: 76
Blacklist: BOOT                        entries: 49
Blacklist: c2-dommasterlist.txt        entries: 727
Blacklist: dga-feed-high.csv           entries: 390793
Blacklist: suspiciousdomains_High.txt  entries: 19
Blacklist: ZONES                       entries: 50
Blacklist (IPs): hunAzL43.txt          entries: 19065
Blacklist (DNs): hunAzL43.txt          entries: 89118
Blacklist (IPs): RW_URLBL.txt          entries: 73
Blacklist (DNs): RW_URLBL.txt          entries: 11493
Total Blacklisted unique IPs:          entries: 6066
Total Blacklisted unique domain names: entries: 406286
Time to (After BL loaded) 0:00:09.641066
Whitelist: Alexa top1m                 entries: 951179
Whitelist: Cisco top1m                 entries: 234939
Total Whitelist unique domain names:   entries: 1061933
```

FIGURE 6.3 – Liste des blacklists

Construction de la whitelist

Le deuxième type de liste, la whitelist, contient uniquement des FQDN [2]. Elle est l'union du top 1 million des deux listes connues *Alexa* et *Cisco Umbrella*. Afin de pouvoir croiser les données entre la blacklist et la whitelist, nous normalisons les noms de domaine au format *SLD.TLD* (figure 6.3). Au total, la whitelist compte 1061933 noms de domaine uniques.

Croisement des domaines blacklist et whitelist

Lorsque nos deux types de listes sont prêtes, celles-ci sont croisées. Théoriquement, la présence de noms de domaine de la blacklist dans la whitelist ne devrait pas nous étonner (voir 4.4.2). Dans la figure 6.4, nous mettons en évidence la présence à hauteur de 0.10038% des domaines blacklisté dans la whitelist ainsi qu'un échantillon des noms de domaine. Dès lors, une question se pose quant à l'appartenance d'un domaine présent dans les deux listes. Nous décidons de retirer tout simplement ce domaine de la whitelist pour éviter l'augmentation du taux de False Negatives [3]. Lors de la labélisation, ce nom de domaine apparaissant au moins une fois dans une blacklist sera considéré comme malicieux.

```
Intersect domain names number between whitelist and blacklist: 1066
Intersect domain names pourcentage of blacklist data in whitelist: 0.10038298084719094%
Intersect domain names list between whitelist and blacklist:
 - zdusercontent.com
 - hol.es
 - smedia.com.au
 - ure.es
 - paopaoche.net
 - usp.br
 - githubusercontent.com
 - iinet.net.au
 - undip.ac.id
 - furiousgold.com
 - baidu.com
 - ldii.or.id
 - mediafire.com
 - aolcdn.com
 - numediamarketing.com
 - ksmobile.com
 - apf.asso.fr
 - railfan.net
 - beget.tech
 - mailchimp.com
```

FIGURE 6.4 – Intersection des noms de domaine de la whitelist et la blacklist

2. FQDN ou nom absolu est la partie de gauche d'une requête Web par exemple storage.googleapis.com de l'URL *https ://storage.googleapis.com/web-sro/PS219368530BR.zip*

3. Un domaine malicieux est labélisé comme légitime

Labélisation des requêtes

Le résultat de la labélisation de requêtes est montré dans le top 20 des requêtes les plus populaires à la figure 6.5. Les domaines légitimes semblent être correctement labélisés. Par contre, nous remarquons un taux de False Positives [4] élevé avec, entre autres, le nom de domaine *google.com* labélisé comme malicieux. Après vérification, la raison est l'impact du retrait de l'intersection entre la whitelist et la blacklist de la whitelist. De plus, couplé à l'intégration des URLs, ceci biaise l'évaluation de domaines malicieux.
Cette première labélisation ne répond pas à nos attentes. Elle prouve toute la difficulté à labéliser correctement des requêtes sur base du protocole DNS, de listes de références et des noms de domaine.

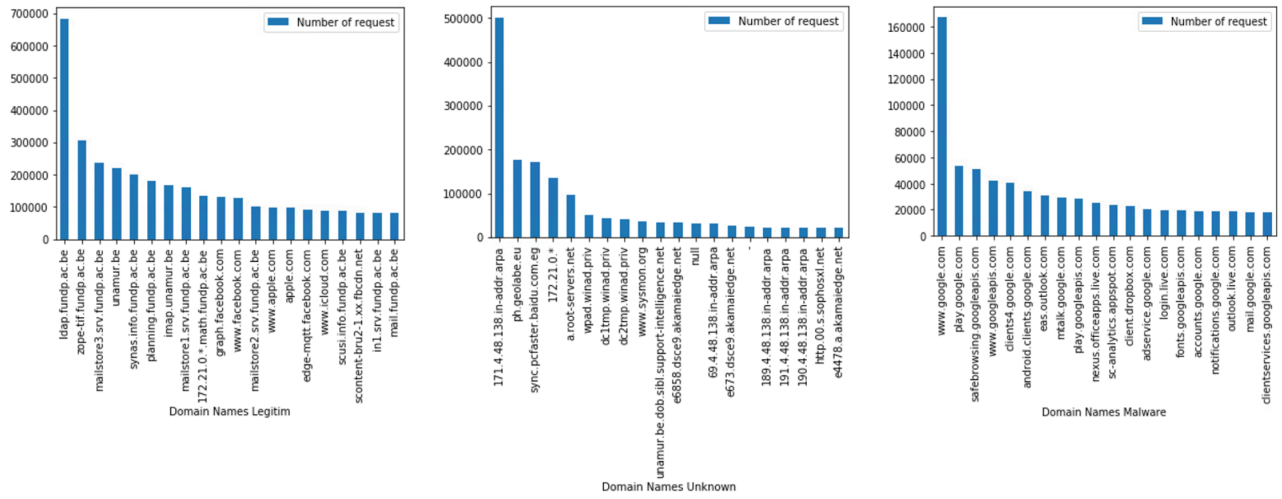

FIGURE 6.5 – Résultats d'une première tentative de labélisation

Notre regard s'est donc porté sur le contenu, puis le croisement des listes de références. En conséquence, si nous voulons éviter les False Positives, nous devons exclure les URLs et être conservateur des données des listes de références. Dans ce cas, il faut accepter être dans l'impossibilité de labéliser les liens directs vers des fichiers (*https ://storage.googleapis.com/websro/PS219368530BR.zip*) abritant des malwares. À titre d'exemple, la figure 6.6 montre le nom de domaine *googleapis.com* présent dans les blacklists d'URLs et la whitelist. Comme nous l'avons vu au chapitre 1, le protocole DNS se charge de la résolution de noms de domaine mais ne prend pas en compte le reste de l'URL. Nous rencontrons un problème pouvant être résolu à l'aide de méthodes de statistiques. L'étude *Exposure* 3.4 propose une de ces solutions ; cependant, l'implémenter demande beaucoup plus de données (deux mois et demi pour *Exposure*).

4. Un domaine légitime est labélisé comme malicieux

FIGURE 6.6 – Nom de domaine *.googleapis.com* dans les blacklists et la whitelist Alexa

6.4.2 Conclusion

Lors de notre première expérience, notre volonté est de labéliser dans une capture un maximum de noms de domaine étant connus comme malicieux ou légitimes. Cependant, nous avons été trop optimistes quant à la labélisation au moyen de toutes les blacklists incluant les URLs. De fait, le protocole DNS ne fournit pas assez d'informations dans une requête pour pouvoir labéliser les URLs malicieuses. L'information contenue dans une requête DNS contient uniquement le FQDN [5]. Même si des solutions existent (étude 3.4), nous n'avons pas assez de données DNS pour les appliquer. Dès lors, dû à un taux de False Positives élevé, cette piste est écartée pour repartir dans une direction différente. De fait, dans la section suivante nous écartons les blacklists contenant des URLs et nous sommes conservateur des blackists et whitelists téléchargées. L'exclusion des blacklists d'URLs peut biaiser nos résultats vu qu'un domaine peut être marqué légitime alors qu'il stocke un fichier malicieux (ex : dropbox.com).

5. FQDN ou nom absolu est la partie de gauche, par exemple storage.googleapis.com, de l'URL *https ://storage.googleapis.com/web-sro/PS219368530BR.zip*

6.4.3 Labélisation de requêtes avec un minimum de FP et FN

Construction de la blacklist et de la whitelist conservatrices

Les résultats de la première expérience ne répondant pas à nos attentes, nous effectuons une labélisation excluant les blacklists contenant des URLs et restons plus conservateurs au niveau des données de ces listes. Nous partons des blacklists et du top 1000 de la whitelist Alexa (comme dans l'étude d'*exposure* [45]) telles qu'elles sont fournies. Ensuite, nous éliminons toutes les entrées apparaissant dans les deux types de listes pour éviter un taux de False Positives et de False Negatives élevé. De fait, un FQDN présent dans les deux listes pourrait biaiser notre labélisation. Dans notre cas, un tel FQDN ne sera pas labélisé. Dans le tableau 6.1, nous croisons les données des différentes whitelists avec la blacklist contenant 375851 entrées. Dans la première partie le croisement est basé sur les FQDN tandis que dans la deuxième partie, le croisement est effectué sur la généralisation au SLD. Sans appel, la comparaison sur le FQDN réduit le nombre de données présentes dans les deux listes. Alexa est en tête de liste avec seulement un FQDN blacklisté dans le top 1 million. Le dataset de Cisco Umbrella semble intéressant pour son top 100000 ; par contre, il contient 11 FQDN dans la suite de la liste. Tranco, QuantCast et Majestic sont semblables et moins intéressants au vu des FQDNs contenus dans le top 100000. À savoir que QuantCast privatise des noms de domaine dans son dataset et fournit un top 486361 et non 1 million. Du côté de l'union entre SLD, nous ne sommes pas surpris des résultats exceptés pour QuantCast. De fait, il est étonnant de voir une union nulle entre QuantCast et les blacklists au format SLD pour son top 10000.

Top	Alexa	Cisco	Tranco	QuantCast	Majestic
FQDN					
500	0	0	0	0	0
1000	0	0	0	0	0
10000	0	0	0	0	1
100000	0	0	2	3	2
1000000	1	11	8	4 [6]	7
SLD					
500	1	1	2	0	2
1000	1	1	2	0	3
10000	3	3	5	0	5
100000	6	8	11	8	11
1000000	13	24	22	14 [7]	19

TABLE 6.1 – Croisement des blacklists et whitelists sans les URLs.

Résultats labélisation de requête

La figure 6.7 montre le résultat de la labélisation de requêtes sur la capture *1384844*. Les résultats affichent le nombre de requêtes labélisées comme légitimes, malicieuses en relation avec une IP malicieuse, malicieuses en relation avec un nom de domaine malicieux et les requêtes non-labélisées. C'est à hauteur de 0.0503%, soit 15028 requêtes sur 29856467, que notre outil labélise les requêtes.
Dans le graphe de gauche de la figure 6.8, le top 20 des noms de domaine légitime semble être correct. Dans le graphe de droite, on aperçoit en tête un nom de domaine qui, a priori, semble légitime *constructivubum.fr*. Cependant, l'adresse IP de résolution est quant à elle malicieuse.

6. QuantCast est composé d'un top 486361 et non 1 million
7. QuantCast est composé d'un top 486361 et non 1 million

Si l'on observe plus loin dans le graphe, l'outil labélise *www.ecolonamur.be*, *www.religieux-saintchristophe.be* ou encore *theatrejardinpassion.be* comme malicieux ! Une analyse en profondeur est réalisée ci-dessous.

```
Capture name:/hdd/maliciousFlowsDetection/capture/1384844/dns.log
 - Started at:  2018-10-16 08:00:03
 - Finish:      2018-10-17 08:03:40
 - Number of requests:  29856467
Whitelist level:1000
Number of unique FQDN:1100623
Scan result information:
 - LEGIT: 402
 - MALICIOUS_DOMAIN: 1
 - MALICIOUS_IP: 151
 - UNKNOWN: 1100069
Labelised %: 0.05033512837729177
```

FIGURE 6.7 – Résultats de labélisation à l'aide des blacklists et de la whitelist

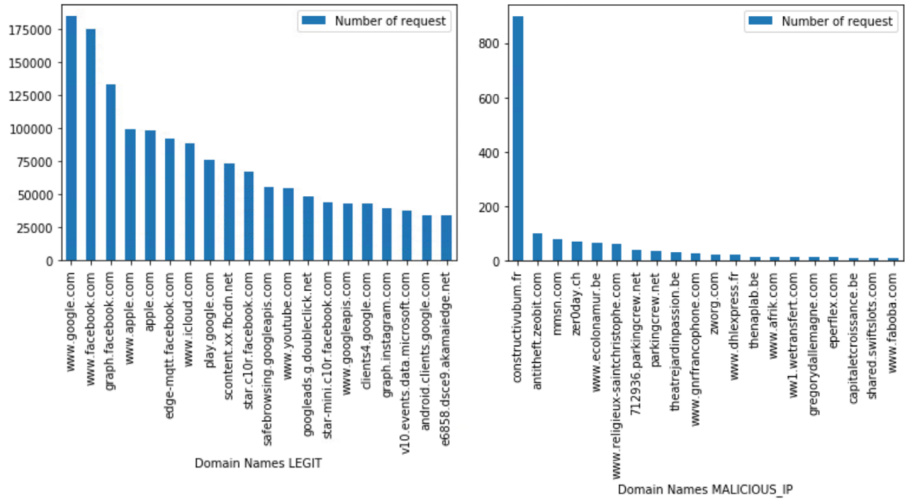

FIGURE 6.8 – Affichage du top 20 des noms de domaine les plus populaires considérés légitimes et malicieux

Analyse des requêtes malicieuses

Un extrait du fichier *labelled_ data.csv* (listing 6.2), permet de visualiser les requêtes individuellement. À gauche, on retrouve la liste des adresses IP des sources émettant les requêtes et a priori infectées ou prochainement infectées. Dans les six premières lignes, on perçoit six FQDN menant à une adresse IP unique *87.98.154.146*. Notre curiosité nous amène à analyser et à comprendre ce phénomène. D'après le résultat d'un *whois*[8], l'adresse IP *87.98.154.146* correspond à un cluster du Registar[9] nommé OVH. À l'aide de *wireshark* (voir section 4.1), nous découvrons que l'adresse IP est en réalité une adresse menant à plusieurs sites web. Ce mécanisme est appelé Server Name Indication (SNI). Notre démarche est détaillée dans le point suivant.
Côté FQDN, nous labélisons *pres.serverhome.com* comme malicieux ce qui est confirmé dans la figure 6.9 par *virustotal.com*.

8. whois est un protocole pratique utilisé dans le but d'accèder aux données d'enregistrement des noms de domaine https ://whois.icann.org/fr/histoire-du-whois
9. Hébergeur de sites web

```
rbaudin@ubuntu07:/hdd/maliciousFlowsDetection$ cat output/labelled_data.csv | grep "ip_source\|MALICIOUS"
ip_source       query                   answers          ip_malicious     label
167.247.108.211 ici2018.be              87.98.154.146    87.98.154.146    MALICIOUS_IP
167.247.228.117 www.ecolonamur.be       87.98.154.146    87.98.154.146    MALICIOUS_IP
167.247.198.63  www.gnrfrancophone.com  87.98.154.146    87.98.154.146    MALICIOUS_IP
167.247.228.83  theatrejardinpassion.be 87.98.154.146    87.98.154.146    MALICIOUS_IP
167.247.198.63  thenaplab.be            87.98.154.146    87.98.154.146    MALICIOUS_IP
167.247.214.241 geluck.com              87.98.154.146    87.98.154.146    MALICIOUS_IP
167.247.198.63  712936.parkingcrew.net  185.53.179.29    185.53.179.29    MALICIOUS_IP
167.247.198.63  783669.parkingcrew.net  185.53.179.29    185.53.179.29    MALICIOUS_IP
167.247.198.63  363063.parkingcrew.net  185.53.179.29    185.53.179.29    MALICIOUS_IP
167.247.198.63  parkingcrew.net         185.53.179.29    185.53.179.29    MALICIOUS_IP
167.247.90.98   www.facw.be             712936.parkingcrew.net,185.53.179.29     185.53.179.29    MALICIOUS_IP
167.247.90.98   www.portail-du-chocolat.be      712936.parkingcrew.net,185.53.179.29     185.53.179.29
    MALICIOUS_IP
167.247.198.63  zer0day.ch              103.224.182.250  103.224.182.250  MALICIOUS_IP
167.247.198.63  ns02.tcpcloud.eu        185.53.178.6     185.53.178.6     MALICIOUS_IP
167.247.198.63  ua1.lunrac.com          217.12.199.90    217.12.199.90    MALICIOUS_IP
167.247.198.63  pres.serverhome.com     141.8.230.20     -                MALICIOUS_DOMAIN
```

Listing 6.2 – Extrait des requêtes malicieuses du fichier *labelled_ data.csv*

L'ensemble des domaines et un extrait des adresses IP sont vérifiés dans des outils (symantec.com, ransomwaretracker.abuse.ch et virustotal.com) en ligne. Ce tableau prouve une fois de plus la difficulté à blacklister une adresse IP car l'impact peut être néfaste aux sites hébergés sur les mêmes serveurs. Ce qui est le cas pour *www.ecolonamur.be*. En d'autres termes, si un utilisateur est dans le réseau incluant un serveur de noms récursifs étant lui même configuré à l'aide de blacklists basées sur des adresses IP, alors cet utilisateur peut voir l'accès à ce type de site refusé.

IP ou Domaine Malicieux	requête DNS	propriétaire	pays	symantec.com	ransomwaretracker.abuse.ch	virustotal.com
87.98.154.146	ici2018.be,www.ecolonamur.be,www.gnrfrancophone.com,theatrejardinpassion.be,thenaplab.be...	OVH, FR	France	sending spam, unauthorized to send email directly to email servers	Distribution Site, Locky	Malicious, Phishing site
185.53.179.29	712936.parkingcrew.net, 783669.parkingcrew.net, 363063.parkingcrew.net,...	TEAMINTERNET-AS, DE	Allemagne	sending spam, unauthorized to send email directly to email servers	Distribution Site, Locky,GlobeImposter	-
103.224.182.250	zer0day.ch	TRELLIAN-AS-AP	Australie	-	Distribution Site, Locky,GlobeImposter,Payment Site, TorrentLocker	Malicious, Suspicious site
185.53.178.6	ns02.tcpcloud.eu	TEAMINTERNET-AS, DE	Allemagne	sending spam, unauthorized to send email directly to email servers	Distribution Site, Locky	-
217.12.199.90	ua1.lunrac.com	ITLAS, UA	Ukraine	-	Botnet C&C, Locky	Malicious site
pres.serverhome.com	pres.serverhome.com	-	Russie	-	-	Malicious, Phishing site

FIGURE 6.9 – Confirmation des requêtes malicieuses

Server Name Indication - SNI

Dans le point précédent, nous mettons en évidence l'utilisation d'une même adresse IP pour plusieurs sites web. Pour comprendre ce phénomène, à l'aide de *wireshark* (voir section 4.1), nous analysons les requêtes DNS puis HTTPS et nous détaillons l'usage du protocole SNI. *Server Name Indication* est une extension du protocole TLS. Il répond à la limitation de plus en plus inévitable des adresses IPv4 sur Internet au vu de son expansion. Pour un hébergeur de contenu web, il est commun de placer plusieurs sites web derrière une même adresse IP. Cependant, lors de l'initiation de la connexion, le processus se complique. De fait, le protocole TLS ne prévoit pas l'échange et la vérification de plusieurs certificats digitaux (pour le protocole HTTPS) derrière une adresse IP unique. C'est pourquoi l'extension SNI apporte des informations supplémentaires lors de la connexion. Techniquement, lors de l'initiation du protocole TLS, l'extension SNI indique le nom de l'hôte du serveur à joindre dans le message de type CLIENT HELLO [10] (figure 6.10). Le serveur prend alors connaissance du site web à retourner à l'initiateur de la requête et peut y joindre le certificat adéquat. [26]

10. CLIENT HELLO est le premier message envoyé lors de l'initiation de la communication via le protocole TLS

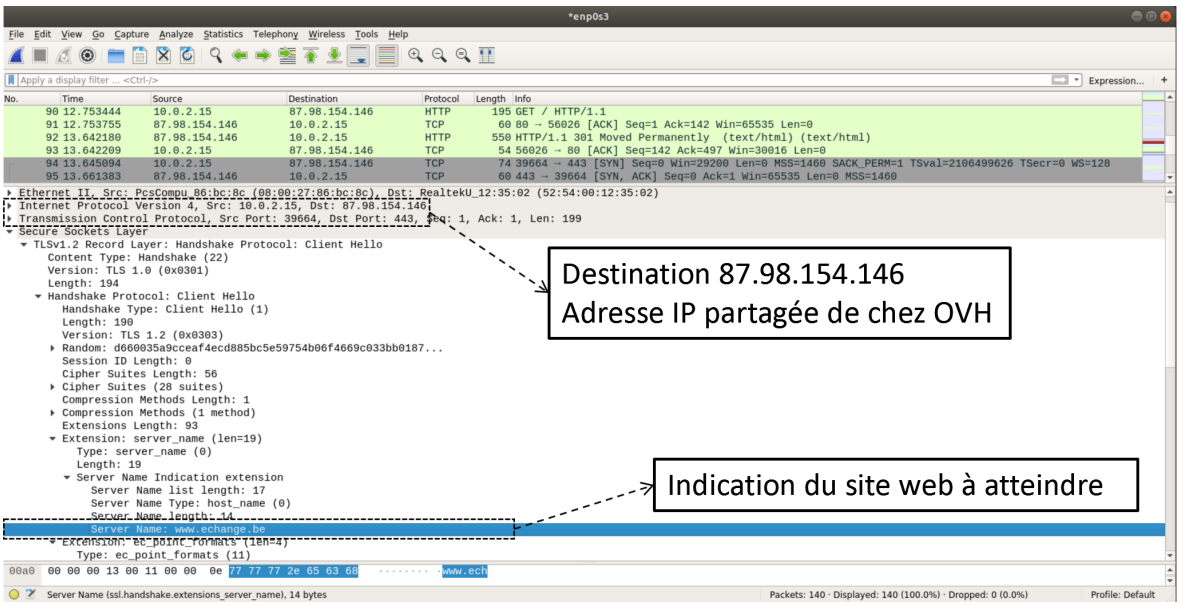

FIGURE 6.10 – Détail de l'extension SNI

6.4.4 Discussion

Dans cette section nous reprenons les éléments essentiels retenus lors de cette deuxième expérience. Nous commençons par discuter des résultats puis par transparence, nous évoquons les défis au cours des tests.

Résultats

Les résultats de 0.0503% de requêtes labélisées ne sont pas très prometteurs. Lors de notre expérience, nous ne devons pas perdre de vue certains points :

- les requêtes labélisées par des adresses IP malicieuses ne peuvent pas être prises en considération. De fait, cela augmente les False Positives du côté des noms de domaine étant eux considérés comme légitimes ;
- les noms de domaine connus pour abriter des fichiers malicieux, tels que *googleapis.com*, sont considérés comme inconnus. Effectivement, ce nom de domaine se trouve dans les deux types de listes et a pour conséquence d'être retiré de nos listes de références pour la labélisation ;
- notre whitelist est composée d'une liste de références, le top 1000 d'Alexa, afin de réduire le risque de False Negatives ;
- nous évitons la généralisation au SLD.TLD car cela biaise notre labélisation. De fait, ce n'est pas parce qu'un FQDN *www1.malwaresite.com* est malicieux que son SLD.TLD *malwaresite.com* l'est également [11] ;
- le croisement entre QuantCast et les blacklists au format SLD est nul. Compte tenu de ce résultat, QuantCast peut être un candidat pour labéliser au format SLD.

11. http ://www.malwaredomains.com/wordpress/ ?page_id=6 *"There needs to be an exact entry in the hosts file for each malware host/domain combination."*

Défis

Dans nos expériences, le code utilisé a tout d'abord été testé sur une capture moins volumineuse. Une fois le code stable, il est appliqué sur un dataset plus conséquent. C'est alors que plusieurs défis se manifestent :

— la taille du dataset est problématique. De fait, les datasets fournis ont une taille de plusieurs GigaBytes ce qui ralentit considérablement le temps de parcours. Plusieurs essais ont eu lieu afin d'améliorer le chargement et le parcours de ces données. Pandas est une librairie qui ralentit considérablement le temps de recherche des données. C'est pourquoi, finalement, nous avons choisi des types de collections simples et adéquates en fonction des besoins comme des sets, listes, dictionnaires ;

— dans les listes de références, chaque projet organise le format de son fichier différemment. En d'autres termes, le code doit être adapté pour chaque nouvelle liste téléchargée d'Internet ;

— IDS Bro est un programme mis à jour, ce qui lui a valu d'inclure un nouveau champ "rtt" dans le fichier dns.log lors de la sortie de la version 2.6.[10] Lors de l'analyse des fichiers dns.log, il a donc fallu récupérer dynamiquement le nom des en-têtes afin d'adapter le code en fonction de la version de fichier dns.log ;

— la librairie *tldextract*[12] a largement été utilisée dans les scripts pour extraire le TLD ou le eTLD (effective TLD voir 1.2.1) efficacement.

6.4.5 Conclusion

Dans cette deuxième expérience, nous labélisons le type de trafic (malicieux ou légitime) d'une capture réseau à l'aide de listes de références. Nous utilisons les *blacklists* sélectionnées dans la section 4.4.2 à l'exception des listes d'URLs. Ensuite les whitelists (liste de classement de popularité de noms de domaine) sont comparées entre elles et sur base des meilleurs résultats et des études existantes, le top 1000 d'Alexa est sélectionné. Ces listes sont alors croisées et le résultat de leur intersection est retiré de ces deux dernières dans le but de réduire le taux de False Positives et de False Negatives. Cela ne biaise pas nos résultats mais diminue le nombre de références et donc le nombre de requêtes labélisées. Nous sommes également conservateurs des entrées de ces listes, c'est à dire que nous ne modifions pas un FQDN pour en garder son nom de domaine. En effet, si nous ne sommes pas conservateurs, cela biaise les résultats comme montré dans la première expérience de cette partie. Après nos efforts, nous avons réussi à labéliser 0.0138% de trafic malicieux[13] et 0.0365% de trafic légitime. Nous montrons également la limite des listes d'adresses IP blacklistées. De fait, un nombre de sites web légitimes sont labélisés comme malicieux car le registrar héberge des sites malicieux et utilise des mécanismes légitimes comme le *SNI*. Malgré 0.0503% de requêtes labélisées, il reste un large nombre de requêtes qui ne le sont pas. Procéder par itération et ajouter les noms de domaine dans une liste manuellement n'est pas envisageable. Une des solution, est d'utiliser des services de résolution à distance comme expliqué au point 4.4.3. La solution que nous avons choisie dans la partie suivante, est d'utiliser des données labelisées dans un environnement contrôlé fourni dans le projet MCFP [6].

12. Exemple de récupération *ExtractResult(subdomain='a.b.c', domain='fun', suffix='ac.be')* https ://-pypi.org/project/tldextract/

13. pourvu d'un False Positive élevé ; voir section 6.4.3

6.5 Partie 2 : Classification à l'aide de machine learning

Dans cette section, nous détaillons la deuxième partie du pipeline illustré à la figure 6.1. La succession des étapes met en place un algorithme de classification permettant de classer les requêtes non-labélisées (ou inconnues), contenues dans le reste de la capture du réseau de l'UNamur de la première partie au 6.4. De plus, le taux de labélisation étant relativement bas dans la première partie 6.4, nous commençons par expliquer la manière dont on récupère des datasets supplémentaires. Ensuite ces datasets, à l'aide de leur fichier d'informations, sont labélisés. Ce processus consiste à labéliser chaque requête contenue dans le dataset comme "botnet" ou "normal", celles-ci servant à l'apprentissage des algorithmes. Lorsque les datasets sont labélisés, les données à extraire sont choisies et préparées sur base d'une liste de features. La liste de features est composée de paramètres permettant de distinguer une requête normale d'une requête de botnet. Les données préparées sont alors utilisées pour entraîner des algorithmes. Ceux-ci sont ensuite validés, testés et évalués dans leur capacité à distinguer une requête légitime d'une requête de botnet. Enfin, l'algorithme le plus performant est utilisé pour classer chaque requête de la capture de l'UNamur soit légitime, soit malicieux. Finalement, les résultats sont affichés, explorés et nuancés.

6.5.1 Le pipeline

Comme l'ont montré les différentes études et leurs pourcentages de réussite, le machine learning est un outil prometteur en terme de détection de trafic botnet. Nous avons également vu dans le chapitre Machine Learning (chapitre 5) les deux types d'apprentissage, à savoir le supervisé et le non supervisé. L'apprentissage non supervisé a pour objectif de prédire des groupes (*clusters*) en fonction du vecteur de features ; cependant, ces clusters restent non-labélisés et donc indéfinis. Or, dans notre cas, nous définissons deux catégories distinctes à savoir "normal" et "botnet" [14]. Puis, compte tenu de ces deux catégories, notre objectif est de classer une requête dans l'un de ces deux groupes. C'est sur base de cette distinction que l'apprentissage supervisé, et plus particulièrement la classification, est préférée dans nos expériences.
La construction d'un algorithme de classification est une tâche ardue. De fait le pipeline, présenté à la figure 6.1, décrit les huit étapes nécessaires avant de pouvoir parcourir notre échantillon du campus de l'UNamur et classer ses requêtes. Chaque étape fait place à des réflexions, guidées par les lectures de références scientifiques et les objectifs de ce projet. Les sections suivantes reprennent les éléments essentiels de chaque étape du pipeline.

6.5.2 Labélisation de datasets de références

Notre première idée est de bénéficier des données détectées dans la partie 1, donc labélisées, pour les utiliser dans l'apprentissage de nos algorithmes. Cependant, les pourcentages de labélisation sont bien trop faibles. De fait, d'une part la capture de l'UNamur ne possède pas assez de requêtes malicieuses et d'autre part, le trafic légitime labélisé à l'aide du top 1000 de whitelist ne fournit que 402 requêtes labélisées (6.4.3). Or, pour entraîner des algorithmes de classification, plus on possède de données, meilleures sont les performances [15]. C'est pourquoi, dans la section 4.4.2, nous mentionnons la possibilité de récupérer des données supplémentaires pouvant être labélisées. Une fois entraînés, les algorithmes généralisent des modèles sachant distinguer une requête malicieuse d'une requête normale.

14. "normal" faisant référence à une requête légitime et "botnet" à malicieux
15. https ://becominghuman.ai/cheat-sheets-for-ai-neural-networks-machine-learning-deep-learning-big-data-678c51b4b463

C'est dans cet objectif, que nous analysons tous les datasets du projet MCFP[6]. Nous déduisons, d'après les fichiers *readme.html*, que les datasets CTU-Malware-* peuvent contenir du trafic botnet et normal, alors que les datasets CTU-Normal-* sont entièrement composés de trafics légitimes. N'ayant besoin que du trafic DNS, nous téléchargeons le fichier *dns.log* du répertoire */bro* (voir 4.3). Bien que les fichiers IDS Bro soient générés, ils ne sont cependant pas labélisés. Pour cela, nous récupérons le ficher *readme.html* contenant toutes les informations du dataset. L'étape de labélisation consiste, pour chaque fichier *readme.html* de chaque dataset, à localiser la ligne correspondant à l'IP de l'hôte infecté (la récupération se fait dans un format *"- Infected host : 192.168.1.130"*). Or, les fichiers *readme.html* peuvent contenir un format différent ou inclure plusieurs adresses IP. Dès lors, un travail manuel est nécessaire afin de compléter la base de données des IPs normales ou infectées et ainsi labéliser complètement les datasets. L'étape de labélisation terminée, nous comptabilisons le nombre de données obtenues pour d'une part, le trafic de botnet et d'autre part, le trafic normal. À l'aide du graphe 6.11, nous mettons en évidence la différence de quantité de données mises à disposition entre les requêtes "botnet" (+/- 992000) et les requêtes "normal" (+/- 20000). En conclusion, pour obtenir un échantillon de trafic botnet et normal équilibré et éviter de biaiser les résultats (section 5.4.3), l'échantillon prélevé dans le trafic botnet aura la même quantité de données (+/- 20000 requêtes).

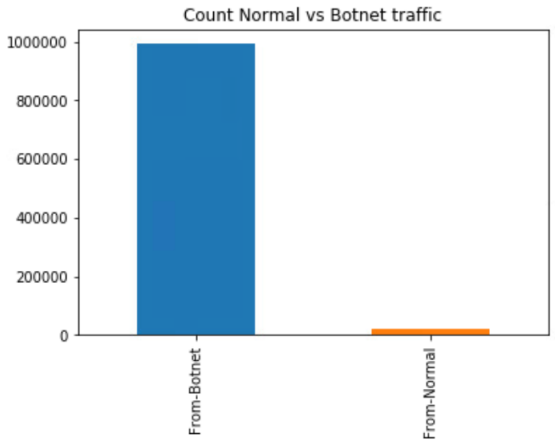

FIGURE 6.11 – Nombre de requêtes de botnets par rapport aux requêtes normales

6.5.3 Création de features

Une fois les datasets labélisés, nous procédons à la sélection du vecteur de features permettant de différencier une requête botnet d'une normale. La création de features est un moyen d'orienter l'apprentissage de l'algorithme de façon manuelle. C'est ce qui permet d'injecter de l'expertise dans les modèles [16]. De manière générale, la recherche de feature est un processus demandant beaucoup de travail. De fait, elle dépend de ce que l'on cherche à classer. Dans notre cas, nous nous focalisons sur les techniques utilisées par les botnets (et donc créés par les botmasters) pour esquiver les techniques de détection. Les deux techniques reprises pour notre expérience sont le Fast-Flux et le Domain-Flux. En effet, comme décrit à la section 2.2.5, le comportement vis-à-vis du protocole DNS des botnets peut être différencié d'un comportement normal. De plus, les différentes études réalisées nous donnent des indications quant aux features à sélectionner. D'après ces dernières, deux classes peuvent être différenciées. La première est la classe réseau, s'associant aux caractéristiques typiques des paramètres réseau d'une requête

16. Un modèle est un algorithme de machine learning entraîné à partir de données

DNS. Dans le cas des botnets, elle permet de caractériser la technique du Fast-Flux. La seconde classe, est la classe lexicale. Elle est typiquement liée, dans le cas des botnets, au FQDN contenus dans la requête à résoudre auprès du serveur DNS.

De manière générale, les features peuvent se calculer de deux manières différentes. Soit sur une donnée contenue dans une requête ; dans notre cas, la longueur d'un nom de domaine (feature F07 du tableau 6.3) est un bon exemple. Soit, par l'intermédiaire d'un travail préparatoire.

Dans notre cas, ce travail est réalisé en deux étapes. Etant donné le dataset de données pDNS de l'UNamur, la première étape vise à rassembler les données de toutes les requêtes concernant un FQDN. La deuxième étape consiste à calculer les features variant dans le temps. C'est le cas pour la feature de la moyenne du TTL [17] (feature F02 du tableau 6.2).

La liste de features ci-dessous est non-exhaustive et peut être complétée par d'autres features. Cependant, il ne faut pas perdre de vue que l'excès de features caractérise toutes les requêtes de "botnet" et "normal". Dans ce cas, on parle de sur-apprentissage (*overfitting* section 5.4.1) et les résultats peuvent être biaisés. De fait, le modèle peut donner de bons résultats lors des tests, mais est incapable de classer toute nouvelle requête. En conséquence, tout l'enjeu est de généraliser un modèle avec assez de features pour permettre la classification de requêtes inconnues mais sans excès pour empêcher le sur-apprentissage.

Features du Fast-Flux

La particularité du Fast-Flux est de changer d'adresse IP régulièrement pour un nom de domaine. La feature F01 est le nombre d'adresses IP unique retournées pour un FQDN. Nous avons également vu que les botnets configurent un TTL relativement petit pour éviter de rester dans le cache des serveurs de noms récursifs. C'est pour cela que la moyenne du TTL est calculée à partir de l'ensemble des TTLs des données du pDNS. En conséquence, les features suivantes peuvent en être déduites :

Feature	Détail	Référence
F01	Nombre d'adresses IP unique dans les réponses	[45]
F02	TTL moyenne	[45]
F03	TTL écart type	[45]
F04	TTL minimum	[16]
F05	TTL maximum	[16]

TABLE 6.2 – Liste de features de *Fast-Flux*

17. *Time To Live* section 1.3.3

Features du Domain-Flux

Côté *Domain-flux*, nous nous focalisons sur le nom de domaine en lui-même. À titre d'exemple, la taille du FQDN a tendance à être plus longue pour un domaine créé à l'aide de DGA (voir 2.2.5) que pour un domaine normal. D'où la création de la feature F06.

Feature	Détail	Référence
F06	La taille du FQDN	[9]
F07	La longueur du nom de domaine	[9]
F08	Le nombre de caractères numériques du FQDN	[9]
F09	Le nombre de caractères non-alphanumériques [18] du FQDN	[9]
F10	Nombre de sous-domaines	[28]
F11	Le nombre de traits d'union du FQDN	[28]
F12	La taille du plus grand sous-domaine	[28]
F13	Le nombre de voyelles du FQDN	[29]
F14	Le nombre de caractères différents du FQDN	[16]
F15	Nombre de consonnes du FQDN	[16]
F16	Le shannon entropy [19] 2LD	[16]
F17	Le shannon entropy 3LD	[16]
F18	Top 100 Alexa	[9]
F19	Top 1000 Alexa	[9]
F20	Top 10000 Alexa	[9]
F21	Top 100000 Alexa	[9]
F22	Top 1000000 Alexa	[9]
F23	Exclu du Top 1000000 Alexa	[9]

TABLE 6.3 – Liste de features de *Domain-Flux*

6.5.4 Extraction et préparation des données d'apprentissage

Sur base de la liste de features créées dans la section précédente, leurs données peuvent être extraites du datasets du projet MCFP.[6] Cependant, comme nous le mentionnons dans la section 6.5.2, le nombre de requêtes de botnets par rapport aux requêtes normales est disproportionné. Pour remédier à ce problème et éviter de biaiser les modèles, deux étapes sont à réaliser. La première étape mélange les requêtes provenant de chaque classe afin d'obtenir un dataset pourvu de requêtes aléatoires. La seconde étape est d'équilibrer le nombre de requêtes de chaque classe. Dans notre cas, le nombre de requêtes normales étant dominé (figure 6.11), c'est le nombre de requêtes de botnet qui est scindé. Par conséquent, la préparation de données fournit un dataset équilibré (figure 6.12) composé de 40778 requêtes aléatoires des deux classes (un extrait du code se trouve en annexe A.6).

18. Dans notre cas, alphanumérique correspond à l'ensemble abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789

19. L'entropy, dans le context de ce travail, représente la fréquence des caractères dans un nom de domaine. À titre d'exemple, "aaaa" aura une entropy de 0 alors que l'entropy de "abcdde" sera de 2.32 *https ://redcanary.com/blog/threat-hunting-entropy/*

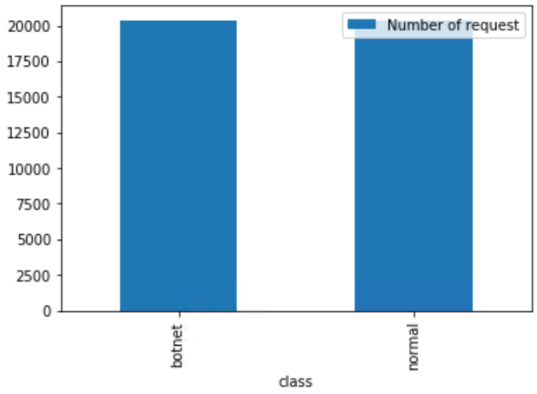

FIGURE 6.12 – Dataset d'apprentissage et de test

6.5.5 Validation des modèles

Lorsque le dataset est préparé, le modèle peut être entraîné puis validé à l'aide de la technique du *cross-validation* vue à la section 5.4.1. Pour rappel, cette méthode permet de valider un modèle en réalisant plusieurs tests (la valeur k, dans notre cas 10) à partir des données de test pour une métrique définie. Le tableau 6.4 affiche la moyenne des résultats pour chaque métrique. Le tableau a été simplifié en affichant uniquement les métriques détaillées dans la section 5.3.3. On aperçoit que l'algorithme *Random Forest* fournit la meilleure performance au vu du *f1 score* étant de 94%. Ce même algorithme possède le meilleur *recall*, 93%, ce qui induit qu'il est le plus efficace pour ne pas détecter des requêtes légitimes comme malicieuses. Dans ce tableau de résultats, plusieurs métriques sont affichées et sont toutes importantes. Par conséquent, et par souci de visualisation, la figure 6.13 fournit une vue de la moyenne des métriques par modèle du tableau 6.4.

En conclusion, cette étape fournit, sur base de la technique du *cross-validation*, les résultats des différents modèles sélectionnés. Ces résultats permettent de mettre en évidence les performances de chaque modèle et de confirmer que ceux-ci ne sont pas biaisés (voir 5.4.1). De plus, lors de cette étape, un candidat apparaît clairement pour la classification future, à savoir le *Random Forest*. Celui-ci est confirmé dans la phase de test de la section suivante.

Modèle	*Accuracy*	*Precision*	*Recall*	*F1 Score*	*AUC*
Logistic Regression	0.83	0.82	0.85	0.84	0.90
Decision Tree	0.93	0.93	0.92	0.93	0.93
Random Forest	0.94	0.94	0.93	0.94	0.98
Nearest Neighbors	0.87	0.87	0.87	0.87	0.93
AdaBoost	0.86	0.84	0.89	0.87	0.92

TABLE 6.4 – Résultats du *cross validation* des différents modèles (voir code annexe A.7)

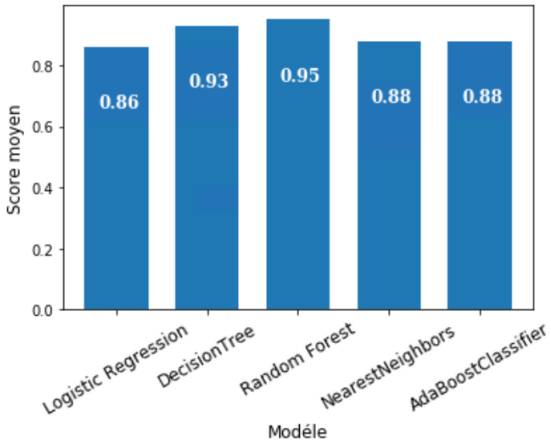

FIGURE 6.13 – Scores moyens des métriques pour chaque modèle

6.5.6 Résultats des tests des modèles

Lorsque les modèles sont validés et qu'il n'y a pas de soupçons d'un modèle biaisé, ceux-ci peuvent être testés. Le tableau 6.5 affiche les résultats des prédictions à partir des données réservées pour les tests. Sur base de ces informations, plusieurs analyses sont envisageables, comme la comparaison des résultats entre les modèles, ou le croisement des résultats pour obtenir une liste finale consolidée. Cependant, dans le cadre de cette expérience, nous sélectionnons le modèle le plus performant de toutes les catégories, à savoir le *Random Forest*.

En conclusion, le *Random Forest* semble être un candidat pour le classement des requêtes du campus de l'UNamur. Mais dans un premier temps, nous allons l'analyser pour mieux comprendre comment il interprète, puis classe les données.

Modèle	*Accuracy*	*Precision*	*Recall*	*F1 Score*	*AUC*
Logistic Regression	0.84	0.83	0.86	0.84	0.91
Decision Tree	0.93	0.93	0.93	0.93	0.93
Random Forest	0.94	0.95	0.94	0.94	0.98
Nearest Neighbors	0.87	0.87	0.88	0.88	0.93
AdaBoost	0.87	0.86	0.89	0.87	0.93

TABLE 6.5 – Résultats de la classification du *test set* sur les différents modèles

6.5.7 Détail du modèle Random Forest

Suite aux données de validation et aux données de test, le Random Forest fournit les meilleurs résultats. Ce modèle étant sélectionné, nous allons tout d'abord analyser sa matrice de confusion, ensuite analyser l'importance des features pour comprendre ses choix et enfin lister une partie des noms de domaine classés comme botnet.

Matrice de confusion

La matrice de confusion à la figure 6.14 montre le classement des requêtes DNS à partir des données de tests, évaluées par le modèle *Random Forest*. Pour la comprendre, elle peut être mise en perspective avec la représentation donnée à la figure 5.1. Une lecture en diagonale suivant les carrés bleus apporte les valeurs que l'on veut maximiser, tandis que l'autre diagonale affiche les valeurs que l'on tente de minimiser. On peut en déduire plusieurs informations. Tout

d'abord, notre modèle obtient de bons résultats puisqu'il réussit à classer 93,89% des requêtes correctement. Cette métrique n'est rien d'autre que l'*accuracy* déjà fournie dans la section précédente. Ensuite, la matrice prouve que le modèle n'est pas biaisé. De fait, il y a un équilibre entre les requêtes normales (représentées par un 0) et les requêtes de botnets (représentées par un 1). Finalement, hors des 498 requêtes classées incorrectement, 233 étaient légitimes et 265 étaient malicieuses. Pour comprendre comment le modèle classe les requêtes, la section suivante détaille l'importance des différentes features.

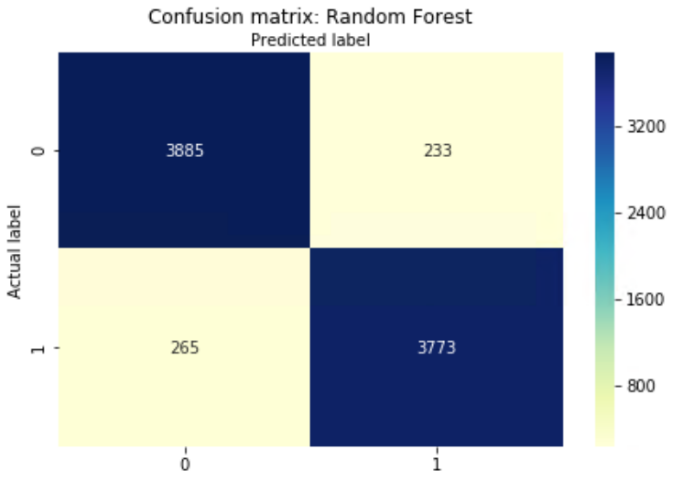

FIGURE 6.14 – Matrice de confusion du modèle *Random Forest* (voir code en annexe A.8)

Importance des features

Pour mieux comprendre le modèle, nous analysons les features prises en compte par le modèle. Une des méthodes du classificateur permet de récupérer l'ordre des features par importance. Son fonctionnement est basé sur la moyenne de l'importance des features dans chaque arbres de la forêt générée. Le tableau 6.15 classe la feature *"shannon entropy du 3LD"* comme la feature la plus importante suivie de la feature *"not_in_alexa"*. En d'autres termes, dans toutes les forêts générées, les deux features citées contribuent le plus aux décisions à prendre. Dès lors, elles sont souvent placées en tête de l'arbre.

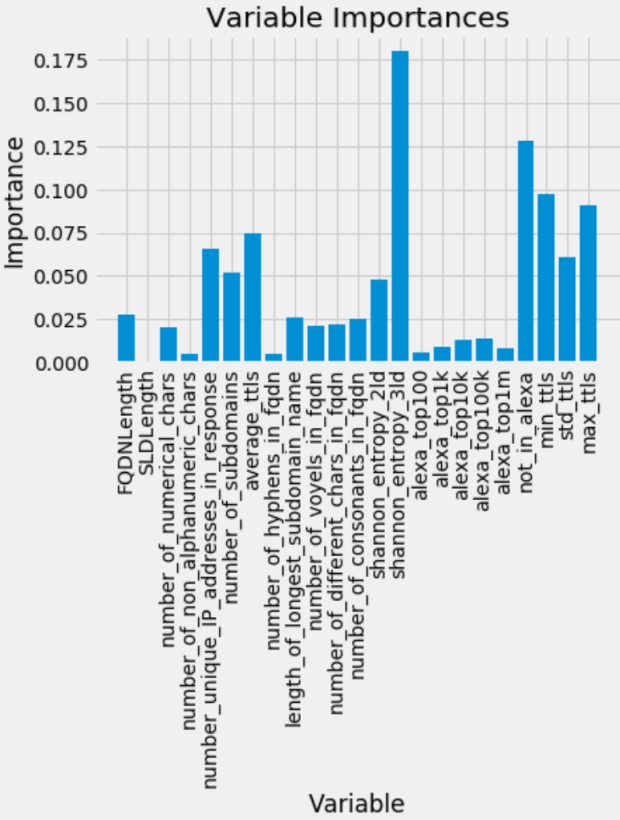

FIGURE 6.15 – Importance des features dans le modèle *Random Forest*

Hyperparamètres

Les modèles sont des algorithmes créés à partir de fonctions dans lesquelles une liste d'hyperparamètres est configurée. Ceux-ci ont une valeur par défaut, mais peuvent être ajustés pour optimiser les résultats. Heureusement, des techniques existent pour permettre de rechercher les meilleurs hyperparamètres, mais elles sont cependant très gourmandes en terme de calculs. Dans le cadre d'un *Random Forest*, nous utilisons la méthode *grid search*. L'idée est de fournir une liste de valeurs pour chaque hyperparamètres et le *grid search* se charge de trouver la meilleure configuration. Celle-ci est recherchée en comparant les résultats d'une métrique spécifiée 5.3.3. Dans notre cas, la métrique *AUC* est utilisée[20]. Dans le cas du *Random Forest*, même si les hyperparamètres peuvent être modifiés, il est également à noter que *"les paramètres par défaut des forêts aléatoires donnent souvent de bons résultats"*.[21] Cependant, si des paramètres sont à tester, les trois plus importants sont :

- *n_estimators* : est le nombre d'arbres dans la forêt. La plupart du temps il est élevé pour construire un ensemble plus robuste et permettre de réduire le sur-apprentissage ;
- *max_features* : configure le nombre de features à prendre en considération lors de chaque noeud de décision. Dans le cadre d'une classification, il est conseillé d'utiliser la racine carrée du nombre de features utilisés ;[21]
- *max_depth* : est la profondeur maximale d'un arbre. Cette valeur doit être, dans la plupart des cas élevée. Cependant, de meilleures performances peuvent être observées avec un nombre plus petit.

Dans le cadre de nos recherches, la meilleure configuration utilisée est composée de 1000 arbres dont la profondeur maximale est de 50 et où un maximum de 17 features sont sélectionnées aléatoirement pour être utilisées dans un noeud de décision(voir code en annexe A.9).

6.5.8 Analyse du dataset de l'UNamur

À l'aide du modèle sélectionné, entraîné et vérifié dans les étapes précédentes, nous classons dans cette section les requêtes de la capture de l'UNamur. Ce processus est effectué en deux étapes. La première est la préparation des données et la deuxième est le classement des données via le modèle *Random Forest*.

Préparation des données

Nous avons vu que l'échantillon de données du campus de l'UNamur est fourni au format *pcap* et transformé en fichier *dns.log* (voir section 6.4.1). Ensuite celui-ci, est analysé dans la première partie 6.4 pour labéliser les requêtes malicieuses et légitimes. Le fichier *unlabelled_data.csv* (dont un aperçu se trouve en annexe A.3) obtenu contient les informations extraites sur base de la liste des features (section 6.5.3). Ce fichier est alors introduit dans la partie 2 du pipeline de la figure 6.1. L'étape suivante est de classer une requête comme soit "botnet", soit "normal".

Classement des requêtes

Le fichier *unlabelled_data.csv* et le modèle étant prêts, les données sont fournies en entrée afin de récupérer en sortie leurs classements respectifs. Au final, 167032 requêtes sont classées comme malicieuses sur 1100229 requêtes DNS, soit 15,18% de la capture. La figure 6.16 rassemble un échantillon des noms de domaine classés comme botnet.

Au moyen du top 20 des noms de domaine des requêtes les plus présentes dans les résultats (figure 6.17), nous mettons en évidence les noms de domaine les plus souvent classés comme

20. https ://arxiv.org/pdf/1804.03515.pdf

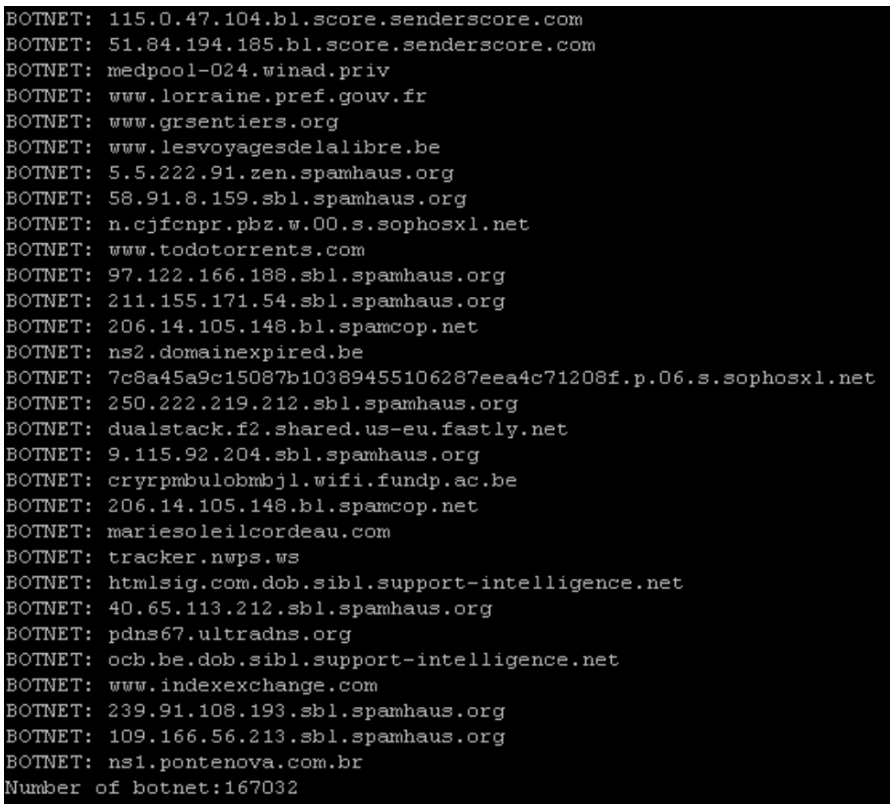

FIGURE 6.16 – Classification en tant que botnet des requêtes de l'UNamur (voir code A.10)

botnet. Après une première lecture, nous constatons que le modèle classe incorrectement des requêtes du nom de domaine *fundp.ac.be*. De fait, des requêtes de type *gbwjxxidb.wifi.fundp.ac.be* ou encore *f806t06t06t.wifi.fundp.ac.be* ressemblent au nom de domaine d'une requête botnet. De plus, nous savons que le domaine *fundp.ac.be* est légitime.

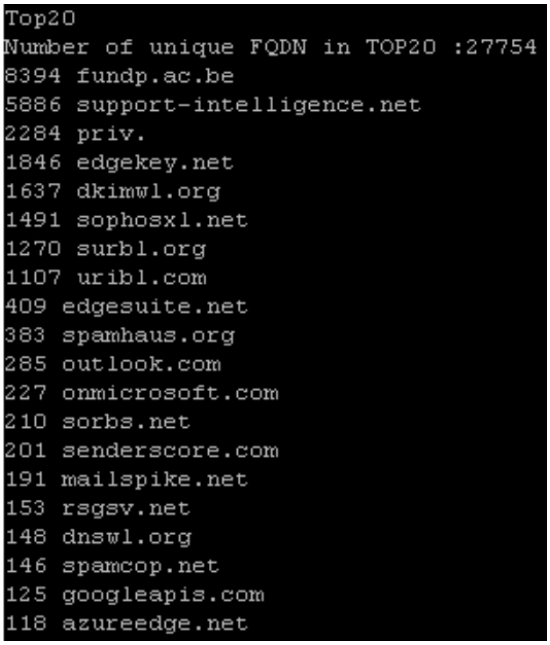

FIGURE 6.17 – Top 20 de la classification en tant que botnet des requêtes de l'UNamur (voir code A.10)

6.5.9 Conclusion

Reconstituons les étapes successives pour comprendre les résultats obtenus. L'idée principale est la détection de trafic malicieux à partir d'une capture effectuée sur le réseau de l'université de l'UNamur. Cette capture n'étant pas labélisée, l'objectif de la première partie de nos expérimentations a été de la labéliser avec un minimum de *False Positives* et de *False Negatives*. À cet égard, une première difficulté est apparue. En effet, la liste des requêtes labélisées comme malicieuses est trop limitée et elle ne peut être mise en place pour entraîner des algorithmes de machine learning. Pour remédier à ce problème, nous mettons en oeuvre des datasets labélisés faisant partie d'un projet nommé MCFP. Après cette action, nous disposons d'un dataset suffisamment conséquent pour entraîner un algorithme de machine learning. Après avoir entraîné les algorithmes, ceux-ci sont validés sur des données du projet MCFP. La phase de validation est prometteuse car elle montre un ensemble des modèles capables de distinguer des requêtes normales de requêtes de botnet avec un score moyen général de 93% dont le modèle le plus performant, 95%.

Les modèles sont alors testés sur des données réservées dont ils n'ont pas connaissance mais appartenant toujours au projet MCFP. Il en ressort des résultats satisfaisants et semblables à la phase de validation. Dès lors, le modèle le plus performant est séléctionné pour être utilisé sur des données de la capture de l'UNamur. Finalement la liste de requêtes classées ne donne pas les résultats escomptés. De fait, 167032 sur 1100229 requêtes DNS sont classées comme botnet. Pour vérifier les résultats, un aperçu aléatoire d'une liste de noms de domaine montre des requêtes légitimes et appartenant au service décrit au point 4.4.3 classées comme botnet, soit un taux de *False Positives* élevé.

Lors de notre réflexion quant aux résultats, nous avons procédé par élimination afin d'orienter d'éventuels futurs travaux vers une piste d'amélioration. Tout d'abord, nos features semblent au point sur base des résultats obtenus dans les tests des modèles (section 6.5.6). Ensuite, ces mêmes tests montrent des hyperparamètres adéquats ne requérant, dans un premier temps, aucun changement. Enfin, le nombre de requêtes de botnet montré dans la section 6.5.2 est suffisant. Par contre, l'interprétation de nos résultats prend en considération deux éléments essentiels. D'une part, les datasets du projet MCFP, et d'autre part, le dataset de l'UNamur. Le contenu des datasets joue un rôle important sur l'apprentissage des modèles. Nous pouvons caractériser ces datasets, dont le contenu est différent, comme étant de profils différents. Ainsi, il est judicieux d'entraîner un algorithme sur un profil similaire, voire même au sein du profil que l'on veut tester. Notamment, pour commencer, en reprenant uniquement le trafic normal. Or, nous avons déjà signalé, dans la première partie du chapitre 6, que labéliser un dataset avec un faible FP est une tâche ardue. De fait, une telle étape demande un nombre important de données et la mise en place d'algorithmes de réputation de noms de domaine. Une autre solution peut être de labéliser les requêtes manuellement. Cependant, cette tâche ingrate demande une expertise et du temps...

À savoir qu'il reste en suspend la question de la privatisation des données au sein du réseau, ou encore les techniques d'attaques (section 2.4) pouvant biaiser la labélisation.

Conclusion

C'est dans le cadre d'une recherche scientifique que le présent travail propose une piste de réflexion dans la détection du trafic malicieux, et plus particulièrement dans la menace dissimulée par les *botnets* dans les réseaux informatiques tels que celui de l'UNamur. Les *botnets*, à l'affût de techniques échappatoires, utilisent le protocole DNS pour améliorer leur résistance face aux systèmes de détection. Avant de démarrer ce travail, nous avions une compétence ordinaire dans le monde du langage python et une approche imprécise dans les technologies du *machine learning*. Ce travail relève d'un défi personnel et il est le fruit d'une année de travail acharné et passionné à propos de la sécurité de nos réseaux au travers du protocole DNS. Il met en perspective les techniques d'intelligence artificielle, plus particulièrement le *machine learning*.

Notre travail débute par un état de l'art du protocole DNS. Cette partie a pour objectif de comprendre l'architecture, les propriétés et le fonctionnement de ce mécanisme en place depuis la naissance d'Internet. À l'aide de ces éléments, nous situons le rôle central du DNS, gravé dans la stack de la communication des ressources réseaux. Ces ressources sont gérées dans une hiérarchie et sont confiées à des organisations dont l'avantage est d'améliorer leurs performances et leurs disponibilités de manière permanente. De plus, des mécanismes comme le *Content Delivery Network* ou le *Round-Robin* sont venus se greffer sur ce protocole et sont destinés à l'allègement du trafic circulant sur de longues distances. La gestion de ressources distribuées, couplée aux avantages et aux mécanismes apportés par le protocole DNS nous amène à réfléchir sur sa sécurité.

Le protocole DNS, véhiculant des informations inoffensives et historiquement non sensibles, n'a pas été bâti sur des fondations de sécurité. Et c'est bien là son talon d'Achille. Par conséquent, nous nous sommes interrogés sur la problématique de sa sécurité que nous envisageons en deux phases.

La première étant le vecteur d'attaques visant les serveurs DNS et ses utilisateurs. Celui-ci s'appuie principalement sur deux faiblesses du protocole DNS, à savoir son intégrité et son identité. Pour pallier ces carences, des extensions sont créées et progressivement adoptées par l'infrastructure du protocole DNS.

La deuxième étant l'exploitation du protocole DNS dans un but malicieux ou, plus concrètement, la manière dont les hackers profitent des faiblesses de ce dernier pour créer des techniques d'esquives par rapport aux systèmes de détection existants. De fait, ces techniques d'échappatoires sont tellement rapides que la vitesse de détection des systèmes existants est inversement proportionnelle à leur efficacité. Cela nous conduit vers une prise de connaissance d'un type de menace le plus répandu, à hauteur de 58%, à savoir les *botnets*. Ces derniers, consistent en un réseau d'ordinateurs infectés par des *malwares*, que nous identifions par un principe assez simple qui est de rester en communication avec leur serveur de contrôle C&C. Afin de garantir cette communication, les *botnets* utilisent principalement deux techniques qui sont, le *Fast-Flux* et le *Domain-Flux*. C'est pour cette raison que notre regard s'est principalement porté sur ces deux techniques.

Dans l'optique de la construction d'un outil de détection de trafic malicieux, nous nous sommes tout d'abord inspirés des expériences réalisées dans des études existantes. Ces approches nous ont permis de développer un canevas imbriquant une succession d'étapes, appelé le *pipeline*. La conception et la mise en place de ce *pipeline* est un parcours ardu demandant un environnement informatique solide. C'est pour cette raison que nous avons constitué de toute pièce un environnement capable de supporter la mise en œuvre du *pipeline*. De fait, une machine puissante est mise en action sur laquelle des librairies python sont exploitées. Ensuite, deux listes de références sont construites, à savoir une liste de noms de domaine malicieux et une liste de noms de domaine légitime. Lors de leurs utilisations, nous découvrons la présence de noms de domaine malicieux à hauteur de 0.1004% dans la liste légitime.

Enfin, la possibilité d'utiliser l'intelligence artificielle nous amène au concept du *machine learning*. Nous choisissons d'utiliser l'apprentissage supervisé et plus précisément la classification. Une sélection d'algorithmes de classification sont parcourus suivie de deux étapes itératives, l'évaluation et l'amélioration.

En tant qu'élément central de notre réflexion, le *pipeline*, outil de détection du trafic malicieux, se compose essentiellement de deux parties : la labélisation et la classification. L'étape de la labélisation nous initie aux types de trafic présents dans la capture de l'UNamur. En effet, dans cette partie les requêtes sont labélisées, soit comme trafic normal, soit comme trafic malicieux. Pour nous aider à orienter le choix du label d'une requête, les deux listes de références sont utilisées. Dans cette première étape, il nous paraît utile de procéder à deux expériences de labélisation. La première consiste en la labélisation de requêtes sur base de généralisation de tous les FQDN en noms de domaine. Le résultat de cette première étape fournit un taux de *False Negatives* relativement élevé. L'investigation de ce phénomène nous amène à conclure qu'une généralisation au nom de domaine ne peut être envisagée qu'à l'aide d'un historique plus conséquent de données. Ces données ainsi récoltées peuvent être utilisées afin d'en déduire des statistiques sur plusieurs critères comme la variation de noms de domaine dans le temps ou la variation du TTL dans le temps... À l'issue de nos tests, les résultats ne sont pas ceux que l'on attendait, c'est pourquoi une deuxième expérience tente de labéliser la capture de l'UNamur. Dans celle-ci, nous décidons de rester plus conservateurs quant aux listes de références utilisées. De plus, nous retirons toutes les URLs présentes dans les *blacklists* et dans les *whitelists*, en effet les URLs contiennent plus qu'un FQDN ; elles contiennent le chemin direct vers un *malware* stocké chez un hébergeur de contenu. Or, le protocole DNS ne prend en compte que le FQDN. Lors du croisement des listes de références, les résultats montrent un nombre de domaine nul pour le top 1000 des noms de domaine légitime. Nous choisissons le top 1000 d'Alexa comme base de domaine légitime pour labéliser la capture de l'UNamur. Après diverses expériences, nous réussissons à labéliser 0.0138% de trafic malicieux et 0.0365% de trafic légitime. Lors de la labélisation nous soulignons les limites de la labélisation sur base des adresses IP. En effet, celles-ci peuvent induire des *False Negatives*. Malgré 0.0503% de requêtes labélisées, nous sommes face à un manque de données pour notre partie suivante, la classification.

Au vu de la difficulté à labeliser les requêtes DNS dans la partie précédente, en complément, nous nous sommes tournés vers des *datasets* créés dans des environnements contrôlés. Ces *datasets* étant documentés, nous labélisons les requêtes légitimes et malicieuses. Ensuite, nous dressons un tableau d'une sélection de 23 *features* différenciant une requête de *botnet* d'une requête légitime. La sélection reprend des *features* de deux classes, à savoir la classe lexicale où l'accent est mis sur le nom de domaine et la classe réseau étant en relation avec les caractéristiques du protocole DNS. Puis, pour éviter tout déséquilibre dans l'apprentissage, le même nombre de requêtes malicieuses et légitimes est sélectionné. Après l'étape d'apprentissage de l'algorithme, l'étape de validation fournit un score moyen de 93% pour l'ensemble des

modèles dont le plus performant avec un score de 95%, ce qui est confirmé lors de l'étape de test. Dès lors, nous choisissons l'algorithme le plus performant, à savoir le *random forest*. Pour comprendre le fonctionnement du classement des requêtes, nous analysons la manière dont il exploite les *features*. Hors des 23 *features*, le *random forest* classe la *feature shannon_ entropy_ - 3ld* comme la plus importante de sa forêt. Ensuite, nous montrons comment nous améliorons les performances du modèle en testant plusieurs hyperparamètres dans une méthode nommée *grid search*. La configuration optimale obtenue est une forêt de 1000 arbres dans lesquels la profondeur maximale est de 50 niveaux et dont un nombre maximal de 17 *features* peut faire partie des nœuds de décision. Finalement, au vu des résultats satisfaisants des étapes précédentes d'apprentissages et des tests, nous injectons dans le modèle ainsi consolidé, notre *dataset* de l'UNamur. C'est à hauteur de 15,18% que le modèle classe le nombre de requêtes comme venant de *botnet*. Notre démarche accompagnée de ses résultats nous amène à les nuancer. Bien que la validation et la phase de test soient satisfaisantes, il n'en reste pas moins que les résultats ne sont pas ceux que nous avions escomptés.

Notre démarche suivante est alors d'interpréter nos résultats dans le contexte de ce travail. Vu le manque de requêtes labélisées dans la partie 1, induisant un apprentissage sur des données additionnelles, le modèle n'a pas connaissance de certains types de requêtes de l'UNamur. Une des pistes d'amélioration serait de labéliser ces requêtes qui sont, d'un point de vue lexical, similaires aux *botnets*, comme légitimes. À cet égard, deux options sont possibles : la première serait de construire un algorithme de réputation de noms de domaine sur base d'une capture réseau de plusieurs jours. La deuxième serait de labéliser manuellement un maximum de requêtes DNS. Une fois mise en place, la première solution a l'avantage de pouvoir labeliser un grand nombre de requêtes et toute future requête. Cependant, un de ses inconvénients est la difficulté à mettre en place un algorithme précis. La deuxième solution a l'avantage de pouvoir déterminer le nombre de requêtes labélisées après un temps déterminé, tandis que son inconvénient réside dans le fait de trouver l'expertise et le temps pour labéliser chaque requête.

Pour conclure ce travail, nous nous sommes appropriés les concepts du *machine learning* qui, antérieurement, nous étaient lointains. De plus, nous avions peu ou prou de connaissance dans le langage python. Notre parcours au travers des recherches scientifiques nous a permis de concevoir et de mettre en œuvre une démarche à suivre, le *pipeline*. Ce dernier, imbriquant toutes les étapes nécessaires au *machine learning*. Ces concepts ont été appliqués dans un outil de détection de trafic malicieux pouvant servir de base à de projets futurs. Ce travail permet également de prendre conscience des menaces planant dans les réseaux et de l'enjeu d'harmoniser les techniques de l'intelligence artificielle pour y remédier. Il se veut aussi être un outil de détection du trafic malicieux que nous serions heureux de partager avec une communauté plus large.

Bibliographie

[1] Botnet - definition - trend micro USA. https://www.trendmicro.com/vinfo/us/security/definition/botnet.

[2] Botnets threat analysis and detection. https://www.researchgate.net/publication/312037580_Botnets_Threat_Analysis_and_Detection. Accédé le 13/11/2018.

[3] dig(1) : DNS lookup utility - linux man page. https ://linux.die.net/man/1/dig. Accédé le 17/01/2019.

[4] Introduction — bro 2.6.1 documentation. https ://www.zeek.org/sphinx/intro/index.html. Accédé le 16/01/2019.

[5] ISO 3166 codes des noms de pays. http://www.iso.org/cms/render/live/fr/sites/isoorg/home/standards/popular-standards/iso-3166-country-codes.html. Accédé le 13/01/2019.

[6] Malware capture facility project. http ://mcfp.weebly.com/. Accédé le 18/01/2019.

[7] Jeffrey Aboud and Jaime Lyndon A. Yaneza. Spy-phishing – a new breedof blended threats. https://www.trendmicro.de/cloud-content/us/pdfs/security-intelligence/white-papers/wp_spyphishing_102006.pdf. Accédé le 26/02/2019.

[8] Abuse. Zeus tracker. https://zeustracker.abuse.ch/ztdns.php. Accédé le 24/02/2019.

[9] Blake Anderson and David McGrew. Identifying encrypted malware traffic with contextual flow data. https://www.bluebridge.lt/content/uploads/2018/04/Identifying-Encrypted-Malware-Traffic-with-Contextual-Flow-Data.pdf. Accédé le 04/05/2019.

[10] IDS Bro. https://www.zeek.org/download/NEWS.bro.html. Accédé le 13/02/2019.

[11] Cisco. Dns best practices, network protections, and attack identification. https://www.cisco.com/c/en/us/about/security-center/dns-best-practices.html. Accédé le 26/02/2019.

[12] Cisco. February 2019 threat repor. https://www.cisco.com/c/dam/m/digital/elq-cmcglobal/witb/1948933/CybersecuritySeries_THRT_01_0219_r2.pdf?ecid=14396&dtid=esosls000514&ccid=cc000160&oid=anrsc015216. Accédé le 25/02/2019.

[13] Cisco. Introduction to the domain name system. https ://www.cisco.com/c/en/us/td/docs/net_mgmt/prime/network_registrar/8-2/user/guide/CPNR_8_2_User_Guide/UG15_DNS.pdf. Accédé le 05/10/2018.

[14] Yin Lihua Liu Xiaoyi Cui Xiang, Fang Binxing and Zang Tiannin. Andbot : Towards advanced mobile botnets. https://www.usenix.org/legacy/events/leet11/tech/full_papers/Xiang.pdf. Accédé le 04/05/2019.

[15] Curlie. Curlie. https://curlie.org. Accédé le 10/05/2019.

[16] Pedro Marques Da Luz. Botnet detection using passive DNS. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=2ahUKEwjAwcLfzdPeAhVQIlAKHVKXCOUQFjAAegQICRAC&url=https%3A%2F%2Fwww.ru.nl%

2Fpublish%2Fpages%2F769526%2Fz-thesis_pedroluz.pdf&usg=AOvVaw3kDRYufio8Gc_s6_ZtFloX. Accédé le 13/11/2018.

[17] Damballa. Gunter ollmann, vp of research, damballa, inc. http://www.technicalinfo.net/papers/PDF/WP_Botnet_Communications_Primer_(2009-06-04).pdf. Accédé le 02/04/2019.

[18] Sai Deng. UCF research guides : Metadata : Research data & dataset. https ://-guides.ucf.edu/metadata. Accédé le 18/01/2019.

[19] Niklas Donges. The logistic regression algorithm. https://machinelearning-blog.com/2018/04/23/logistic-regression-101/. Accédé le 28/04/2019.

[20] Vincent Englebert. Cours master : Architecture Styles. Unamur, 2017.

[21] Andreas C.Müller et Sarah Guido. Le Machine learning avec python. O'Reilly, 2017.

[22] CIS Center for Internet Security. Top 10 malware january 2019. https://www.cisecurity.org/blog/top-10-malware-january-2019/. Accédé le 25/02/2019.

[23] Benjamin Edwards Tyler Moore George Stelle Steven Hofmeyr Stephanie Forrest. Beyond the blacklist : Modeling malware spread and the effect of interventions. https://www.researchgate.net/publication/221665037_Beyond_the_Blacklist_Modeling_Malware_Spread_and_the_Effect_ofInterventions. Accédé le 20/02/2019.

[24] Mozilla Foundation. Public suffix list. https://publicsuffix.org/. Accédé le 23/01/2019.

[25] Josh Fruhlinger. What is malware ? how to prevent, detect and recover from it. https://www.csoonline.com/article/3295877/malware/what-is-malware-viruses-worms-trojans-and-beyond.html. Accédé le 25/02/2019.

[26] Globalsign. What is server name indication (sni) ? https://www.globalsign.com/en/blog/what-is-server-name-indication/. Accédé le 20/04/2019.

[27] Gavin Hackeling. Mastering Machine Learning with scikit-learn. Packt, July 2017.

[28] Shuai Hao and Haining Wang. Exploring domain name-based features on the effectiveness of dns caching. https://ccronline.sigcomm.org/wp-content/uploads/2017/01/p36-hao.pdf. Accédé le 04/05/2019.

[29] Xuan Dau Hoang and Quynh Chi Nguyen. Botnet detection based on machine learning techniques using dns query data. https ://www.mdpi.com/1999-5903/10/5/43. Accédé le 21/10/2018.

[30] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C Freiling. Measuring and detecting fast-flux service networks. https://user.informatik.uni-goettingen.de/~krieck/docs/2008-ndss.pdf. Accédé le 13/01/2019.

[31] Linh Vu Hong. DNS traffic analysis for network-based malware detection. page 86.

[32] Linh Vu Hong. Dns traffic analysis for network-based malware detection. http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/6309/pdf/imm6309.pdf. Accédé le 02/04/2019.

[33] IANA. Domain name system (dns) parameters. https://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml. Accédé le 26/01/2019.

[34] ICANN. Dnssec – qu'est-ce ? pourquoi est-ce important ? https://www.icann.org/resources/pages/dnssec-what-is-it-why-important-2019-03-05-en. Accédé le 20/02/2019.

[35] ICANN. What does icann do ? https://www.icann.org/resources/pages/what-2012-02-25-en. Accédé le 23/01/2019.

[36] IETF. Domain name system security extensions. https://tools.ietf.org/html/rfc2535. Accédé le 17/02/2019.

[37] IETF. Domain names - concepts and facilities. https ://tools.ietf.org/html/rfc1034. Accédé le 03/10/2018.

[38] IETF. Domain names - implementation and specification. https ://www.ietf.org/rfc/rfc1035.txt. Accédé le 03/10/2018.

[39] ietf. RFC1591 - domain name system structure and delegation. https ://www.ietf.org/rfc/rfc1591.txt. Accédé le 13/01/2019.

[40] IETF. User datagram protocol. https ://tools.ietf.org/html/rfc768. Accédé le 03/09/2018.

[41] IETF. User datagram protocol. https ://www.ietf.org/rfc/rfc1912.txt. Accédé le 03/09/2018.

[42] Manos Antonakakis Roberto Perdisci Wenke Lee Nikolaos Vasiloglou II and David Dagon. Detecting malware domains at the upper dns hierarchy. https://www.usenix.org/legacy/event/sec11/tech/full_papers/Antonakakis.pdf. Accédé le 21/02/2019.

[43] Keycdn. What is dns spoofing ? https ://www.keycdn.com/support/dns-spoofing. Accédé le 10/10/2018.

[44] Victor Le Pochat Tom Van Goethem Samaneh Tajalizadehkhoob Maciej Korczynski and Wouter Joosen. Tranco : A research-oriented top sites ranking hardened against manipulation. https://arxiv.org/pdf/1806.01156.pdf. Accédé le 02/02/2019.

[45] Leyla Bilge Engin Kirda Christopher Kruegel and Marco Balduzz. Exposure : Finding malicious domains using passive dns analysis. https://www.cs.ucsb.edu/~chris/research/doc/ndss11_exposure.pdf. Accédé le 23/02/2019.

[46] Manos Antonakakis Roberto Perdisci David Dagon Wenke Lee and Nick Feamster. Building a dynamic reputation system for dns. https://astrolavos.gatech.edu/articles/Antonakakis.pdf. Accédé le 22/02/2019.

[47] Cricket Liu and Paul Albitz. Dns and bind. http://web.deu.edu.tr/doc/oreily/networking/dnsbind/ch01_02.htm. Accédé le 10/01/2019.

[48] Trend Micro. Defending against man-in-the-middle attacks. https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/infosec-guide-defending-against-man-in-the-middle-attacks. Accédé le 26/02/2019.

[49] Laurent Miny. Botnet detection in encrypted traffic - a machine learning approach. . Accédé le 10/10/2018.

[50] Aaron Hackworth Nicholas Ianelli. Botnets as a vehicle for online crime. https://resources.sei.cmu.edu/asset_files/WhitePaper/2005_019_001_51249.pdf. Accédé le 26/02/2019.

[51] University of Prague. Stratosphere ips project. https://www.stratosphereips.org. Accédé le 10/03/2019.

[52] Phishtank. http://www.phishtank.com/. Accédé le 25/02/2019.

[53] Steve Ragan. Three types of dns attacks and how to deal with them. https://www.csoonline.com/article/2133916/three-types-of-dns-attacks-and-how-to-deal-with-them.html. Accédé le 26/02/2019.

[54] Rod Rasmussen. Do you know what your dns resolver is doing right now ? https://www.securityweek.com/do-you-know-what-your-dns-resolver-doing-right-now. Accédé le 25/02/2019.

[55] scikit-learn Developers. Scikit-learn. https://scikit-learn.org/. Accédé le 28/04/2019.

[56] Paul Simoneau. The osi model - understanding the seven layers of computer networks. https://learningnetwork.cisco.com/servlet/JiveServlet/download/15623-2-56420/OSIModel.pdf. Accédé le 19/02/2019.

[57] Sorbs. The spam and open relay blocking system. http://www.sorbs.net/. Accédé le 24/02/2019.

[58] František Střasák. Detection of https malware traffic. https://dspace.cvut.cz/bitstream/handle/10467/68528/F3-BP-2017-Strasak-Frantisek-strasak_thesis_2017.pdf. Accédé le 10/03/2019.

[59] SURBL. Surbl - uri reputation data. http://www.surbl.org/lists. Accédé le 24/02/2019.

[60] CISCO SYSTEMS. Introduction to the domain name system. https://www.cisco.com/c/en/us/td/docs/net_mgmt/prime/network_registrar/8-1/user/guide/CPNR_8_1_User_Guide/UG15_DNS.pdf/index.html. Accédé le 13/01/2019.

[61] Florian Weimer. Passive dns replication. http://www.enyo.de/fw/software/dnslogger/first2005-paper.pdf. Accédé le 03/04/2019.

[62] Stéphane Bortzmeyer Kevin Meynell Hugo Salgado Dan York and Jan Žorž. Introduction to dns privacy. https://www.internetsociety.org/resources/deploy360/dns-privacy/intro/. Accédé le 14/04/2019.

[63] ZEEK. base/protocols/dns/main.bro — zeek user manual vv2.6.1. https://docs.zeek.org/en/stable/scripts/base/protocols/dns/main.bro.html#type-DNS::Info. Accédé le 19/01/2019.

Annexes

A.1 Bro détails

Bro Logs

dns.log

DNS query/response details

Field	Type	Description
ts	time	Timestamp of the DNS request
uid	string	Unique id of the connection
id	record	ID record with orig/resp host/port. See conn.log
proto	proto	Protocol of DNS transaction – TCP or UDP
trans_id	count	16 bit identifier assigned by DNS client; responses match
query	string	Domain name subject of the query
qclass	count	Value specifying the query class
qclass_name	string	Descriptive name of the query class (e.g. C_INTERNET)
qtype	count	Value specifying the query type
qtype_name	string	Name of the query type (e.g. A, AAAA, PTR)
rcode	count	Response code value in the DNS response
rcode_name	string	Descriptive name of the response code (e.g. NOERROR, NXDOMAIN)
QR	bool	Was this a query or a response? T = response, F = query
AA	bool	Authoritative Answer. T = server is authoritative for query
TC	bool	Truncation. T = message was truncated
RD	bool	Recursion Desired. T = request recursive lookup of query
RA	bool	Recursion Available. T = server supports recursive queries
Z	count	Reserved field, should be zero in all queries & responses
answers	vector	List of resource descriptions in answer to the query
TTLs	vector	Caching intervals of the answers
rejected	bool	Whether the DNS query was rejected by the server

FIGURE 18 – Bro Logs dns.log

A.2 Code - labéliser la capture

```
def main():
    captureDSName = config.dataset_capture+"1384844/dns.log"
    # set the top 1000 for alexa
    level = 1000
    start_time = time()
    WhitelistDS.CISCO = False
    WhitelistDS.ALEXA = True
    WhitelistDS.QUANTCAST = False
    WhitelistDS.MAJESTIC = False

    # load blacklist domains
    blacklist = BlacklistDS()
    blacklist.loadData(SLD = False, urlsIncluded = False)
    blacklist.showNumbers()

    # load whitelist domains
    whitelist = WhitelistDS()
    whitelist.loadData(SLD = False, blacklist = blacklist.blacklistDS, level = level)

    # remove intersect between the backlist and whitelist
    blacklist.generateCleanBlackList(whitelist.whitelistDS)
    whitelist.generateCleanWhiteList(blacklist.blacklistDS)
    print ("Whitelist and Blacklist load: "+  str(datetime.timedelta(seconds=time() - start_time)))

    # load the capture from UNamur (dns.log)
    dnsFlowList = DNSFlowList(blacklist.blacklistDS_cleaned, blacklist.blacklistIPs, whitelist.
     whitelistDS_cleaned, level)
    start_time = time()
    print("Begin scanning.")
    dnsFlowList.scanDataset(captureDSName)
    print("Whitelist level:"+str(level))
    dnsFlowList.showScanResult()
    print ("Capture scan: "+  str(datetime.timedelta(seconds=time() - start_time)))

    # show top 20 graph for only UNKNOWN label request
    label = DNSFlowList.UNKNOWN
    dnsFlowList.top20(label)
    df = pd.DataFrame(dnsFlowList.sorted_top20)
    # set the new index to Domain Names
    df.columns = ['Domain Names '+label,'Number of request']
    df = df.set_index(df['Domain Names '+label])
    # plot the 20 first
    df.head(20).plot.bar()

    # show top 20 graph for only MALICIOUS label request
    label = DNSFlowList.MALICIOUS_IP
    dnsFlowList.top20(label)
    df = pd.DataFrame(dnsFlowList.sorted_top20)
    # set the new index to Domain Names
    df.columns = ['Domain Names '+label,'Number of request']
    df = df.set_index(df['Domain Names '+label])
    df.head(20).plot.bar()

    # show top 20 graph for only LEGIT label request
    label = DNSFlowList.LEGIT
    dnsFlowList.top20(label)
    df = pd.DataFrame(dnsFlowList.sorted_top20)
    # set the new index to Domain Names
    df.columns = ['Domain Names '+label,'Number of request']
    df = df.set_index(df['Domain Names '+label])
    df.head(20).plot.bar()

    # save all requests with its according label in plain text
    # un extrait du fichier se trouve au point "6.4.3 Labelisation de requetes avec un minimum de FP et FN"
    dnsFlowList.save_all_results_plain_text()
    # save all data unlabeled in feature format
    # un extrait du fichier se trouve en annexe "Extrait du fichier unlabelled_data.csv"
    dnsFlowList.save_unlabeled_data()
    # save all data labeled as legit in feature format
    dnsFlowList.saveLegitFeatures()
    return
```

A.3 Extrait du fichier unlabelled_data.csv

```
FQDNLength,SLDLength,number_of_numerical_chars,number_of_non_alphanumeric_chars,number_unique_IP_addresses_in_response,number_of_subdomains,average_ttls,
    number_of_hyphens_in_fqdn,length_of_longest_subdomain_name,number_of_voyels_in_fqdn,number_of_different_chars_in_fqdn,number_of_consonants_in_fqdn,
    shannon_entropy_2ld,shannon_entropy_3ld,alexa_top100,alexa_top1k,alexa_top10k,alexa_top100k,alexa_top1m,not_in_alexa,min_ttls,std_ttls,max_ttls
27,15,0,0,16,4,43853341.2866963,0,11,9,14,15,3.0957952550009344,2.845350936622437,0,1,0,0,0,0,0.0,431769436.2026625,4294967000.0
29,10,0,1,3,5,91520112.65437339,1,7,7,15,17,2.2516291673878226,3.3082708345352603,0,0,0,1,0,0,0.0,620229772.0351759,4294967000.0
26,14,5,0,7,4,104200802.66447161,0,10,9,15,9,2.6464393446710153,3.0957952550009344,0,0,0,0,0,1,0.0,660818534.7834051,4294967000.0
22,14,0,0,11,3,11702963.694822889,0,10,5,13,15,3.321928094887362,2.2359263506290326,0,0,0,0,1,0,1.0,223889797.76423675,4294967000.0
16,12,0,0,9,3,60.0,0,8,5,12,9,3.0,1.584962500721156,0,0,0,0,1,0,60.0,0.0,60.0
22,12,2,0,43,5,64725559.40993034,0,8,3,10,13,2.5,2.5032583347756456,0,0,0,0,0,1,0.0,523262790.94933015,4294967000.0
26,14,0,0,31,3,13406604.787873605,0,11,10,14,14,2.9219280948873623,3.0957952550009344,0,0,0,0,1,0,0.0,239582508.89205253,4294967000.0
37,13,3,1,1,6,4962538.825953023,1,13,8,16,20,2.9477027792200903,3.522711955680758,1,0,0,0,0,0,1.0,145892640.72998607,4294967000.0
26,11,1,0,1,5,86400.0,0,10,7,18,14,2.3219280948873626,3.521640636343319,0,0,0,0,1,0,86400.0,0.0,86400.0
21,9,0,0,2,4,163449238.37636316,0,8,3,10,15,2.3219280948873626,2.9139770731827523,1,0,0,0,0,0,0.0,821762302.89893,4294967000.0
20,11,0,0,1,4,86400.0,0,8,5,12,12,2.3219280948873626,2.4056390622295662,0,0,0,0,1,0,86400.0,0.0,86400.0
18,12,0,1,2,3,113356736.31034553,1,8,7,11,8,2.75,2.3219280948873626,1,0,0,0,0,0,0.0,688486234.7968014,4294967000.0
36,11,0,0,0,8,-1,0,8,8,15,21,2.3219280948873626,3.7516291673878235,0,0,0,0,1,0,-1,-1,-1
20,12,0,1,2,3,110493597.86836824,1,8,6,13,11,2.75,2.807354922057604,1,0,0,0,0,0,0.0,679968471.8292775,4294967000.0
18,12,0,0,2,3,18643557.97102431,0,8,6,12,10,2.75,2.3219280948873626,1,0,0,0,0,0,0.0,282348147.75734615,4294967000.0
56,17,1,2,1,4,20.0,2,33,16,25,34,3.085055102756477,4.396229408796212,0,0,0,0,0,1,20.0,0.0,20.0
221,12,119,0,0,10,300.0,0,63,15,22,78,2.5,4.064550841212766,0,0,0,0,0,1,300.0,0.0,300.0
18,13,3,0,1,3,2954859.6055369666,0,9,4,12,9,3.169925001442312,2.0,0,0,0,0,0,1,0,0.0,112579263.62475143,4294967000.0
27,16,11,1,0,6,86400.0,1,7,4,14,6,1.584962500721156,2.9219280948873623,0,0,0,0,0,1,86400.0,0.0,86400.0
13,10,0,0,1,3,1141204.3667012858,0,7,6,9,5,2.5216406363433186,1.0,0,0,0,0,0,1,0.0,69945660.24073939,4294967000.0
14,10,0,0,3,3,23345775.621348962,0,6,4,8,8,2.584962500721156,-0.0,0,1,0,0,0,0,0,0.0,315768680.0940601,4294967000.0
13,9,0,0,7,3,12902881.850003725,0,5,3,8,8,1.9219280948873623,-0.0,1,0,0,0,0,0,0.0,235025150.501935,4294967000.0
16,11,0,0,1,3,50324873.049419954,0,7,4,9,10,2.1280852788913944,2.0,0,1,0,0,0,0,0.0,462176072.5939521,4294967000.0
27,16,11,1,0,6,86400.0,1,7,4,14,6,1.584962500721156,2.7219280948873362,0,0,0,0,0,1,86400.0,0.0,86400.0
27,16,11,1,0,6,86400.0,1,7,4,13,6,1.584962500721156,2.7219280948873362,0,0,0,0,0,1,86400.0,0.0,86400.0
17,9,0,0,0,3,600.0,0,7,6,10,9,2.2516291673878226,2.5216406363433186,0,0,0,1,0,0,600.0,0.0,600.0
16,11,0,0,0,3,4,60.0,0,5,5,12,8,2.3219280948873626,2.0,0,0,0,0,1,0,60.0,0.0,60.0
19,11,1,0,1,5,86400.0,0,5,4,14,10,2.3219280948873626,2.807354922057604,0,0,0,0,1,0,86400.0,0.0,86400.0
19,11,1,0,1,5,86400.0,0,5,4,14,10,2.3219280948873626,2.807354922057604,0,0,0,0,1,0,86400.0,0.0,86400.0
25,16,9,1,0,6,86400.0,1,7,4,12,6,1.584962500721156,2.5,0,0,0,0,0,1,86400.0,0.0,86400.0
20,11,1,0,1,5,86400.0,0,5,5,15,10,2.3219280948873626,3.0,0,0,0,0,0,1,0,86400.0,0.0,86400.0
27,10,0,0,1,5,60956150.761282995,0,7,7,13,16,2.2516291673878226,3.1556390622295662,0,0,0,1,0,0,0.0,508024406.36110616,4294967000.0
22,12,0,0,1,3,1955.3181818181818,0,9,5,13,15,2.5,2.9477027792200903,0,0,0,0,1,0,22.0,1645.3717840814093,3600.0
18,11,0,0,2,3,11678328.195166163,0,7,6,10,10,2.1280852788913944,2.584962500721156,1,0,0,0,0,0,0.0,223653571.192912,4294967000.0
16,11,0,0,1,4,86400.0,0,5,4,10,9,2.3219280948873626,2.0,0,0,0,0,1,0,86400.0,0.0,86400.0
23,11,0,0,3,5,300.0,0,7,8,13,11,2.3219280948873626,3.0957952550009344,0,0,0,0,1,0,300.0,0.0,300.0
22,12,0,1,2,3,96853887.76113798,1,9,7,13,12,2.75,2.503258334775646,1,0,0,0,0,0,0.0,637651512.3589061,4294967000.0
35,12,13,0,0,9,9.942307692307692,0,8,4,17,10,2.5,3.174674231978642,0,0,0,0,0,1,1.0,0.6816306298735805,10.0
24,15,0,0,396,3,177057346.86425766,0,11,5,14,17,3.0957952550009344,2.5,0,0,0,0,1,0,0.0,853877048.6657083,4294967000.0
...
```

A.4 Visualisation d'une requête DNS avec wireshark

```
No.     Time           Source                Destination           Protocol Length Info
1109330 66055.935546134 64.102.6.247          10.48.35.97           DNS      498    Standard query response 0
    xc8ba A opac.unamur.be CNAME bib.sipr.ucl.ac.be A 130.104.5.68 NS c.ns.dns.be NS b.ns.dns.be NS d.ns.dns.
    be NS y.ns.dns.be NS a.ns.dns.be NS x.ns.dns.be A 194.0.6.1 A 194.0.37.1 A 194.0.43.1 A 194.0.44.1 A
    194.0.1.10 A 120.29.253.8 AAAA 2001:678:9::1 AAAA 2001:678:64::1 AAAA 2001:678:68::1 AAAA 2001:678:6c::1
    AAAA 2001:678:4::a AAAA 2001:dcd:7::8 OPT

Frame 1109330: 498 bytes on wire (3984 bits), 498 bytes captured (3984 bits) on interface 0
Ethernet II, Src: Cisco_2a:c4:a3 (00:06:f6:2a:c4:a3), Dst: Vmware_8d:00:0a (00:50:56:8d:00:0a)
Internet Protocol Version 4, Src: 64.102.6.247, Dst: 10.48.35.97
User Datagram Protocol, Src Port: 53, Dst Port: 43642
Domain Name System (response)
    Transaction ID: 0xc8ba
    Flags: 0x8180 Standard query response, No error
        1... .... .... .... = Response: Message is a response
        .000 0... .... .... = Opcode: Standard query (0)
        .... .0.. .... .... = Authoritative: Server is not an authority for domain
        .... ..0. .... .... = Truncated: Message is not truncated
        .... ...1 .... .... = Recursion desired: Do query recursively
        .... .... 1... .... = Recursion available: Server can do recursive queries
        .... .... .0.. .... = Z: reserved (0)
        .... .... ..0. .... = Answer authenticated: Answer/authority portion was not authenticated by the
    server
        .... .... ...0 .... = Non-authenticated data: Unacceptable
        .... .... .... 0000 = Reply code: No error (0)
    Questions: 1
    Answer RRs: 2
    Authority RRs: 6
    Additional RRs: 13
    Queries
        opac.unamur.be: type A, class IN
            Name: opac.unamur.be
            [Name Length: 14]
            [Label Count: 3]
            Type: A (Host Address) (1)
            Class: IN (0x0001)
    Answers
        opac.unamur.be: type CNAME, class IN, cname bib.sipr.ucl.ac.be
            Name: opac.unamur.be
            Type: CNAME (Canonical NAME for an alias) (5)
            Class: IN (0x0001)
            Time to live: 600
            Data length: 18
            CNAME: bib.sipr.ucl.ac.be
        bib.sipr.ucl.ac.be: type A, class IN, addr 130.104.5.68
            Name: bib.sipr.ucl.ac.be
            Type: A (Host Address) (1)
            Class: IN (0x0001)
            Time to live: 81262
            Data length: 4
            Address: 130.104.5.68
    Authoritative nameservers
        be: type NS, class IN, ns c.ns.dns.be
            Name: be
            Type: NS (authoritative Name Server) (2)
            Class: IN (0x0001)
            Time to live: 171767
            Data length: 11
            Name Server: c.ns.dns.be
        be: type NS, class IN, ns b.ns.dns.be
            Name: be
            Type: NS (authoritative Name Server) (2)
            Class: IN (0x0001)
            Time to live: 171767
            Data length: 4
            Name Server: b.ns.dns.be
        be: type NS, class IN, ns d.ns.dns.be
            Name: be
            Type: NS (authoritative Name Server) (2)
            Class: IN (0x0001)
            Time to live: 171767
            Data length: 4
            Name Server: d.ns.dns.be
        be: type NS, class IN, ns y.ns.dns.be
            Name: be
            Type: NS (authoritative Name Server) (2)
            Class: IN (0x0001)
            Time to live: 171767
            Data length: 4
            Name Server: y.ns.dns.be
        be: type NS, class IN, ns a.ns.dns.be
            Name: be
            Type: NS (authoritative Name Server) (2)
            Class: IN (0x0001)
            Time to live: 171767
            Data length: 4
            Name Server: a.ns.dns.be
        be: type NS, class IN, ns x.ns.dns.be
            Name: be
            Type: NS (authoritative Name Server) (2)
            Class: IN (0x0001)
            Time to live: 171767
            Data length: 4
            Name Server: x.ns.dns.be
    Additional records
        a.ns.dns.be: type A, class IN, addr 194.0.6.1
            Name: a.ns.dns.be
            Type: A (Host Address) (1)
            Class: IN (0x0001)
            Time to live: 171767
```

```
            Data length: 4
            Address: 194.0.6.1
        b.ns.dns.be: type A, class IN, addr 194.0.37.1
            Name: b.ns.dns.be
            Type: A (Host Address) (1)
            Class: IN (0x0001)
            Time to live: 171767
            Data length: 4
            Address: 194.0.37.1
        c.ns.dns.be: type A, class IN, addr 194.0.43.1
            Name: c.ns.dns.be
            Type: A (Host Address) (1)
            Class: IN (0x0001)
            Time to live: 171767
            Data length: 4
            Address: 194.0.43.1
        d.ns.dns.be: type A, class IN, addr 194.0.44.1
            Name: d.ns.dns.be
            Type: A (Host Address) (1)
            Class: IN (0x0001)
            Time to live: 171767
            Data length: 4
            Address: 194.0.44.1
        x.ns.dns.be: type A, class IN, addr 194.0.1.10
            Name: x.ns.dns.be
            Type: A (Host Address) (1)
            Class: IN (0x0001)
            Time to live: 171767
            Data length: 4
            Address: 194.0.1.10
        y.ns.dns.be: type A, class IN, addr 120.29.253.8
            Name: y.ns.dns.be
            Type: A (Host Address) (1)
            Class: IN (0x0001)
            Time to live: 172087
            Data length: 4
            Address: 120.29.253.8
        a.ns.dns.be: type AAAA, class IN, addr 2001:678:9::1
            Name: a.ns.dns.be
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
            Time to live: 172087
            Data length: 16
            AAAA Address: 2001:678:9::1
        b.ns.dns.be: type AAAA, class IN, addr 2001:678:64::1
            Name: b.ns.dns.be
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
            Time to live: 172087
            Data length: 16
            AAAA Address: 2001:678:64::1
        c.ns.dns.be: type AAAA, class IN, addr 2001:678:68::1
            Name: c.ns.dns.be
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
            Time to live: 172087
            Data length: 16
            AAAA Address: 2001:678:68::1
        d.ns.dns.be: type AAAA, class IN, addr 2001:678:6c::1
            Name: d.ns.dns.be
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
            Time to live: 171767
            Data length: 16
            AAAA Address: 2001:678:6c::1
        x.ns.dns.be: type AAAA, class IN, addr 2001:678:4::a
            Name: x.ns.dns.be
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
            Time to live: 171767
            Data length: 16
            AAAA Address: 2001:678:4::a
        y.ns.dns.be: type AAAA, class IN, addr 2001:dcd:7::8
            Name: y.ns.dns.be
            Type: AAAA (IPv6 Address) (28)
            Class: IN (0x0001)
            Time to live: 171767
            Data length: 16
            AAAA Address: 2001:dcd:7::8
        <Root>: type OPT
            Name: <Root>
            Type: OPT (41)
            UDP payload size: 4096
            Higher bits in extended RCODE: 0x00
            EDNS0 version: 0
            Z: 0x0000
                0... .... .... .... = DO bit: Cannot handle DNSSEC security RRs
                .000 0000 0000 0000 = Reserved: 0x0000
            Data length: 0
    [Request In: 1109299]
    [Time: 0.298623178 seconds]
```

A.5 Machine virtuelle

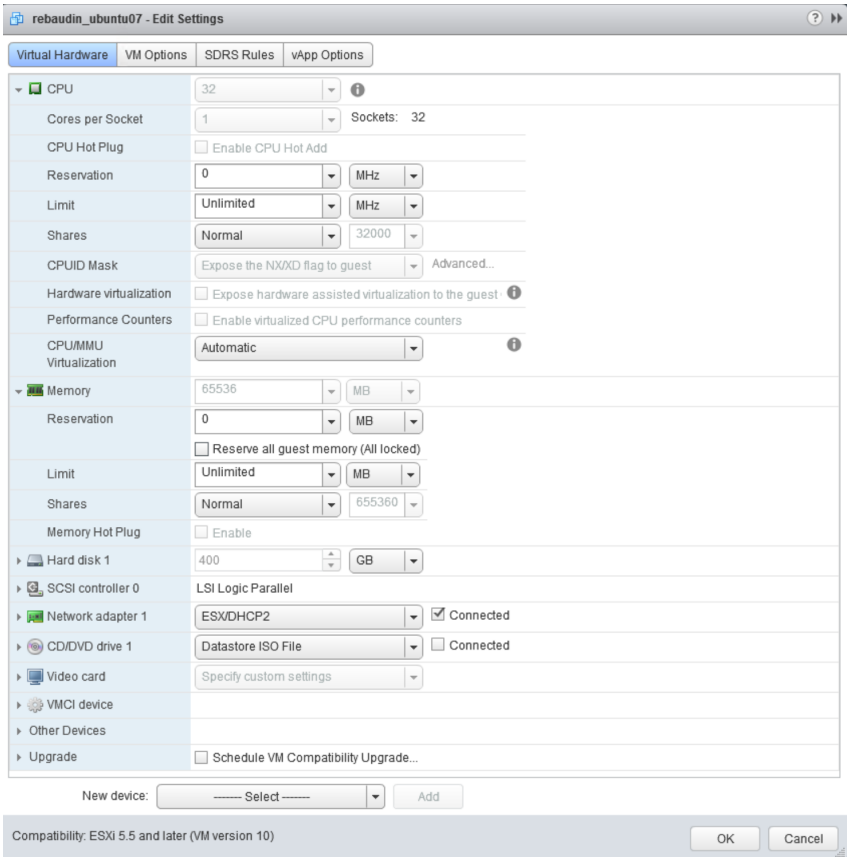

FIGURE 19 – Configuration de la machine virtuelle

A.6 Code - création, mélange et équilibrage des données

```
features_selection = ["FQDNLength","SLDLength","number_of_numerical_chars","number_of_non_alphanumeric_chars"
    ,"number_unique_IP_addresses_in_response","number_of_subdomains","\
average_ttls","number_of_hyphens_in_fqdn","length_of_longest_subdomain_name","number_of_voyels_in_fqdn","
    number_of_different_chars_in_fqdn","number_of_consonants_in_fqdn","\
shannon_entropy_2ld","shannon_entropy_3ld","alexa_top100","alexa_top1k","alexa_top10k","alexa_top100k","
    alexa_top1m","not_in_alexa","min_ttls","std_ttls","max_ttls"]

# reading extracted MCFP data from the features_data.csv
print("Reading file: features_data.csv")
# read the file. The first line is the column title and the result is a dataframe
data_df = pd.read_csv(config.output + "features_data.csv", encoding='utf-8')

# 0 = normal
# 1 = botnet
unique_labels = [0,1]
normal = data_df[data_df['label'] == 0]
botnet = data_df[data_df['label'] == 1]

print('botnet:', str(len(botnet)))
print('normal:', str(len(normal)))

# decrease botnet number of data to the normal number of data (20389)
botnet_under = botnet.sample(len(normal.index), random_state=42 )
# create a dataframe from both
df_balanced = pd.concat([botnet_under, normal], axis=0)

print('Random under-sampling:')
print(df_balanced.label.value_counts())
# result of previous line:
# 1 20389
# 0 20389

# value will be only the data without the label.
# output is a list like  [[19.0, 1.0, 0.0, 0.0, 1.0, 3.0, 1200.0, 0.0, 11.0, 3.0, 11.0, 14.0,
    2.845350936622437, -0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0],[...]...]
values = df_balanced.loc[:,features_selection].values
# labels is a list like [0 1...
labels = np.array(df_balanced['label'].tolist())
# mix/shuffle the data
X, y = shuffle(values, labels)
# create the 20/80 pourcent training and testing data
X_train, X_test, y_train, y_test = sklearn.model_selection.train_test_split(X, y, test_size=0.2, random_state
    = 42)
```

A.7 Code - validation de modèle

```
# contruct random Forest Classifier
rfc = RandomForestClassifier( n_estimators= 1000, max_features = 11, random_state = 42, max_depth=50)
# list of metrics to compare
scoring = ['accuracy', 'f1', 'recall', 'precision', 'roc_auc']
test_scoring = ['test_accuracy', 'test_f1', 'test_recall', 'test_precision', 'test_roc_auc']

# cross validate is a methode to compute several metrics
scores = cross_validate(rfc, X_train, y_train, scoring=scoring,cv=10, return_train_score=False)
print(name+" & "+str("%.2f" % scores["test_accuracy"].mean())+" & "+str("%.2f" % scores["test_precision"].
    mean())+" & "+str("%.2f" % scores["test_recall"].mean())+" & "+str("%.2f" % scores["test_f1"].mean())+" &
     "+str("%.2f" % scores["test_roc_auc"].mean())+ "\\\\")
sorted(scores)
sum_average = 0
# print average of all the metrics
for key, value in scores.items():
    print("Key:"+key + " value:"+str("%.2f" % value.mean()))
    if(key in test_scoring):
        sum_average += value.mean()

# keep 2 numbers after the comma
average_score = float("%.2f" % (sum_average/len(scoring)))
print("average_score value:"+str(average_score))
# result previous line
# Key:fit_time value:103.31
# Key:test_accuracy value:0.94
# Key:test_f1 value:0.94
# Key:test_recall value:0.93
# Key:test_precision value:0.94
# Key:test_roc_auc value:0.98
# Key: average_score value:0.95
# Cross Validation summary
```

A.8 Code - matrice de confusion

```
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# create the model
rfc = RandomForestClassifier( n_estimators= 1000, max_features = 17, random_state = 42, max_depth=50)
# fit the classifier with data
rfc.fit(X_train, y_train)
# predict data from X_test.
# y_pred is the predicted labels
y_pred = rfc.predict(X_test)
# construct the confusion matrix
cnf_matrix = sklearn.metrics.confusion_matrix(y_test, y_pred)
print(cnf_matrix)
# result of the previous line
# [[3905  159]
# [ 229 3863]]
# print result in a graph
# 0 = normal, 1 = botnet
unique_labels = [0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(unique_labels))
plt.xticks(tick_marks, unique_labels)
plt.yticks(tick_marks, unique_labels)
# heatmap creation
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix: Random Forest', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
# The result is shown in the section: "Detail du modele Random Forest"
```

A.9 Code - recherche des meilleurs hyperparamètres

```
# @param features_selection: is the list of features in string
def grid_search_RF(X_train, y_train, features_selection):
    # random forest n_jobs = 1 to avoid parallel computation
    rfc = RandomForestClassifier(n_jobs=1)
    # set a list per hyper-parameters
    param_grid_rfc = {"n_estimators":[10,50,100,500,1000] ,
                      "max_features":[2, int(math.sqrt(len(features_selection))) , 11, 17, len(
     features_selection)],
                      "max_depth" :[10,50,100,500,1000]
                      }
    # set the scoring to roc_auc because it is advised for classification model
    classifier = GridSearchCV(rfc, param_grid_rfc, cv=10, n_jobs=1, scoring ="roc_auc")
    start = time()
    # fit the classifier with data
    classifier.fit(X_train, y_train)
    print("GridSearchCV took %.2f seconds for %d candidate parameter settings." % (time() - start, len(
     classifier.cv_results_['params'])))
    # print result
    report(classifier.cv_results_)
    return

# Utility function to report best scores
# thanks to https://scikit-learn.org/0.18/auto_examples/model_selection/randomized_search.html
def report(results, n_top=10):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate]))
            print("")
```

A.10 Code - classification dataset UNamur

```
# file unlabelled_data.csv see pipeline
unlabeled_data_file_name = "unlabelled_data.csv"
print("Reading file: "+unlabeled_data_file_name)
# read the data to get the unknown data in a dataframe
data_df_unknown_data = pd.read_csv(unlabeled_data_file_name, encoding='utf-8')
# transform to <class 'numpy.ndarray'>
data_df_unknown_data = data_df_unknown_data.loc[:,features_selection].values
# read the data to get the unknown data in a list
list_unknown_data = load_unknown_data(unlabeled_data_file_name)
print(str(len(list_unknown_data)))
#print(data_df_unknown_data.tolist())
#[20.0, 1.0, 1.0, 0.0, 1.0, 5.0, 86400.0, 0.0, 5.0, 5.0, 15.0, 10.0, 2.321928094887362, 3.0, 0.0, 0.0, 0.0,
     0.0, 1.0, 0.0]

# read the file where feature line is linked to the readable information
# i.e: <[20.0, 1.0, 1.0, 0.0, 1.0, 5.0, 86400.0, 0.0, 5.0, 5.0, 15.0, 10.0, 2.321928094887362, 3.0, 0.0, 0.0,
     0.0, 0.0, 1.0, 0.0], object_DNSFlowAggregate>
print("Reading file: "+file_name_feature_to_original)
feature_to_originaldata = dict()
feature_to_originaldata = pickle.load(open(file_name_feature_to_original, 'rb'))
print("\nPredict unknown data with RF")
print("\n----------------------------")
# predict label from unlabelled_data.csv
y_pred = rfc.predict(data_df_unknown_data)
# get 30 random number to print random result
list_random_number = list()
for x in range(30):
    random_number = random.randint(1,1000)
    list_random_number.append(random_number)
bot = 0
i=0
list_botnet = list()
list_botnet_top20 = dict()
# count and treat the botnet predicted
while(i < len(y_pred)):
    if(y_pred[i] == 1):
        key = ','.join(list_unknown_data[i])
        # get back the original FQDN from the feature line
        list_botnet.append((feature_to_originaldata[key+"\n"]["dnsFlowAggregateObject"]).FQDN)
        # build <FQDN, domainName> for the top 20 graph
        list_botnet_top20[(feature_to_originaldata[key+"\n"]["dnsFlowAggregateObject"]).FQDN] = (
     feature_to_originaldata[key+"\n"]["dnsFlowAggregateObject"]).domainName
        bot += 1
    i += 1
# print random result shown in "Figure 6.16 Classification en tant que botnet des requ tes de  l UNamur "
for line in list_random_number:
    print(list_botnet[line])

# save the result to a file classified_data.csv
print("Saving file classified_data.csv")
with open("classified_data.csv", 'w') as f1:
    for key in list_botnet:
        f1.write(key+"\n")
f1.close()
print("File Saved.")
# print the top 20 shown in "Figure 6.17 Top 20 de la classification en tant que botnet des requ tes de
     l UNamur "
print("Top20")
show_top20_sld(list_botnet_top20, unlabeled_data_file_name, "RF")
print("Number of request botnet:"+str(bot)+ " and number of unique FQDN:"+str(len(list_botnet)) )

# read every line from pathfile and add them to list_unknown_data list
def load_unknown_data(pathfile):
    list_unknown_data = list()
    with open(pathfile, 'r') as csvFile:
        csvReader = csv.reader(csvFile, delimiter=',', quoting=csv.QUOTE_MINIMAL)
        i = 1
        for row in csvReader:
            if(i == 1):
                i += 1
                continue
            list_unknown_data.append(row)
    return list_unknown_data
```