ÉCOLE ROYALE MILITAIRE
167ème Promotion Polytechnique
Cdt Jean Saussez

Année Académique 2016-2017
2ème Master

Détection d'APT basée sur le traitement de graphes : analyse paramétrisable et interactive de logs de proxy

par
Sous-Lieutenant d'Aviation Candidat Officier de Carrière
Thomas GILON

Mémoire de fin d'études, Département CISS
présenté pour l'obtention du diplôme de master
en Sciences de l'Ingénieur
sous la direction du Professeur Dr. Ir. Wim MEES
Bruxelles, 2017

Préambule

À l'heure actuelle, il est évident que la cybersécurité ne peut plus être négligée. La Défense a le devoir d'y être un acteur de référence. En tant qu'Officier-Élève de l'École Royale Militaire, il me semble important de contribuer aux recherches effectuées dans ce domaine. Bien que, depuis toujours, mon affinité pour l'informatique m'ait poussé à m'informer sur le sujet, ce sont les différents cours liés à la cybersécurité suivis durant les modules CISS qui m'ont convaincu de faire mon mémoire de fin d'études sur ce sujet.

Ce mémoire de fin d'études est l'aboutissement de cinq années de travail intensif. Il est clair que je ne serais pas parvenu à la fin de ces cinq années sans le cadre académique de l'école. Grâce à son enseignement, il m'a été possible d'évoluer énormément en cinq ans. Je souhaite particulièrement remercier le Professeur Dr. Ir. Wim MEES, mon promoteur, pour ses conseils avisés et le temps consacré à ce mémoire. Je souhaite également remercier le Capitaine-Commandant d'Avi Ir. Thibault DEBATTY, mon deuxième lecteur, pour son suivi et sa disponibilité. Finalement, je remercie Georgi NIKOLOV pour son aide à la réalisation de l'interface graphique.

Je souhaite également m'adresser à l'entourage dévoué que j'ai pu avoir et souligner l'importance de l'esprit de groupe construit avec ma promotion, la 167$^{\text{ème}}$ Promotion Polytechnique. Il est évident que l'entraide et le partage à tout niveau m'ont permis de retirer un maximum de ces cinq années. Je souhaite adresser un remerciement tout particulier à ces camarades fantastiques qui se reconnaîtront bien. Finalement, sur le plan privé, je tiens particulièrement à remercier ma famille et Héloïse pour leur investissement indirect dans ma formation.

Thomas GILON

Table des matières

Table des figures

Acronymes

API Application Programming Interface
APT Advanced Persistent Threat
AUC Area Under Curve
C2 Command and Control
CCB Centre for Cyber Security Belgium
CCDCOE Cooperative Cyber Defence Centre of Excellence
CDN Content Delivery Network
CERT Computer Emergency Response Team
CSOC Cyber Security Operations Center
CSS Cascading Style Sheets
CSV Comma-Separated Values
DMZ DeMilitarized Zone
EDA European Defence Agency
HTTP HyperText Transfer Protocol
IDS Intrusion Detection System
IP Internet Protocol
JSON JavaScript Object Notation
JVM Java Virtual Machine
MASFAD Military multi-Agent System For APT Detection
NCIRC NATO Computer Incident Response Capability
OTAN Organisation du Traité de l'Atlantique Nord
OWA Ordered Weighted Average
ROC Receiver Operating Characteristic
RPC Remote Procedure Call
RUCD Research Unit Cyber Defence
TLD Top Level Domain
TOR The Onion Router
URL Uniform Resource Locator
UTC Coordinated Universal Time
WOWA Weighted Ordered Weighted Average

1. Introduction

1.1 Introduction générale

Ce travail s'intéresse à la détection d'Advanced Persistent Threat (APT). Plus précisément, ce travail se concentre sur l'étude de logs de proxy et sur la détection de domaines servant de canal de Command and Control (C2) pour une APT. L'algorithme conçu modélise le trafic contenu dans un log de proxy au moyen d'un graphe et tente de détecter une infection d'APT en cherchant une anomalie dans ce graphe. L'objectif du travail est de concevoir un algorithme permettant à un(e) analyste de travailler de manière interactive avec un ensemble de paramètres dont les valeurs auront été choisies par ses soins et qui permettront de s'adapter au mieux au contenu étudié.

Cette introduction est d'abord consacrée à la description du contexte dans lequel se cadre le travail. Cette description a pour objectif de se concentrer sur les informations relevantes pour le lecteur. Ensuite, les notions clés abordées dans le travail sont présentées et la problématique conduisant à ce travail est détaillée. Par après, un aperçu du travail précédent effectué sur le sujet est donné. Finalement, la structure du mémoire est présentée.

1.2 Contexte

La technologie de l'information évolue à grands pas. La dépendance de nos marchés économiques et de la société en général à ces nouvelles technologies est devenue non négligeable [51]. Il est maintenant commun d'avoir en permanence accès à internet. Toutefois, ce monde en plein essor n'est pas sans risque et les menaces évoluent tout aussi rapidement que ces technologies. Les incidents liés à la cybersécurité peuvent avoir de nombreux impacts qui dépassent largement le monde numérique. Ces attaques peuvent déstabiliser des services essentiels tels que l'approvisionnement en eau, les soins de santé ou encore l'approvisionnement en électricité.

Sur les 8 dernières années, 7,1 milliards d'identités ont été exposées publiquement suite à des fuites de données [56]. En 2016, de grandes campagnes de vols de données et d'espionnages ont pu être identifiées. Celles-ci nourrissent des fins telles que l'influence de l'opinion publique, l'instauration d'une atmosphère d'instabilité ou encore l'influence de résultats politiques. Les événements survenus lors des élections présidentielles américaines, fin 2016, peuvent être cités comme exemple.

Évidemment, divers niveaux de compétence existent dans les attaques menées. Cela va du simple vandalisme informatique aux attaques criminelles [51]. Parmi ces attaques, les attaques de type *Advanced Persistent Threat* font partie des attaques les plus avancées. Elles peuvent viser des capacités militaires, mais également le secteur civil. Les APT ont des capacités [31] telles que l'infection avec succès de plusieurs organisations gouvernementales, le vol d'informations pendant plusieurs années ou encore le vol d'informations provenant de réseaux isolés physiquement. L'entièreté de ces actions est évidemment effectuée sans être découverte.

Dans ce contexte de menace croissante, différents acteurs se mettent en place pour assurer la sécurité du cyberespace. Ci-après sont présentés différents projets mis en place ces dernières années dans le but d'assurer une plus grande cybersécurité, tant au niveau belge qu'au niveau européen et de l'Organisation du Traité de l'Atlantique Nord (OTAN).

1.2.1 Niveau belge

La Défense est une organisation se tournant graduellement vers la technologie [60]. Au travers de la numérisation de la Défense, l'environnement cybernétique devient une dimension fondamentale. Dans cet environnement, la Défense va déployer des capacités cybernétiques défensives, offensives et de renseignements. À l'heure actuelle, la Défense ne possède pas de capacités offensives, mais celle-ci possède bien un Cyber Security Operations Center (CSOC).

> *"Une cybercapacité militaire contribue à la défense des systèmes d'armes et de communications cruciaux pour la Défense, mais doit également pouvoir fournir une contribution à des actions offensives au sein de l'engagement expéditionnaire."*
>
> La vision stratégique pour la Défense (Juin 2016) [60]

Dans le plan stratégique de la Défense, il est prévu de mettre en place, d'ici 2030, une dimension capacitaire *Renseignement-Cyber-Influence* qui sera composée de 199 personnes (ainsi que quelques réserves). En Mars 2017, la Défense recrutait encore 11 spécialistes en cybersécurité [34]. Pour soutenir ce développement, un investissement majeur en matériel de 10,2 milliards € (2015) sera effectué entre 2016 et 2030.

Au niveau fédéral, depuis 2009, une Computer Emergency Response Team (CERT) a été mise en place : CERT.be [7]. Depuis son instauration, le nombre d'incidents rapportés et gérés par CERT.be ne cesse d'augmenter [8]. Depuis le 1er Janvier 2017, le CERT.be est coordonné par le Center for Cyber Security Belgium (CCB). La Défense contribue aussi à la sécurité intérieure avec ses actions coordonnées par le CCB.

Le CCB sert de niche d'informations et contribue également à *la gestion et la réponse aux incidents en cas d'incidents ou crises d'ampleur nationale auprès d'infrastructures critiques ou de services essentiels* [12]. Ce centre se trouve sous l'autorité directe du Premier Ministre et concentre son action sur trois types de clients [13]. Le type de client *At Home* cible les mesures de protections de la population avec, par exemple, la campagne *Take Back The Internet*. Le type *At Work* a pour objectif d'aider les petites et moyennes entreprises à prendre leur sécurité informatique en main en mettant à leur disposition des cours en ligne. Finalement, le type *Vital Sector* met en place les moyens nécessaires pour protéger les secteurs suivants : l'énergie, la mobilité, les télécommunications, le secteur financier, l'eau potable, la santé publique et le gouvernement [12].

1.2.2 Niveau européen et de l'OTAN

Au niveau européen, un CERT visant à coopérer avec les CERT nationaux a été mis en place en 2012. Son objectif est de pouvoir répondre aux incidents liés à la sécurité de l'information et aux cybermenaces [20]. Un projet a également été lancé par l'European Defence Agency (EDA) et a pour but de développer des méthodes permettant de détecter plus rapidement les infections d'APT. En effet, bien trop d'infections sont découvertes trop tard, voire pas du tout [19]. Les recherches menées à l'École Royale Militaire se cadrent dans cette initiative et des détails peuvent être retrouvés à la section 1.4.

Au niveau de l'OTAN, un centre d'excellence a été mis en place en 2008 : le NATO Cooperative Cyber Defence Centre of Excellence (CCDCOE). Ce centre est dédié à la cyberdéfense et est

consacré à la recherche et l'entraînement. Il est accrédité OTAN et mélange les expertises militaires, industrielles et gouvernementales pour apporter un regard multiple sur la cyberdéfense [42] [49]. Une NATO Computer Incident Response Capability (NCIRC) a également été créée et est chargée de prévenir, détecter et répondre aux incidents sur les réseaux de l'OTAN, ainsi que de rétablir une situation normale après un incident [41] [49]. La capacité opérationnelle totale du centre a été atteinte en 2014.

1.3 Notions clés

Cette section est consacrée à la mise au clair de certains concepts largement utilisés dans ce travail. L'objectif est de donner au lecteur la connaissance suffisante nécessaire pour cerner les concepts manipulés. Quatre notions sont détaillées : la notion d'Advanced Persistent Threat, la notion de serveur proxy, la notion de graphe et, finalement, la notion de nom de domaine.

1.3.1 Advanced Persistent Threat

Dans cette section, le lecteur acquiert les connaissances de base liées aux APT. L'objectif n'est pas d'entrer dans les détails de toutes les méthodes pouvant être utilisées pour infecter une machine, dissimuler l'infection ou encore maintenir un canal de communication avec un C2. L'objectif est bien de donner au lecteur une connaissance de base suffisante sur ce que représente ce type d'infection et lui permettre de comprendre clairement ce que ce travail cible comme menace.

La section est divisée en trois parties. La première décrit de manière générale ce qu'est une APT, quelles sont les particularités d'une APT et pourquoi avoir recours à une APT. Une seconde partie détaille la mise en oeuvre d'un canal de communication avec un C2. Finalement, la troisième et dernière partie donne une brève introduction des deux méthodes pouvant être utilisées pour détecter une infection.

Description

Le terme *Advanced Persistent Threat* (APT) est un terme bien choisi. En effet, chacun de ces trois mots est porteur d'un sens clair qu'il est intéressant de détailler, car il permet de comprendre ce que représente ce type de menace [22].

Advanced L'APT constitue une menace sophistiquée, organisée et engagée [16]. Ce type d'attaque a la capacité de compromettre un réseau sécurisé et de maintenir l'accès à un ensemble d'informations sensibles contenues dans ce réseau, tout en restant dissimulé. Cet accès peut être maintenu plusieurs mois (voire plusieurs années) avant d'être détecté.

Persistent L'APT constitue une menace persistante dans le temps, c'est-à-dire qu'il est difficile de se protéger de manière constante et permanente contre ce type d'attaque [17]. De plus, une fois infecté, il est très difficile d'éradiquer complètement l'infection. Vu les techniques sophistiquées utilisées, il est impossible d'utiliser les techniques de détection classique, car celles-ci se révèlent inefficaces. Après la phase d'infection initiale, une phase d'attente relativement longue peut avoir lieu avant la prochaine action de ce code malicieux.

Threat L'APT constitue une menace, car les auteurs de ce type de code ont à la fois les capacités et l'intention d'accéder à un réseau sensible.

Les méthodes d'infections sont multiples et une des techniques les plus performantes pour rester dissimulé lors de la phase d'intrusion est d'utiliser des vulnérabilités *zero-day* [55]. Ce type de vulnérabilités a la particularité d'être inconnu du vendeur du logiciel vulnérable. L'attaquant

exploite donc une faille avant même que le vendeur ait conscience de la faille et ait le temps de développer une mise à jour permettant la correction de la faille exploitée. Les méthodes de détection conventionnelles sont donc incapables de détecter l'attaque, car celle-ci est inconnue [23]. La manipulation de ce type de faille nécessite un très haut niveau de compétence. La recherche de failles *zero-day* ainsi que l'élaboration d'une méthode d'exploitation de cette faille sont très longues. L'achat de ce type de failles sur les marchés noirs en ligne est donc très coûteux. Ceci explique en partie pourquoi les APT ne sont pas utilisées par de simples hackers.

D'autres techniques telles que le *social engineering* permettent également d'obtenir un accès au réseau. Cette méthode exploite la vulnérabilité que constitue la nature humaine en ciblant des profils spécifiques avec des accroches qui leur sont adaptées. Ce type d'attaque nécessite une préparation méticuleuse.

Une APT est une attaque très ciblée et organisée. La mise en oeuvre de codes inédits et rédigés uniquement pour l'attaque en question est une principale caractéristique des APT. Une longue phase de préparation est nécessaire pour cerner au mieux les objectifs de l'attaque, les vulnérabilités de la victime et ses systèmes de défense. Les victimes de ce type d'attaque sont de grosses organisations telles que des organisations gouvernementales ou encore des multinationales. Cette problématique n'est donc pas exclusivement militaire, bien que les réseaux militaires et fédéraux constituent une des cibles principales.

Ces attaques établissent une présence continue et inaperçue dans un réseau sécurisé. Ceci permet de nourrir des objectifs divers, par exemple [16] :

- la volonté de trouver des gains financiers,
- la volonté de prendre l'avantage dans le cadre d'une compétitivité entre deux organisations,
- la volonté de récolter des informations sensibles et des documents confidentiels,
- la volonté de voler et d'exploiter des propriétés intellectuelles,
- la volonté de créer des dégâts moraux ou un scandale.

Command and Control center

L'APT s'avère donc être un élément infiltré dans un réseau sécurisé. Afin d'interagir avec les machines compromises ou d'exfiltrer de l'information, un canal de communication doit être établi avec un C2. Ce canal de communication doit être transporté par un protocole autorisé par les systèmes de sécurité de la victime. Plusieurs architectures typiques existent. Celles-ci peuvent être divisées en deux catégories : la structure centralisée et la structure décentralisée [23].

L'objectif n'est pas de détailler le fonctionnement de chacune des structures ainsi que les techniques utilisées pour masquer les données échangées entre le C2 et les machines infectées. Cette introduction permet au lecteur d'avoir une idée du large panel de possibilités qui s'offre à l'attaquant pour commander une APT.

Une structure centralisée est une structure où seul un nombre restreint de serveurs est utilisé pour coordonner les communications C2. Cette structure a le gros avantage d'être très facilement contrôlée. Celle-ci a toutefois le désavantage d'être très sensible aux attaques stratégiques dirigées sur le C2. De plus, sa capacité d'adaptation à de plus larges infections s'avère limitée. Une structure décentralisée permet, elle, une plus grande adaptation, une plus grande tolérance aux attaques, une plus grande disponibilité et plus de redondance. Il n'y a pas de noeud unique et il est nécessaire d'attaquer un plus large ensemble de noeuds pour désactiver les capacités de C2.

Différentes méthodes peuvent ensuite être utilisées pour masquer le trafic de données et le(s) serveur(s) utilisé(s) comme C2. Des techniques comme le chiffrement des données, l'utilisation de réseaux d'anonymisation (par exemple, le réseau TOR, The Onion Router) ou encore l'utilisation des médias sociaux sont utilisées. Mais d'autres stratégies sont également utilisées comme, par

exemple, la dissimulation des informations dans des pages web ou dans des images a priori anodines.

Finalement, la méthode utilisée pour établir des connexions avec le C2 peut varier. Deux types d'APT sont distinguées, à savoir les APT périodiques et les APT basées sur le flux de données de l'utilisateur infecté. Les APT périodiques se connectent périodiquement à leur C2, et ce, indépendamment de l'activité de l'utilisateur. Cette méthode est relativement peu efficace en matière de dissimulation, car des requêtes sont effectuées lorsqu'aucune autre ne l'est (Fig.1.1a). Une deuxième méthode consiste à établir des connexions vers le C2 uniquement lorsque l'utilisateur effectue lui aussi des requêtes. Ceci assure une plus grande dissimulation des communications car les requêtes sont dissimulées parmi d'autres requêtes (Fig.1.1b).

Détection

Deux grandes catégories de détections peuvent être distinguées [23]. La première méthode consiste à chercher des indicateurs connus attestant de l'infection d'une machine (par exemple, la connexion à un certain domaine). Ce type de détection est limité, car cela n'implique la détection que d'infections connues. Une deuxième méthode consiste à chercher des anomalies dans le trafic d'un réseau. Ces anomalies sont détectées comme une déviation par rapport à un certain modèle.

Cette deuxième approche est plus puissante, car elle permet la détection d'infections a priori inconnues. Toutefois, cette deuxième approche nécessite la capacité de modéliser ce qui est défini comme étant du trafic normal et la capacité de différencier ce modèle de trafic normal du modèle d'un trafic infecté. L'objectif étant d'obtenir une détection maximale, tout en minimisant le nombre de fausses alarmes.

1.3.2 Serveur Proxy

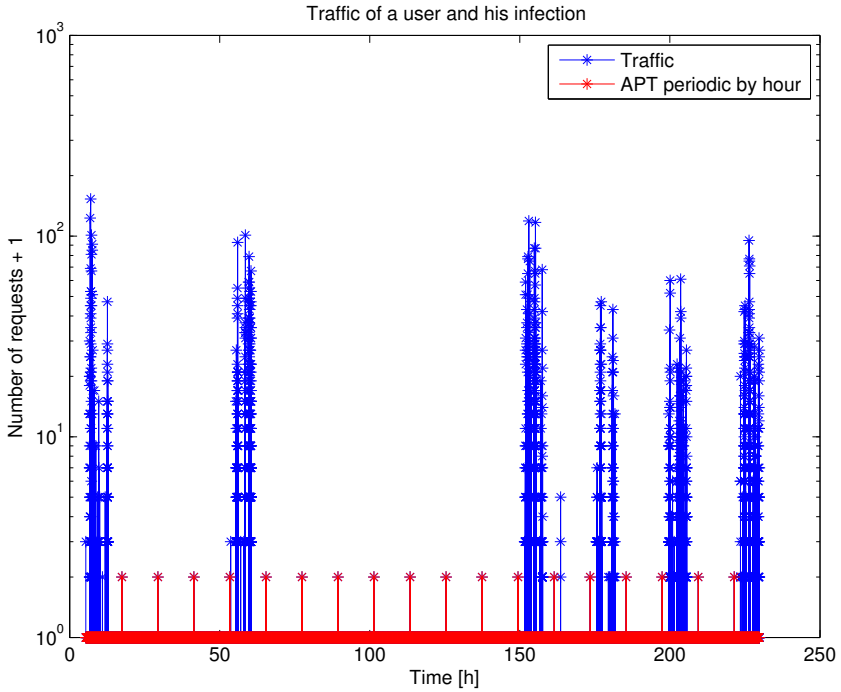
Typiquement, dans les topologies de réseaux d'entreprises, un serveur proxy est utilisé. Ce type de serveur se retrouve généralement dans la DeMilitarized Zone (DMZ) et a pour rôle de servir à la fois de point de passage obligé et d'intermédiaire entre les utilisateurs et internet. Le terme *proxy* sera abondamment utilisé pour se référer à un serveur proxy HTTP (HyperText Transfer Protocol) servant d'application gateway (L7 du modèle OSI, Open Systems Interconnection [1]) [37].

Lorsqu'un utilisateur souhaite naviguer sur une page web (Fig.1.2), il envoie ses requêtes au proxy (**1.**). Ce dernier s'occupe d'établir la connexion avec le serveur distant et de demander la page (**2.**). Une fois celle-ci obtenue (**3.**), le proxy la transfère à l'utilisateur (**4.**). Le proxy travaille donc au niveau des requêtes individuelles et sert d'intermédiaire entre le client et le serveur distant.

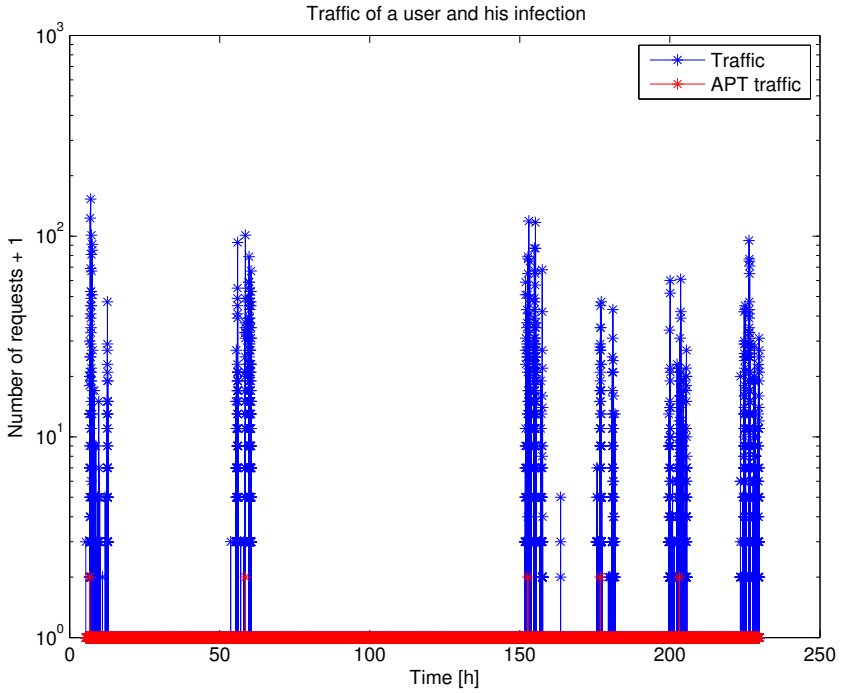
Il y a plusieurs intérêts à utiliser ce type d'appareil. Un proxy permet de bloquer l'accès à certaines URL (Uniform Resource Locator) ou de les rediriger vers une autre URL. Le proxy peut également être utilisé comme cache. Lorsqu'une page est demandée, celui-ci peut alors délivrer la page à partir de sa cache locale au lieu de devoir la demander au serveur distant. Finalement, le proxy permet également de filtrer certaines données comme, par exemple, des virus connus.

L'ensemble des requêtes effectuées par un utilisateur ainsi que les actions prises par le proxy sont écrites dans un fichier log. Ce fichier log contient donc de précieuses informations pouvant servir à l'identification d'infection. Le détail des données enregistrées est donné à la section 4.2.1.

1. Le lecteur souhaitant avoir plus d'informations au sujet du modèle OSI trouvera de plus amples informations à l'adresse suivante : `https://en.wikipedia.org/wiki/OSI_model`.



(a) Trafic d'un utilisateur infecté par une APT périodique



(b) Trafic d'un utilisateur infecté par une APT basée sur le flux de données

FIGURE 1.1 – Représentation graphique de l'intensité du trafic d'un utilisateur, sur 10 jours

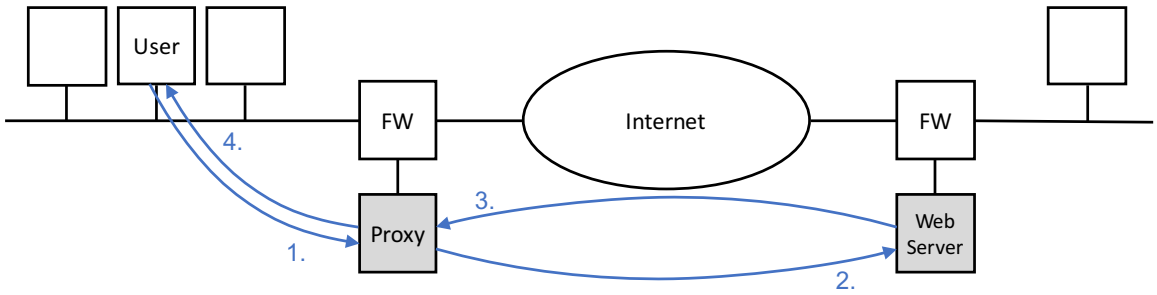


FIGURE 1.2 – Schéma d'un réseau muni d'un proxy et trajectoire suivie par une requête [35]

1.3.3 Graphes

Conceptuellement, un *graphe* est composé d'un ensemble de *noeuds* et d'un ensemble de *similarités* reliant ces noeuds [52]. Mais, une définition plus précise peut être donnée. Un *graphe* est défini comme composé d'un ensemble non vide et fini de *noeuds* et d'un ensemble fini de paires non ordonnées et distinctes de noeuds, appelées *similarités* [62] (Fig.1.3a).

Il est nécessaire de définir certains types de graphes. Un *graphe directif* est un graphe pour lequel chaque similarité a une orientation [5] (Fig.1.3b). Un *graphe complet* est, quant à lui, un graphe contenant toutes les similarités possibles entre tous les noeuds possibles [52] (Fig.1.3c). Un *graphe k-régulier* est un graphe dans lequel chaque noeud possède k similarités définies avec d'autres noeuds [5]. Finalement, un *k-NN graphe* est un graphe k-régulier où les k noeuds voisins sont les voisins ayant la plus grande valeur de similarité avec le noeud étudié. Dans ce travail, l'ensemble des graphes utilisés sont des graphes directifs. Certains graphes sont des k-NN graphes et d'autres pas.

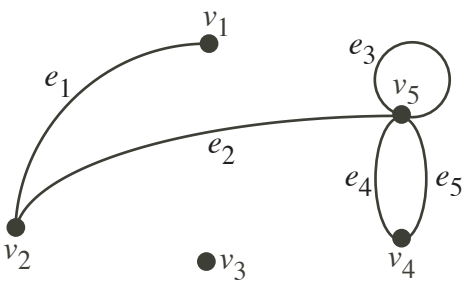
Le terme de *pruning* sera utilisé pour désigner l'action qui consiste à supprimer toutes les similarités dont la valeur est inférieure à un certain seuil, le *seuil de pruning*. Le terme de *clustering* sera lui utilisé pour désigner l'action qui consiste à chercher des structures cachées dans un graphe [3]. Le problème du clustering peut être décomposé en trois phases. D'abord, il est nécessaire de trouver la similarité entre les noeuds du graphe. Ensuite, il faut grouper les noeuds, en se basant sur leurs similarités. Finalement, une évaluation du clustering est effectuée. La création de clusters n'est pas étudiée dans le travail car la librairie utilisée pour gérer les graphes (`info.debatty.java.graphs` [14]) implémente déjà cette fonctionnalité.

1.3.4 Nom de domaine

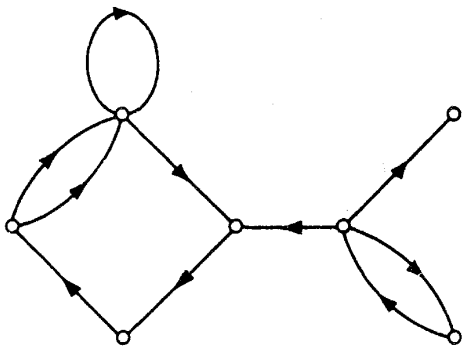
Les noms de domaine ont une structure en arbre (Fig.1.4) [38]. Chacun des noeuds de cet arbre est nommé *label* (ex. : `google`). Deux labels frères (ex. : `www` et `support`) doivent être différents, alors que cette condition n'est pas d'application pour deux labels quelconques. Le label *null* est réservé à la racine de l'arbre (`root`).

Le *nom d'un domaine* peut être défini comme une liste de labels séparés par des ".", partant du noeud considéré et allant jusqu'à la racine (ex. : `support.google.com`). De gauche à droite, les noeuds retrouvés vont donc des noeuds les plus éloignés aux noeuds les plus proches de la racine. Il est normalement nécessaire d'écrire un nom de domaine en terminant avec un point. Ceci indique que le dernier domaine est le domaine *null*, donc la racine. Toutefois, il est communément accepté de ne pas écrire le point final. Pour exemple, `support.google.com` sera communément utilisé pour signifier `support.google.com.` .

Pour terminer, la notion de *domaine de premier niveau* est introduite (ex. : `com`). Celle-ci désigne le label le plus proche de la racine et est nommée *Top Level Domain* (TLD).

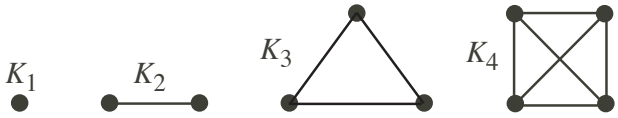


(a) Graphe avec $v_{i\in[1,5]}$, l'ensemble de noeuds et $e_{i\in[1,5]}$, l'ensemble de similarités



(b) Graphe directif

.



(c) Graphes complets

FIGURE 1.3 – Illustrations de différents graphes ; [52] pour Fig.1.3a et Fig.1.3c, [5] pour Fig.1.3b

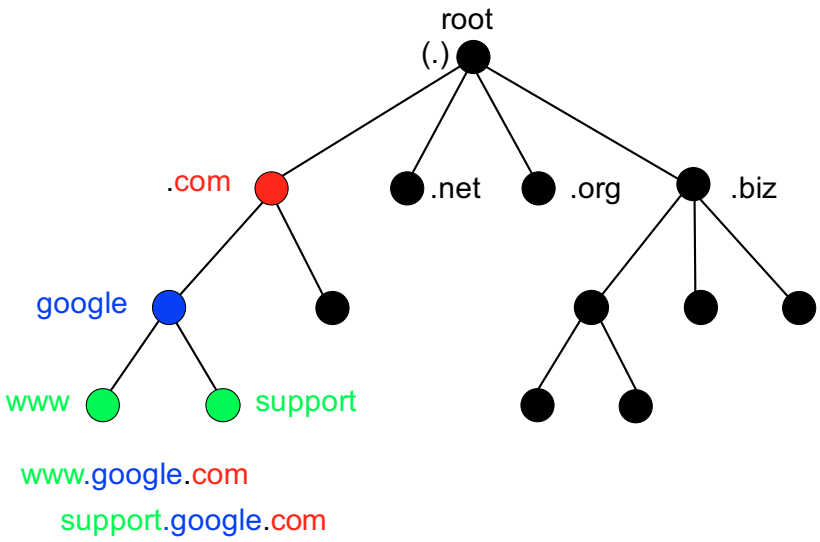


FIGURE 1.4 – Extrait de la structure en arbre des noms de domaine [26]

1.4 Problématique

Dans ce contexte de menaces croissantes liées à l'essor de la technologie de l'information, les APT sont identifiées comme faisant partie des attaques les plus sophistiquées qui soient. Il est nécessaire de mettre en oeuvre des moyens performants permettant la lutte contre ce type de menaces ciblées.

La cellule de recherche en cybersécurité (Research Unit Cyber Defence, RUCD) de l'École Royale Militaire fait des recherches actives sur le développement d'un Intrusion Detection System (IDS) combinant plusieurs approches basées sur des techniques d'agrégation de données (plutôt que sur des signatures) [36]. Ce système a pour intérêt d'assurer une grande probabilité de détection tout en conservant une faible probabilité de fausses alarmes.

L'objectif du projet Military multi-Agent System For APT Detection (MASFAD) mené par la RUCD est de concevoir un système permettant la détection d'APT. Ce projet se cadre dans les recherches effectuées par l'EDA. Cette détection se base sur des caractéristiques typiques d'APT. Des agents ciblant chacun une caractéristique spécifique vont être implémentés. Chacun d'eux produira des résultats attirant l'attention sur des événements suspects. Ces résultats seront ensuite agrégés par le système de manière à pouvoir travailler en étroite collaboration avec un expert responsable de l'analyse de ces événements suspects. Le système est conçu pour accepter l'ajout de nouveaux agents en cas de découverte de nouvelles caractéristiques.

L'existence d'un canal de communication entre l'APT et son C2 a été détaillée précédemment (Section 1.3.1). Ce travail exploite ce phénomène et base sa technique de détection sur cette caractéristique fondamentale. L'hypothèse est faite que les domaines utilisés par ces APT sont des domaines compromis ne participant pas au trafic normal des utilisateurs et que les C2 utilisés sont de structure centralisée. L'approche consiste à modéliser le trafic contenu dans un log de proxy au moyen d'un graphe. Cette modélisation doit être telle que le domaine utilisé comme C2 soit une anomalie dans ce graphe.

La plus-value de l'analyse humaine est intégrée dans la problématique. Pour ce faire, la solution construite doit être suffisamment flexible pour être interactive et permettre à un(e) analyste d'étudier le log d'un proxy sous une forme non conventionnelle (les graphes), au moyen d'un algorithme dont les paramètres lui sont accessibles.

En résumé, cet algorithme doit donc être conçu pour permettre de modéliser un log de proxy par un graphe. Ce graphe doit exploiter l'existence d'un canal de communication centralisé entre un C2 et une APT et doit faire ressortir le domaine C2 comme une anomalie du graphe. L'algorithme doit être paramétrisable et doit permettre à un(e) analyste une étude interactive du log.

1.5 Travail précédent

Ce travail ne constitue pas le premier travail de la RUCD sur la question. Il fait suite au travail du Lt Mattijs HERTSENS [28]. Les résultats obtenus lors de ce travail ont été suffisamment concluants que pour justifier un effort supplémentaire dans le domaine. De plus, la problématique reste toujours d'actualité.

Dans ce travail, trois boucles de développement ont été effectuées. La première étudie des données simulées, la seconde des données réelles et la troisième des données réelles sur une plus longue durée. La première boucle sert à développer l'algorithme, les deux suivantes à l'améliorer. Cet algorithme est relativement empirique et ses paramètres n'ont pas de signification physique particulière. Ce travail a néanmoins permis de montrer avec succès les capacités de détection d'APT que peuvent fournir les graphes.

Il est proposé d'améliorer deux points de ce travail. Premièrement, l'objectif est de parvenir à concevoir des paramètres ayant plus de sens physique pour l'analyste. Deuxièmement, il est souhaité d'augmenter l'interactivité possible entre l'analyste et l'algorithme. Ceci sera obtenu par création d'une interface graphique et par la diminution du temps de calcul nécessaire au calcul de nouveaux résultats, après la modification d'un paramètre de l'algorithme.

1.6 Structure

Le chapitre 1 fait office d'introduction au mémoire. Le chapitre 2 se consacre au phénomène physique qui devra être ciblé par l'algorithme. Il donne donc les détails du modèle d'un trafic considéré comme normal et d'un trafic comportant une APT. Le graphe modélisant ces trafics est ensuite décrit. Le trafic réel ne correspond pas toujours à la description physique donnée et certaines déviations sont présentées. Des méthodes permettant l'atténuation des effets de ces déviations sont également proposées.

Le chapitre suivant, le chapitre 3, entre dans les détails de la conception de l'algorithme qui implémente le modèle du chapitre précédent. Il détaille les trois parties de l'algorithme développé et introduit l'interface conçue pour l'utilisateur. Pour chacune des parties de l'algorithme, une introduction, un détail du fonctionnement et une explication de l'utilisation sont donnés. Une dernière partie est consacrée à la vérification de la concordance entre le phénomène physique décrit au chapitre 2 et l'algorithme décrit dans ce chapitre-ci.

Une importance toute particulière est accordée à la continuation du travail. Le chapitre 4 s'y consacre entièrement en décrivant de manière relativement détaillée les points clés du développement de l'algorithme présenté au chapitre précédent. La représentation de certaines données y est décrite. Ensuite, différents outils écrits durant le travail sont ici présentés. Ceux-ci sont répartis en deux catégories : l'aide au développement et l'aide à l'analyse.

Le chapitre 5 étudie les paramètres mis à la disposition de l'analyste dans l'algorithme développé. L'objectif est de parvenir à conseiller au mieux l'analyste en ce qui concerne l'utilisation de ces paramètres. Les deux premières parties du chapitre se consacrent à la présentation des données utilisées et à la simulation de leur infection. Ensuite, une étude exhaustive des paramètres est effectuée. Puis, pour valider la combinaison de paramètres choisis, différents nouveaux cas sont étudiés et les résultats de l'algorithme évalués. Une brève explication sur les domaines isolés par l'algorithme est ensuite fournie. Finalement, un résumé des résultats obtenus dans le chapitre est donné sous forme de conseils à l'analyste.

Le dernier chapitre, le chapitre 6, fait office de conclusion de ce mémoire. Ce chapitre détaille les différentes améliorations possibles de la solution conçue lors de ce travail. Il décrit également un bug connu existant encore toujours lors de la finalisation du travail. Il termine finalement ce mémoire par une conclusion générale.

2. Modélisation

2.1 Introduction

Ce chapitre introduit, dans sa première section, le phénomène physique qui va être ciblé par l'algorithme. Le trafic normal, ainsi que celui lié à une APT, y sont décrits. La section suivante présente le graphe qui découle directement du phénomène physique décrit à la section précédente. Le C2 d'une APT est modélisé comme une anomalie dans ce graphe et ses caractéristiques sont présentées. Dans la dernière section, certains effets liés au trafic réel sont discutés. Ceux-ci impactent directement la modélisation des sites en question. Pour chacun des effets, une approche permettant d'atténuer cet effet est décrite.

2.2 Phénomène physique

Lorsqu'un utilisateur choisit de naviguer sur une page web, il a la possibilité d'y accéder par différents moyens. L'utilisation d'un bookmark, l'entrée directe de l'adresse dans la barre de navigation ou encore l'utilisation d'un lien obtenu dans un moteur de recherche sont les méthodes les plus courantes. Pour composer complètement une page, d'autres URL sont chargées. Ces URL peuvent donc être décrites comme des enfants de la page initiale en question. Ces URL enfants appartiennent à différents domaines. Il y a une relation claire entre la page initialement demandée et les URL enfants. Les URL enfants ne sont pas accédées si la page parente n'est pas chargée. Chaque fois qu'une page est chargée, le lien entre les domaines parents et enfants doit être renforcé dans le modèle.

L'idée est de modéliser le log d'un proxy comme un ensemble de pages web. Chacune de ces pages web a été chargée aux moyens d'une succession d'accès à une URL parente et des URL enfants (et donc, à des domaines parents et enfants). Dans les logs de proxy considérés, de nombreux utilisateurs consultent diverses pages web ayant chacune plusieurs URL enfants (et donc domaines enfants). L'approche choisie est de modéliser, dans un premier temps, l'ensemble des sites consultés pour un utilisateur donné. Dans un deuxième temps, sur base des résultats obtenus pour chacun des utilisateurs individuellement, une modélisation globale du log de proxy est donnée.

L'objectif est de parvenir à détecter les connexions d'une APT à son C2. L'APT est forcément modélisée comme un domaine enfant d'un site consulté par l'utilisateur, et ce, à chaque connexion au C2. Toutefois, cette APT est enfant de nombreux sites différents et n'est que peu liée à ses différents parents (car peu de répétition de ce lien dans le log). Ce dernier critère est fondamental car il permet de distinguer les APT de certains autres types de domaines chargés sur les pages web (ex. : `google-analytics.com`). Ces domaines sont également liés à de nombreux parents, mais la récurrence de leur apparition comme domaine enfant assure une intensité plus élevée de leur lien. L'APT est, quant à elle, modélisée comme un enfant illégitime, occasionnel et faiblement lié à de nombreuses pages web différentes (et donc domaines différents).

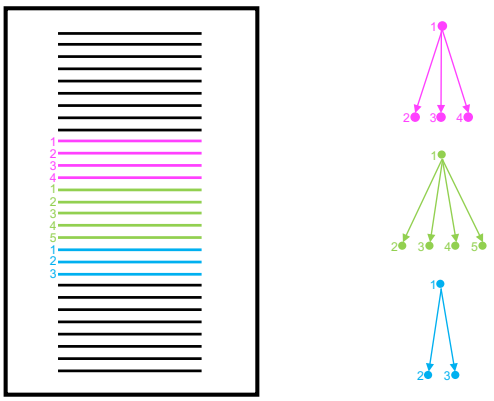


FIGURE 2.1 – Modélistation d'un log de proxy comme un ensemble de pages web, chacune composée d'un domaine parent (1) et de domaines enfants (2 à 5) (Impression artistique)

2.3 Description du graphe

L'objectif est de cibler les domaines servant de C2 à une APT. Il semble donc normal de s'intéresser à des domaines pour construire un graphe. C'est l'interdépendance entre les différents domaines présents dans le log d'un proxy qui est présentée sous forme de graphe. Les noeuds de ce graphe sont des domaines et les similarités entre ces noeuds expriment l'intensité de la relation parent/enfant entre deux domaines. Les graphes considérés sont directifs et la similarité va du domaine parent au domaine enfant. Vu que l'APT n'est théoriquement l'enfant d'aucun parent, elle devrait idéalement ne pas avoir de liens.

L'analyse du log d'un proxy permet d'identifier plusieurs utilisateurs. Chaque utilisateur a consulté de nombreux domaines (parents et enfants) en naviguant de domaine parent en domaine parent. Un graphe peut être construit pour chaque utilisateur. Celui-ci devrait idéalement représenter un ensemble de structures, chacune de ces structures représentant un site (Fig.2.1). Chaque lien entre un domaine parent et un domaine enfant est renforcé à chaque apparition de ce lien dans le log. Afin de modéliser le fichier log en entier, il est intéressant de fusionner les graphes individuels. Ce graphe fusionné est lui aussi un ensemble de structures, autrement dit, un ensemble de sites. Dans le cas où un lien entre deux domaines est retrouvé chez plusieurs utilisateurs, l'intensité de ce lien est alors renforcée dans le graphe fusionné. La fusion des graphes individuels permet donc d'améliorer la modélisation des sites.

L'APT est une anomalie dans ces graphes (Fig.2.2). En effet, le comportement recherché est un enfant illégitime, occasionnel et donc faiblement lié à de nombreux parents. Dans le graphe de l'utilisateur infecté, cette APT est liée à de nombreux parents, mais de manière faible. Contrairement à l'APT, les autres liens ont, eux, été renforcés par répétition. Dans le graphe global, ces autres liens ont également été renforcés par la fusion des utilisateurs. Les liens vers l'APT restent, quant à eux, faibles (si l'on admet qu'il n'y a que peu d'utilisateurs qui sont infectés). Comme l'APT n'a de liens forts avec aucun parent, il est possible d'isoler celle-ci en coupant les liens faibles.

Dès lors, il est important de choisir au mieux la définition de la similarité entre deux noeuds du graphe. Une approche temporelle est relativement intuitive. La similarité entre deux domaines consultés est d'autant plus forte que ceux-ci sont proches temporellement. Une autre approche peut être de regarder les noms de domaine. Bien que chacun des noms de domaine soit différent, il n'est pas impossible que certains noms de domaine aient des labels en commun. Dans ce cas, il est clair que la similarité entre ces deux domaines est plus grande également [1]. Ces observations permettront un choix avisé des similarités entre les noeuds des graphes construits.

1. La similarité entre les domaines `static.tijd.be` et `images.tijd.be` est plus grande que celle entre `static.tijd.be` et `google.be`.

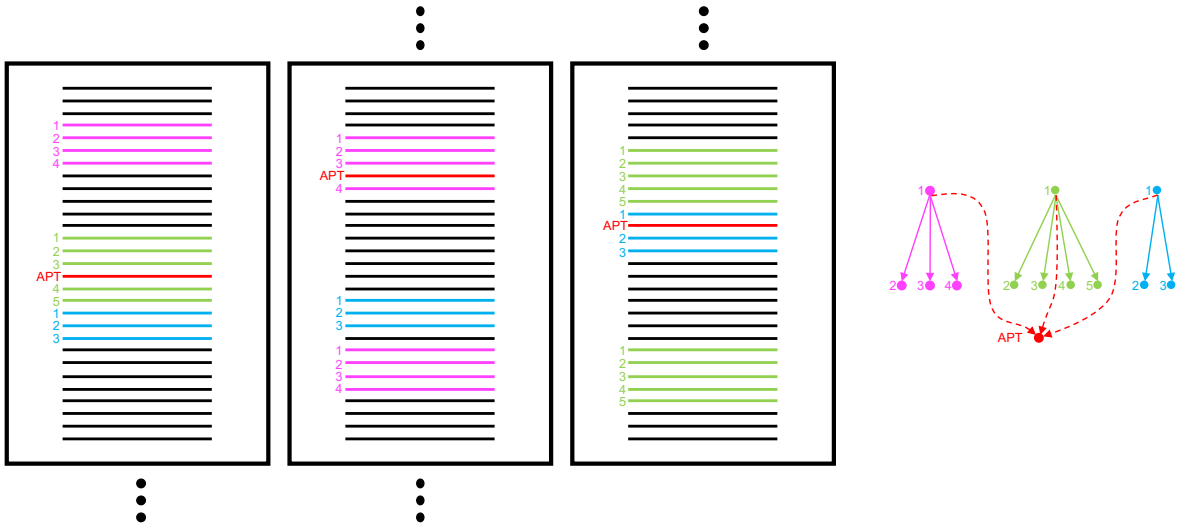


FIGURE 2.2 – Modélistation d'un log de proxy infecté par une APT (Impression artistique)

2.4 Contraintes réelles

Il est clair que la navigation d'un utilisateur ne se limite pas à la répétition du chargement complet d'une page web, suivi d'une période de consultation de la page. Cette approche est relativement limitée comme base pour la modélisation d'un log de proxy. Une approche plus complète doit tenir compte de différents effets. Cinq de ces effets sont discutés dans cette section : le chargement simultané de pages, le chargement interrompu de pages, l'activité de processus en arrière-plan, les pages dynamiques et, finalement, la problématique de la détermination des URL parentes et enfants. Pour chacun de ces effets, une approche permettant de l'atténuer est proposée.

2.4.1 Chargements simultanés

Il n'est pas rare qu'un utilisateur consulte plusieurs pages en même temps, lisant l'une pendant le chargement de l'autre. Ceci implique un mélange de plusieurs pages au sein du log de proxy (Fig.2.3). Les URL enfants de chacune des pages se mélangent, rendant la distinction de l'origine de ces enfants plus difficile. De plus, les URL enfants d'une des pages pourraient être interprétées comme des enfants illégitimes et occasionnels pour une autre page. Ce phénomène augmente donc grandement la difficulté de la création du modèle d'une page web et la détection des APT. Pour retrouver l'entièreté des enfants d'une page donnée, il est alors nécessaire de regarder suffisamment loin dans la suite du log de proxy. La répétition de la page web permettra de renforcer les liens vers les URL enfants légitimes. Les liens faibles seront eux éliminés. Il est également intéressant de considérer une autre approche que celle purement temporelle. L'analyse des noms de domaine est un choix avisé.

2.4.2 Chargement interrompu

L'utilisateur pourrait cliquer sur un lien contenu dans la page avant même que celle-ci ne soit totalement chargée. Il pourrait également manuellement interrompre le chargement de la page ou encore fermer son navigateur avant le chargement complet d'une page. Dans ces différents cas, le log ne contient pas l'entièreté des URL enfants liées à la page dont le chargement a été interrompu (Fig.2.4). Ce type de comportement a tendance à affaiblir l'intensité du lien entre certaines des URL enfants et leur URL parente. Le risque est alors de juger certaines des URL enfants comme faiblement liées à l'URL parente et de les juger suspicieuses. À nouveau, la répétition de la

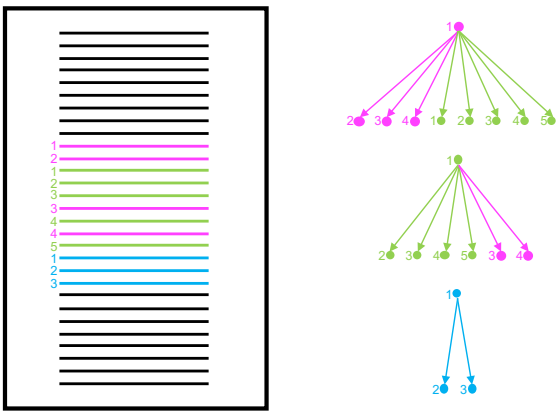


FIGURE 2.3 – Effet du chargement simultané de deux pages web (Impression artistique)

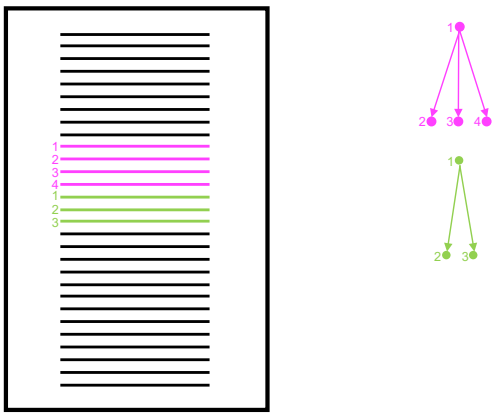


FIGURE 2.4 – Effet du chargement interrompu d'une page web (Impression artistique)

page web augmente la probabilité de capturer dans la modélisation l'ensemble des URL enfants. Pour augmenter le lien de certaines URL enfants avec leur URL parente, l'analyse des noms de domaine pourrait également être utilisée.

2.4.3 Processus en arrière-plan

Certains processus travaillent en arrière-plan et fournissent des services tels que les mises à jour automatiques. Ces services vont également générer du trafic répertorié dans le log de proxy. Toutefois, ce trafic est indépendant de l'utilisateur humain. L'utilisateur a autorisé (parfois inconsciemment) ces processus à faire ces connexions, contrairement aux APT. Ces connexions autorisées ne sont pas des URL enfants de pages visitées par l'utilisateur. Elles ont les mêmes caractéristiques qu'une APT : URL enfant occasionnelle et faiblement liée à des URL parentes. Un système de *white listing* peut être mis en place pour éviter ces fausses alarmes.

2.4.4 Pages dynamiques

De nos jours, les pages sont beaucoup plus dynamiques qu'elles ne l'étaient auparavant. Les URL enfants d'un même site peuvent donc varier au cours du temps. Au plus la page est dynamique, au plus ses URL enfants varient rapidement. Si les URL enfants varient rapidement, les liens entre URL parentes et enfants n'ont pas le temps d'être consolidés par répétition. Ceci induit un risque de juger ces liens comme étant des liens suspicieux. Ils sont non permanents et faiblement liés à l'URL parente, tout comme l'APT. Mais ils ne sont en rien illégitimes. Pour renforcer le

lien, une approche autre que celle purement temporelle doit être imaginée. L'analyse des noms de domaine pourrait répondre à cette problématique.

2.4.5 Détermination des URL parentes et enfants

Lorsqu'un log de proxy est consulté, il n'est pas possible de déterminer les URL parentes et enfants. En effet, il n'y a pas d'information qui permette d'indiquer les URL chargées volontairement par l'utilisateur et les URL chargées en conséquence d'une autre URL. En pratique, il est nécessaire de considérer chacune des URL comme parente et de calculer les liens vers tous les enfants à celle-ci. Pour des raisons évidentes, il semble clair que seul un nombre limité de liens pourra être calculé pour chacun des parents. La modélisation correcte d'un site web repose donc sur la répétition de certains liens qui s'en trouvent renforcés. Les liens faibles sont eux supprimés.

2.5 Conclusion

Ce chapitre décrit clairement le phénomène physique ciblé par l'algorithme. Grâce à la connaissance du phénomène, il a été possible de décrire les caractéristiques du graphe souhaité et les caractéristiques d'une APT dans ce graphe. Il s'agit d'un graphe de domaines, issu de la fusion des graphes individuels de chacun des utilisateurs. La similarité définie entre deux domaines reflète l'intensité du lien parent/enfant. Cette similarité se doit d'être définie de manière mixte. En effet, une approche purement temporelle n'est pas suffisante pour atténuer les effets des contraintes réelles. L'intensité de la relation entre deux domaines se doit d'être renforcée par la répétition du lien chez un même utilisateur ou entre deux utilisateurs. Le C2 d'une APT se caractérise par son isolement dans ce graphe, après suppression des liens faibles.

Finalement, certains effets ont été mentionnés. Ceux-ci permettent de prendre conscience du fait que la navigation n'est pas aussi linéaire que l'on pourrait l'espérer et que la détermination des URL parentes et enfants n'est pas si simple. Pour chacun des effets, une solution a été proposée pour limiter l'impact sur le graphe produit. Il en ressort la nécessité de répétition des sites web pour renforcer la modélisation de leur structure, la nécessité de considérer suffisamment d'URL enfants par URL parente pour capturer l'entièreté de la structure d'une page, la nécessité d'utilisation d'une autre approche que celle purement temporelle pour déterminer les URL enfants et, finalement, la nécessité de white listing.

3. Processing

3.1 Introduction

Maintenant que le modèle est connu, la conception du processing peut être entamée. L'objectif du travail est de proposer un algorithme qui soit paramétrisable et interactif. De plus, il est souhaité qu'aucune information ne soit a priori supprimée et que l'analyste soit le (la) seul(e) acteur (actrice) responsable de la suppression d'informations avant le rendu final. Le design même du processing a un impact crucial sur ces objectifs. Afin d'assurer l'interactivité entre l'interface utilisateur et l'analyste, il est nécessaire de précalculer un maximum de données. De plus, pour assurer à l'analyste l'accès à un maximum de paramètres, il est aussi nécessaire de limiter le nombre de choix effectués a priori.

Ce chapitre commence par introduire la structure de l'algorithme et les raisons pour lesquelles l'algorithme a été divisé en trois segments : le Core, le Batch Processor et le Server. Le Core est ensuite introduit. Celui-ci définit tant des objets Java que des méthodes qui sont utilisés dans les autres segments. Le chapitre donne après les détails du fonctionnement, la séquence de processing et l'utilisation du Batch Processor et du Server. Puis, l'interface utilisateur conçue est présentée, ainsi que les paramètres et les outils d'analyse qu'elle propose. Finalement, le processing est révisé afin de s'assurer de la concordance de celui-ci avec le phénomène physique présenté au chapitre précédent.

3.2 Structure de l'algorithme

Il est intéressant de détailler les raisons pour lesquelles l'algorithme proposé a une architecture en trois segments. Cette conception tire profit de l'expérience du travail du Lt Mattijs HERTSENS [28]. Celui-ci proposait un algorithme nécessitant un long temps d'attente, lors de la modification d'un paramètre, pour obtenir la mise à jour des résultats. Or, l'objectif est de concevoir un algorithme permettant de faire varier des paramètres de manière interactive. La partie la plus lourde du point de vue de la charge de calcul est la génération des graphes. Un effort particulier est fourni à ce niveau.

Idéalement, un log de proxy serait modélisé par un graphe complet. Ce graphe contiendrait toutes les similarités et la modélisation se ferait en tenant compte de l'entièreté des contributions pour renforcer, par répétition, la similarité entre certains noeuds.

Toutefois, travailler avec un graphe complet a un impact certain sur la quantité d'informations à stocker en mémoire. Pour un graphe complet, il y a $n-1$ similarités à stocker par noeud. Au total, c'est donc $n(n-1)$ similarités à stocker et $n(n-1)/2$ à calculer [1].

Dans la librairie `info.debatty.java.graphs` [14], les graphes sont représentés comme un ensemble de noeuds. À chaque noeud est associé un ensemble de liens vers d'autres noeuds. L'utilisation des références Java permet de stocker ces liens de manière efficace. Les liens ne sont

1. Les similarités sont symétriques.

que des références vers d'autres noeuds (c'est-à-dire 64 bits au maximum sur 64-bit JVM, Java Virtual Machine [47]), auxquelles sont associées une valeur de similarité (c'est-à-dire un double, donc 64 bits en Java [48]). Un lien demande donc 128 bits, soit 16 bytes, pour être stocké.

Si un graphe de 100 000 noeuds est considéré ($n = 100\,000$), il est nécessaire d'établir $n - 1$ liens ($n - 1 = 99\,999$) par noeud pour créer un graphe complet. Or, beaucoup de ces liens ont des valeurs de similarité très faibles. De plus, rien que pour stocker les liens, il est nécessaire de stocker presque 160GB de données. À cela, il faut encore ajouter l'espace nécessaire au stockage des noeuds. Si, par contre, un k-NN graphe est utilisé, seulement les k liens les plus intenses (c'est-à-dire les k liens ayant la plus grande valeur de similarité) doivent être stockés par noeud. Dans l'exemple précédent, seulement 160MB sont alors nécessaires pour stocker les liens lorsque $k = 100$.

Il faut être prudent par rapport aux impacts de ce choix sur la qualité du modèle. Ces effets sont discutés dans la section 3.7.1. Un choix avisé de la valeur du k doit être fait pour assurer une modélisation correcte du log. Ce choix est crucial et l'analyste est guidé(e) dans ce choix à la section 5.4.1.

Pour comprendre la division de l'algorithme en trois segments, il est également intéressant d'étudier la charge que représente le calcul de k-NN graphes. Si un k-NN graphe est construit de manière naïve, une méthode par force brute est utilisée [15]. Celle-ci consiste, pour chaque noeud du graphe, à calculer toutes les similarités possibles et à sélectionner les k similarités les plus intenses. Cette méthode a donc une complexité en $O(n^2)$. Ce type d'algorithme est très lent, même lorsque le calcul est parallélisé. De plus, il est important de souligner qu'il n'y a pas qu'un k-NN graphe à calculer. Le nombre de graphes à calculer est proportionnel au nombre d'utilisateurs présents dans le log.

D'autres algorithmes ont été développés pour diminuer la charge de calcul d'un k-NN graphe. L'objectif du travail n'est pas d'étudier le fonctionnement de ces algorithmes, mais bien de les utiliser. L'algorithme *NN-Descent* est l'un d'entre eux et il a été montré expérimentalement que cet algorithme a une complexité de $O(n^{1.14})$ [18]. Il a été implémenté dans la librairie `info.debatty.java.graphs` et sera utilisé dans le cadre de ce travail.

Malgré les efforts fournis pour diminuer la charge de calcul, le temps de calcul des graphes reste très long. Ceci ne permet pas d'interactions entre l'analyste et l'algorithme. Pour s'assurer d'atteindre les objectifs fixés quant à l'interactivité, un préprocessing est utilisé.

Ce préprocessing est responsable du calcul des k-NN graphes. Le traitement de ces graphes est alors effectué dans un autre segment de l'algorithme. Ce traitement a pour objectif de créer le graphe correspondant au phénomène physique en se basant sur les données précalculées. Puisqu'une phase de préprocessing est nécessaire, celle-ci va être exploitée pour précalculer un maximum d'informations. Certaines définitions d'objets Java ou de méthodes sont communes aux deux segments. Un troisième segment est alors conçu pour servir de point de référence unique pour ces définitions.

L'algorithme est donc divisé en trois segments. Le premier, le *Core*, est responsable de la définition tant des objets Java que des méthodes qui sont utilisés dans les deux segments suivants. Le deuxième segment, le *Batch Processor*, se charge du préprocessing. Le troisième et dernier segment, le *Server*, s'occupe, lui, de la finalisation du calcul du graphe et de son traitement pour isoler les C2. À l'algorithme est associée une interface graphique. Celle-ci achève la mise en oeuvre d'un système interactif. Elle présente les résultats de l'algorithme, ainsi que les paramètres et outils d'analyse qu'elle propose.

Δt (s)	$\mu_{\Delta t}$	Δt (s)	$\mu_{\Delta t}$
∞	0,00	5	0,17
10	0,09	4	0,20
9	0,10	3	0,25
8	0,11	2	0,33
7	0,13	1	0,50
6	0,14	0	1,00

FIGURE 3.1 – Similarité temporelle pour différents Δt

3.3 Core

Le Core permet de définir tant des objets Java que des méthodes qui sont utilisés dans les deux segments suivants de l'algorithme. Cette section s'intéresse particulièrement aux similarités pouvant être utilisées dans les graphes. Actuellement, deux similarités sont implémentées et présentées dans cette section : une similarité temporelle et une similarité basée sur les noms de domaine. Ces choix ne sont en rien limitant et le travail nécessaire pour l'ajout de nouvelles similarités a volontairement été réduit au minimum. Ceci permet de faciliter l'adaptation de l'algorithme aux besoins de l'analyste. L'introduction de nouvelles similarités peut être l'objet d'un travail futur ayant pour objectif d'améliorer l'algorithme (Section 6.2.1). Afin de tirer profit des avantages de chacune des similarités, l'algorithme est conçu pour pouvoir combiner les graphes de chacune de ces similarités. Ce dernier point est détaillé dans la section sur le Batch Processor (Section 3.4).

3.3.1 Similarité temporelle

Cette similarité se calcule sur base de la différence temporelle de deux entrées dans un log, appelées requêtes. La similarité est définie comme suit :

$$\mu_{\Delta t} = \frac{1}{1 + |\Delta t|} \tag{3.1}$$

avec Δt défini comme la différence temporelle entre les horodatages (*timestamps*) respectifs des deux requêtes.

Il peut être directement remarqué que cette similarité ne fait pas de distinction entre les voisins passés et futurs. La similarité est symétrique. Pour cette similarité, il est donc préférable de parler de voisins plutôt que d'enfants. Afin de mieux s'accorder à la modélisation souhaitée, un traitement dans le Batch Processor permettra la sélection des enfants parmi ces voisins.

Le choix même de la définition de $\mu_{\Delta t}$ permet d'assurer que la similarité varie entre 0 ($\Delta t = \infty$) et 1 ($\Delta t = 0$). La similarité est donc normalisée. De plus, son comportement est parfaitement identique à ce qui est attendu : celle-ci est faible pour un grand Δt et importante pour un faible Δt. Au plus deux requêtes sont proches temporellement, au plus leur similarité temporelle $\mu_{\Delta t}$ augmente.

La résolution de la différence temporelle n'est pas définie. Celle-ci est fonction de la résolution temporelle des logs de proxy. Pour les logs étudiés, la résolution est typiquement de 1 seconde. Il est important de remarquer que la résolution temporelle du Δt implique directement que la similarité $\mu_{\Delta t}$ est quantifiée et que ses valeurs peuvent être prédites (Fig.3.1).

β	β_{tot}	μ_{Domain}
1	2	0,50
1	3	0,33
1	4	0,25
2	3	0,67
2	4	0,50
3	4	0,75

FIGURE 3.2 – Similarité basée sur les noms de domaine pour un nombre donné de labels communs

3.3.2 Similarité basée sur les noms de domaine

Cette similarité se calcule entre deux noms de domaine de requêtes. Elle est définie comme suit :

$$\mu_{Dom} = \frac{\beta}{\beta_{tot}} \tag{3.2}$$

avec β défini comme le nombre de labels (Section 1.3.4) communs entre ces deux noms de domaine, en partant du Top Level Domain (TLD), TLD exclus [2] mais identiques ; β_{tot} défini comme le maximum entre les deux nombres de labels des noms de domaine considérés, TLD exclus.

Cette similarité est née de l'observation des domaines isolés lors des tests (Section 5.9). Il a été constaté que certains domaines typiques d'un site se retrouvaient isolés, car ils n'étaient que très peu de fois appelés (ex : `cnn.com` détaché du domaine principal `edition.cnn.com`). L'utilisation de cette similarité permet d'éviter leur isolement.

La similarité est à nouveau symétrique et normalisée. Celle-ci varie de 0 (aucun label en commun) à 1 (tous les labels en commun) et augmente pour un nombre croissant de labels en commun. Les valeurs des similarités sont elles aussi quantifiées et peuvent être prédites. La discussion est limitée aux domaines de maximum quatre labels (TLD non compris) et la similarité d'un domaine avec lui-même n'est pas considérée (Fig.3.2).

3.4 Batch Processor

Le Batch Processor est utilisé pour augmenter l'interactivité avec l'analyste en précalculant un maximum de données, tout en limitant le nombre de paramètres fixés a priori. Pour comprendre le Batch Processor, il est nécessaire de se souvenir de l'objectif fixé dans le chapitre de modélisation, lors de la description du graphe final (Section 2.3). L'objectif est d'obtenir un graphe de domaines dont les similarités expriment l'intensité de la relation parent/enfant entre deux domaines. Ce graphe est issu de la fusion des graphes individuels de chacun des utilisateurs.

Cette section présente d'abord les détails de fonctionnement du Batch Processor. Ensuite, la séquence de processing est résumée. Finalement, l'utilisation du Batch Processor est expliquée.

3.4.1 Détails de fonctionnement

Différentes similarités ont été définies dans la section précédente (Section 3.3). Afin de tirer profit des avantages de chacune de ces similarités, il est proposé de calculer les graphes de chacune

2. L'inclusion du TLD crée des similarités entre des domaines qui n'ont a priori aucun lien (ex. : `tijd.be` et `rtbf.be` auraient 0,5 de similarité si le TLD était inclus. Or, il est préférable qu'ils aient 0 de similarité.).

d'entre elles et de permettre à l'analyste de choisir les proportions dans lesquelles ces différentes similarités sont fusionnées. Les paramètres de fusion ne sont donc pas connus a priori, mais les graphes peuvent déjà être calculés pour chaque similarité séparément. Le Batch Processor va donc précalculer les graphes de chacune des similarités, et ce, pour chaque utilisateur contenu dans le log. Tout comme pour la fusion des similarités, la fusion des utilisateurs ne peut pas non plus être faite a priori, car l'analyste dispose du choix des utilisateurs étudiés dans le graphe final.

k-NN Graphes

Pour des raisons évidentes d'espace mémoire et de temps de calcul (Section 3.2), il n'est pas envisageable de calculer un graphe complet pour chacune des similarités de chacun des utilisateurs. Il a été choisi de travailler avec des k-NN graphes. Dans ces graphes, la similarité est définie comme directionnelle, du nœud parent vers ses k voisins. Le paramètre k est un paramètre qui doit être fixé a priori (sa valeur par défaut est fixée à 20). La section 5.4.1 a pour objectif de conseiller au mieux l'analyste dans le choix de ce paramètre.

Graphes de requêtes et sélection des requêtes enfants

Le Batch Processor prend un fichier log comme entrée. Après parsing et séparation des requêtes par utilisateur, il est possible de calculer des k-NN graphes de requêtes pour chacune des similarités définies. Lors de la définition des similarités (Section 3.3), il a été souligné que les similarités étaient symétriques. Or, l'objectif est de modéliser un lien du parent vers l'enfant. Pour ce faire, chaque nœud va être nettoyé des voisins qui le précèdent temporellement. Ainsi, l'ensemble des voisins d'un nœud devient l'ensemble de ses enfants temporels. Bien que la relation parent/enfant ait été décrite comme faisant partie intégrante du modèle (Chapitre 2), il est laissé à l'analyste la possibilité de désactiver la sélection des enfants parmi les voisins. Ce paramètre doit être donné a priori. Une fois cette sélection faite, les graphes manipulés ne sont plus systématiquement des k-NN graphes. En effet, certains noeuds n'ont plus k voisins (mais un nombre inférieur d'enfants).

Graphes de domaines

Jusqu'à présent, le Batch Processor produit des graphes de requêtes pour chacune des similarités et chacun des utilisateurs. Afin d'obtenir des graphes de domaines, il est choisi de définir la similarité entre deux domaines comme la somme des similarités entre les requêtes appartenant à ces deux domaines. Une fois ces graphes produits, ils sont stockés sur disque. Il est intéressant de noter que, à partir de maintenant, les graphes manipulés ne sont plus des k-NN graphes, même si la sélection des enfants n'a pas été faite. En effet, après création des graphes de domaines, le nombre de voisins peut finalement être supérieur, inférieur ou égal à k. Un fichier par utilisateur est créé et est nommé : `ip.address.ser` (ex. : `192.168.2.167.ser`). Les listes des utilisateurs (`users.ser`) et des différents sous-réseaux (`subnets.ser`), ainsi que la valeur de k pour les k-NN graphes des différentes similarités (`k.ser`), sont également sauvegardées sur disque.

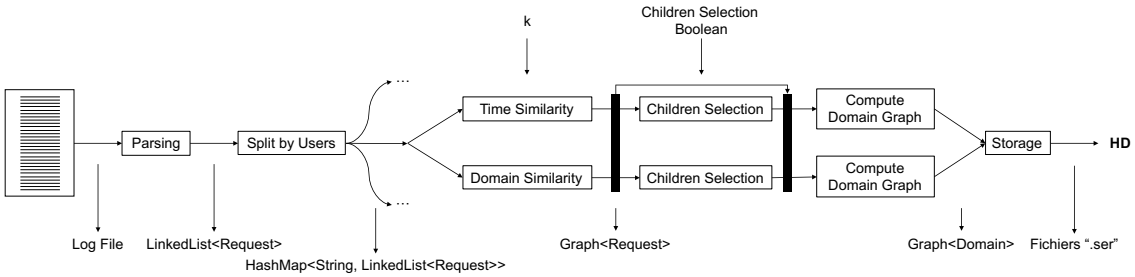


FIGURE 3.3 – Schéma du traitement effectué par le Batch Processor

```
MacBook-Pro-de-Thomas:batch Thomas$ ./analyze.sh -h
usage: java -jar batch-<version>.jar
 -c <arg>   Select only temporal children (option, default: true)
 -f <arg>   Specify format of input file (squid or json) (option, default
            : squid)
 -h         Show this help
 -i <arg>   Input log file (required)
 -k <arg>   Impose k value of k-NN graphs (option, default: 20)
 -o <arg>   Output directory for graphs (required)
 -x <arg>   Overwrite existing graphs (option, default : false)
MacBook-Pro-de-Thomas:batch Thomas$
```

FIGURE 3.4 – Utilisation du Batch Processor

3.4.2 Processing

Le processing peut être résumé par les étapes suivantes (Fig.3.3 [3]) :

1. parsing ;
2. décomposition du log par utilisateur ;
3. calcul des k-NN graphes de requêtes pour chacune des similarités, pour chacun des utilisateurs ;
4. sélection des requêtes enfants parmi les requêtes voisines (optionnel) ;
5. calcul des graphes de domaines pour chacune des similarités, pour chacun des utilisateurs ;
6. stockage des données suivantes sur disque : graphes relatifs à chacun des utilisateurs (`ip.address.ser`), `users.ser`, `subnets.ser` et `k.ser`.

3.4.3 Utilisation

Le Batch Processor prend les arguments suivants comme entrée (Fig.3.4) :

- **-i** : le fichier log à analyser ;
- **-o** : le dossier dans lequel les données seront stockées ;
- **-f** : le format du fichier log [4] (*squid* ou *json*) (optionnel, par défaut : *squid*) ;
- **-k** : la valeur de k (optionnel, par défaut : 20) ;
- **-c** : le choix concernant la sélection des enfants (optionnel, par défaut : *true*) ;
- **-x** : le choix concernant le recalcul des données déjà présentes dans le dossier de sortie (optionnel, par défaut : *false*).

Une commande typique est donnée ci-après :

```
1 ./analyze.sh -i <proxy log file> -o <graphs directory> -k 50 -f squid
```

3. Le détail de la structure des données est présenté dans la section 4.2.
4. Les formats de fichier log sont présentés à la section 4.2.1.

3.5 Server

Le Server donne accès de manière interactive aux paramètres permettant de terminer la construction du graphe. Il va également traiter le graphe pour en isoler les domaines aux comportements similaires à ce qui a été décrit comme un potentiel C2 d'une APT (Section 2.3). La conception du Server est cruciale, car ce module assure l'interactivité entre l'algorithme et l'analyste. L'objectif est de produire un classement de domaines permettant d'identifier les domaines aux comportements typés C2. Le Server exploite les données sauvegardées par le Batch Processor pour produire ces résultats.

De la même manière que cela avait été fait pour le Batch Processor, les détails de fonctionnement sont d'abord expliqués. Ensuite, le processing est résumé. Finalement, l'utilisation du Server est détaillée.

3.5.1 Détails de fonctionnement

Le Batch Processor a produit, pour chaque utilisateur, un graphe de domaines par similarité définie dans le Core. Pour rappel, ces graphes ne sont plus des k-NN graphes suite à la sélection des enfants parmi les voisins de chacun des nœuds et à la fusion des requêtes en domaines. À partir de ces graphes, il est nécessaire de fusionner les graphes de similarités pour obtenir le graphe de domaines de chacun des utilisateurs. Ensuite, il est nécessaire de fusionner entre eux le graphe de chacun des utilisateurs pour obtenir le graphe final. Pour terminer, ce graphe est traité pour faire ressortir les domaines suspicieux.

L'algorithme se veut tel que l'ensemble des données initiales soit conservé jusqu'à la fin du traitement. Les seules données manquantes sont celles écartées par l'analyste. À la fin du traitement, il doit être possible de retrouver l'ensemble des requêtes composant un domaine, et donc de remonter jusqu'aux entrées dans le log.

Fusions des similarités et des utilisateurs

Ces deux fusions ne peuvent pas être faites a priori. En effet, pour la fusion des similarités, il est nécessaire d'avoir le choix de l'analyste quant aux proportions dans lesquelles les graphes des similarités vont être fusionnés. Pour la fusion des utilisateurs, il est nécessaire d'avoir le choix de l'analyste quant aux utilisateurs qui doivent être intégrés dans l'étude.

Une somme pondérée des similarités entre deux domaines est utilisée pour fusionner les graphes des similarités. Les pondérations de cette somme sont un ensemble de paramètres que l'analyste doit déterminer. La section 5.4.2 a pour objectif de conseiller au mieux l'analyste dans le choix de ces paramètres. Concernant la fusion des utilisateurs, il est possible de choisir un utilisateur particulier ou un sous-réseau. Une somme pondérée est également utilisée, mais la pondération est fixe. Elle est équivalente pour tous les utilisateurs et vaut 1. Les similarités dans le graphe final correspondent donc à la somme des similarités de chacun des graphes individuels obtenus après fusion des similarités.

Le choix d'utiliser des sommes pondérées a été fait, car la somme pondérée est un opérateur permettant de comprendre facilement le processus de fusion. C'est un opérateur relativement facile à débugger et pour lequel il est plutôt simple de prédire les résultats. L'algorithme pourrait être modifié pour utiliser des opérateurs plus complexes comme les opérateurs Ordered Weighted Average (OWA) et Weighted Ordered Weighted Average (WOWA). Ceci fait partie des éléments pouvant être étudiés dans une suite à ce travail (Section 6.2.1).

Une fois ces deux fusions effectuées, le graphe obtenu est un graphe de domaines dont les similarités expriment une intensité de relation parent/enfant, et ce, dans diverses dimensions comme le

temps et les noms de domaine (en fonction des similarités définies). Afin d'isoler au maximum les domaines aux comportements typés C2, une succession de traitement doit encore être effectuée sur ce graphe.

Pruning

Un domaine se comportant comme un C2 est un enfant illégitime, occasionnel et donc faiblement lié à de nombreux parents. Afin de se débarrasser de ces liens faibles, le graphe va être nettoyé de tous les liens dont l'intensité est inférieure à un certain seuil. Ce seuil est défini par l'analyste. La section 5.4.3 a pour objectif de conseiller au mieux l'analyste dans le choix de ce paramètre. La valeur de seuil peut être donnée sous deux formes : une valeur absolue ou une valeur standardisée (z-score). L'utilisation d'un z-score donne l'avantage d'adapter le seuil à la distribution de similarités étudiées. Un z-score de 0,0 permet, par exemple, de couper tous les liens dont la valeur est inférieure à la moyenne. Pour rappel, le z-score est défini comme suit :

$$z = \frac{x - \mu}{\sigma} \tag{3.3}$$

avec x, le score absolu ; μ, la moyenne et σ, l'écart-type.

Clustering et Filtering

Grâce à l'étape précédente, les liens restant dans le graphe sont des liens forts entre domaines. Les domaines fortement liés entre eux font donc, selon le modèle, partie du même ensemble de pages web. Le C2, quant à lui, devrait avoir été isolé par le pruning. Afin de faire ressortir les domaines isolés, des clusters sont cherchés dans le graphe. Cette étape est dénommée *clustering*. Les clusters dont la taille est supérieure à un certain seuil sont éliminés. Cette étape est dénommée *filtering*. À nouveau, le seuil est un paramètre fourni par l'analyste et peut être donné sous deux formes : une valeur absolue ou une valeur standardisée (z-score). La section 5.4.4 a pour objectif de conseiller au mieux l'analyste dans le choix de ce paramètre.

White Listing

Comme cela avait été mentionné dans la section 2.4.3, l'utilisateur n'est pas toujours directement responsable des URL consultées par sa machine. Un processus de white listing a été proposé pour remédier à ce problème. Pour ce faire, une fois le filtering effectué, les clusters restants peuvent être nettoyés des domaines répertoriés dans une white list intégrée dans l'algorithme. Parmi ceux-ci, les domaines publicitaires peuvent être cités, mais également, les domaines connus avec certitude comme les domaines utilisés pour les mises à jour. L'analyste a le choix d'effectuer ou non ce nettoyage, et peut également fournir des domaines supplémentaires à ceux contenus dans la white list. Cette dernière fonctionnalité est dénommée *on the go* white listing.

Il est également possible pour l'analyste de supprimer les domaines qui n'apparaissent qu'un nombre très restreint de fois. Ce critère impose un nombre minimum de requêtes par utilisateur et par domaine sur la période étudiée. Ce paramètre permet de ne pas considérer comme suspicieux certains domaines isolés par leur très faible présence. Le résultat final est ainsi nettoyé de bons nombres de domaines.

Il est intéressant de noter que la liste des hosts sur laquelle se base le white listing est de la plus haute importance. Cette liste doit être fiable, au risque de ne pas afficher un domaine servant de

C2. La liste utilisée est celle construite par Steven Black [4]. Celle-ci est la consolidation de sources réputées pour le blocage de publicités. Actuellement, le white listing cible particulièrement les publicités, mais il peut être appliqué sur n'importe quel domaine reconnu avec certitude.

Initialement, l'idée était de faire le white listing juste après la fusion en domaine. Ainsi, les données traitées par le Server auraient été épurées de nombreux domaines. Ceci aurait allégé significativement la charge de calcul. Toutefois, le fait de supprimer des domaines à cette étape du processus entraînait la fragmentation des structures créées par les similarités. Il en résultait de très nombreux clusters de petite taille. Tout l'intérêt du processing réside justement dans le design de similarités de telle sorte que celles-ci modélisent la structure des sites. L'idée de supprimer des domaines avant la fin du processing a donc été abandonnée. L'option actuellement choisie est une facilitation visuelle pour l'analyste. Le processing s'effectue sur les structures créées par les similarités et l'analyste n'observe que les domaines suspicieux parmi les résultats obtenus.

Classement

Afin de faciliter le travail de l'analyste, un classement des domaines peut être déduit du graphe produit. Ce classement est basé sur trois indices : la somme des similarités des connexions parentes, la somme des similarités des connexions enfants et le nombre de requêtes faites par les utilisateurs vers ce domaine. Le classement est effectué sur base d'une somme pondérée de ces indices. Les pondérations sont fournies par l'analyste.

Après le processing effectué sur le graphe, il est attendu que le C2 soit isolé au maximum. Ceci implique un faible nombre de connexions (tant parentes qu'enfants), et donc, de faibles valeurs des indices associés. De plus, pour rester au maximum discrète, il peut être supposé que l'APT va limiter le nombre de requêtes sortantes. Ceci implique que le nombre de requêtes associées au C2 soit faible et que la valeur de l'indice associé le soit également.

Dans le classement final, les domaines les plus suspicieux ont donc un rang faible et sont proches du début de la liste. Le choix des indices servant au classement des domaines ainsi que l'utilisation d'une somme pondérée de ces indices pourraient faire l'objet d'une étude supplémentaire (Section 6.2.1). La section 5.4.6 a pour objectif de conseiller au mieux l'analyste dans le choix de ces pondérations.

3.5.2 Processing

Le processing peut être résumé par les étapes suivantes (Fig.3.5 [5]) :

1. chargement des données relatives aux utilisateurs sélectionnés par l'analyste (`ip.address.ser`) ainsi que les fichiers `users.ser`, `subnets.ser` et `k.ser` ;
2. fusion des similarités ;
3. fusion des utilisateurs ;
4. pruning ;
5. clustering ;
6. filtering ;
7. white listing (optionnel) ;
8. classement.

5. Le détail de la structure des données est donné dans la section 4.2.

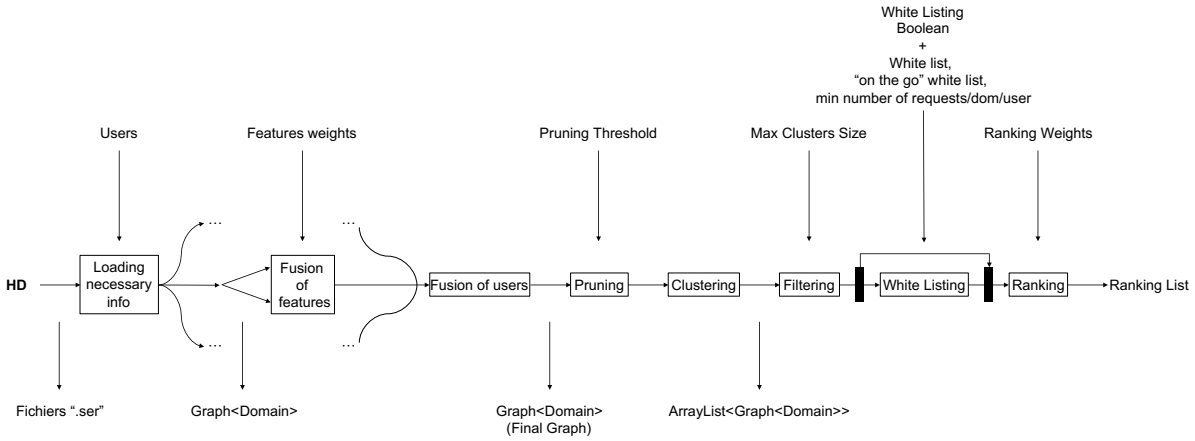


FIGURE 3.5 – Schéma du traitement effectué par le Server

```
MacBook-Pro-de-Thomas:server Thomas$ ./start.sh -h
usage: java -jar server-<version>.jar
 -h              Show this help
 -i <arg>        Input directory with graphs (required)
 -study <arg>    Study output mode (false = web output, true = study
                 output) (option, default: false)
MacBook-Pro-de-Thomas:server Thomas$
```

FIGURE 3.6 – Utilisation du Server

3.5.3 Utilisation

Le Server prend les arguments suivants comme entrée (Fig.3.6) :

- **-i** : le dossier dans lequel les données sont stockées ;
- **-study** : le choix concernant le mode d'opération (en mode *study*, le résultat renvoyé est limité au classement des domaines et au standard output afin d'accélérer les transferts de données) (optionnel, par défaut : *false*).

L'entièreté des paramètres accessibles à l'analyste est mise à sa disposition sur une page web. Celle-ci est détaillée dans la section suivante (Section 3.6). Lors du démarrage du Server, deux instances sont initiées :

- Interface utilisateur : `127.0.0.1:8000`
- Serveur JSON-RPC : `127.0.0.1:8080`

Le serveur JSON-RPC (JavaScript Object Notation, JSON) (Remote Procedure Call, RPC) assure l'échange de données entre l'interface et le Server. C'est ainsi que les paramètres choisis par l'analyste sont transmis au Server et que celui-ci répond avec les résultats calculés. L'interface utilisateur s'occupe alors d'afficher les résultats à l'analyste.

Une commande typique est donnée ci-après :

```
./start.sh -i <graphs directory>
```

3.6 Interface utilisateur

L'interface utilisateur est le composant final assurant l'interactivité entre l'algorithme et l'analyste. Cette interface assure deux rôles principaux. Il permet d'abord à l'analyste de fournir l'ensemble des paramètres nécessaires à l'algorithme. Il permet ensuite, dans une certaine mesure, de faciliter l'analyse des résultats obtenus. D'autres outils d'aide à l'analyse sont discutés dans la section 4.4. Le travail n'a pas pour objectif de réaliser une interface utilisateur. Celle-ci est donc rudimentaire et fonctionnelle (Fig.3.7 et Fig.3.8). Dans un premier temps, la section présente les paramètres disponibles sur l'interface. Ensuite, elle détaille les outils d'analyse proposés par cette interface.

3.6.1 Paramètres

Les paramètres nécessaires au processing du Server ont déjà été présentés dans la section traitant du Server (Section 3.5). Par souci d'exhaustivité, les paramètres disponibles sur l'interface sont repris ci-dessous :

- utilisateur ou sous-réseau étudié ;
- pondérations pour la fusion des similarités ;
- seuil de pruning (en valeur absolue ou en z-score) ;
- taille maximale des clusters affichés (en valeur absolue ou en z-score) ;
- white listing : choix de l'analyste quant à l'utilisation du white listing, du *on the go* white listing (Fig.3.9) et du nombre minimal de requêtes par domaine et par utilisateur ;
- pondérations pour le classement.

3.6.2 Analyse

Pour faciliter le choix du seuil de pruning et de la taille maximale des clusters pour le filtering, il est possible de voir la distribution de ces valeurs sur un histogramme (Fig.3.10 & Fig.3.11) [1] [32] [33]. Pour faciliter la consultation des informations contenues dans ces histogrammes, il est possible de visualiser les coordonnées d'un point à l'aide d'une info-bulle (Fig.3.10) et de sélectionner une zone afin de zoomer dessus (Fig.3.11). Une deuxième fonctionnalité permet d'aider l'analyste dans son choix du seuil de pruning. Cette fonctionnalité permet à celui-ci de visualiser la similarité d'un lien en mettant sa souris dessus (Fig.3.7 et Fig.3.8).

Pour refermer le cycle d'analyse, il est intéressant de voir ce que l'algorithme a permis d'isoler comme requêtes provenant du log lui-même. Il est donc possible d'afficher les requêtes liées aux domaines isolés par l'algorithme (Fig.3.12 [6]). Pour ce faire, deux méthodes sont proposées à l'analyste : soit l'affichage des requêtes uniquement liées aux domaines sélectionnés au moyen de la souris, soit l'affichage de l'entièreté des requêtes liées aux domaines isolés par l'algorithme.

Afin de faciliter le traitement de ces données, il est possible de n'afficher que les requêtes pour un certain type de fichier transféré (Fig.3.13). La liste des types proposés n'est pas exhaustive, et d'autres types peuvent être rajoutés. Les types proposés permettent de tester le fonctionnement de l'option.

Il est parfois intéressant d'exporter le standard output. Un bouton permettant la copie, dans le presse-papiers, du dernier output est disponible à cette fin (Fig.3.14). Finalement, les données consultées peuvent être sensibles. Pour faciliter la publication des graphes étudiés, une option d'anonymisation de la fenêtre a été ajoutée (Fig.3.15).

6. Les requêtes sont affichées systématiquement au format *Squid* (Section 4.2.1), et ce, quel que soit le format initialement utilisé dans le log.

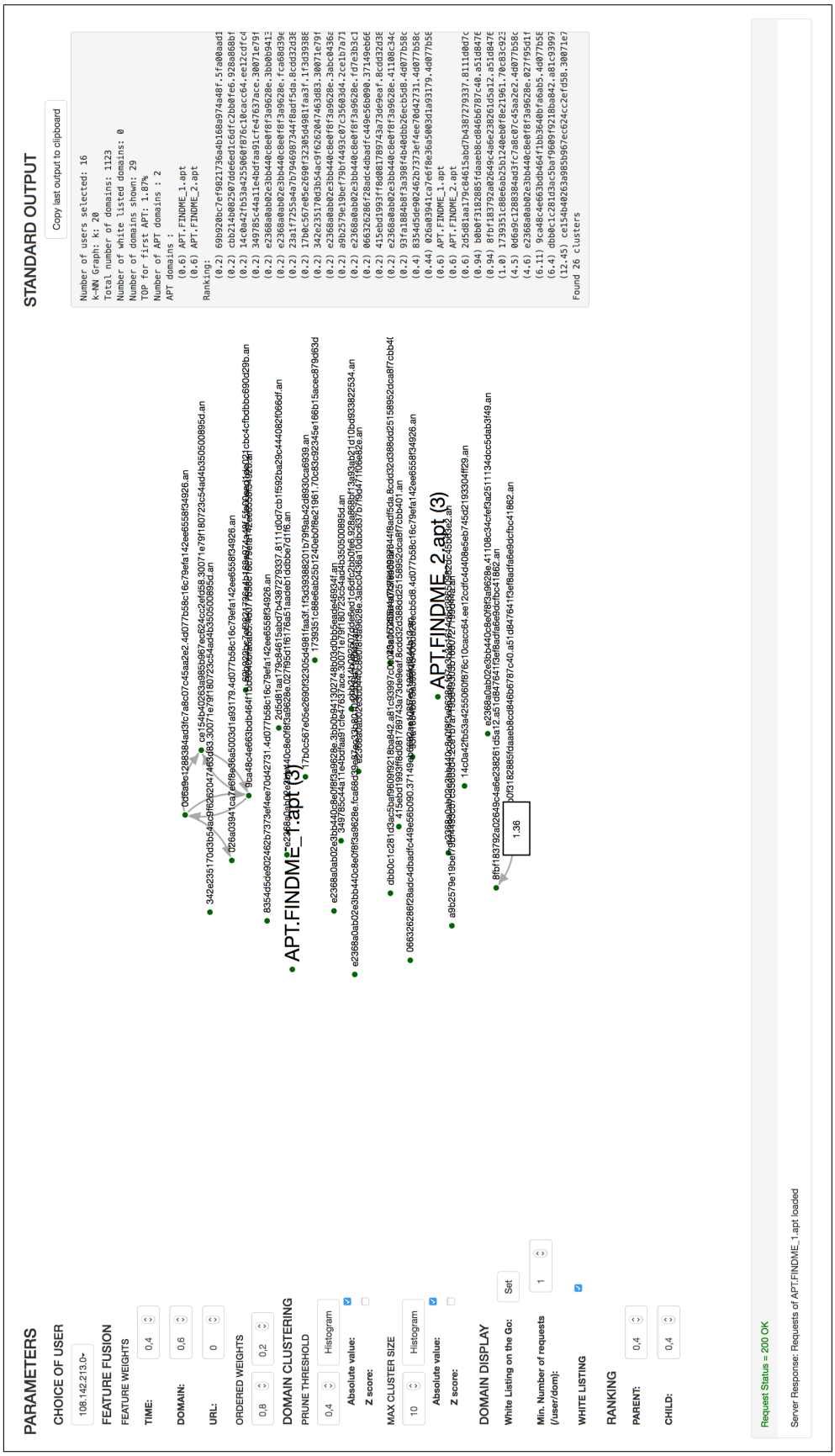


FIGURE 3.7 – Interface utilisateur, exemple de résultat sur des données anonymisées (avec affichage de la similarité d'un lien) (Partie 1)

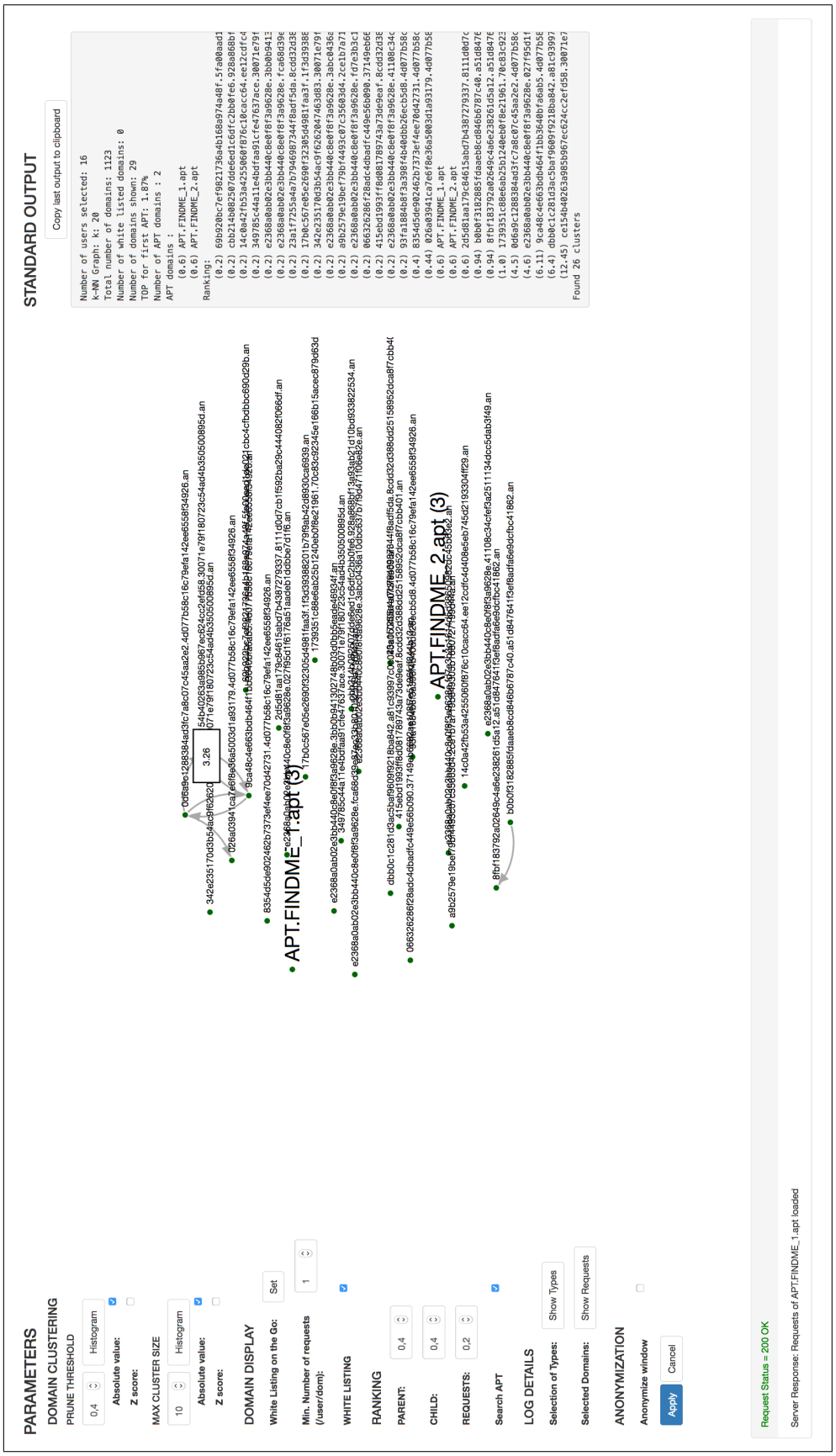


FIGURE 3.8 – Interface utilisateur, exemple de résultat sur des données anonymisées (avec affichage de la similarité d'un lien) (Partie 2)

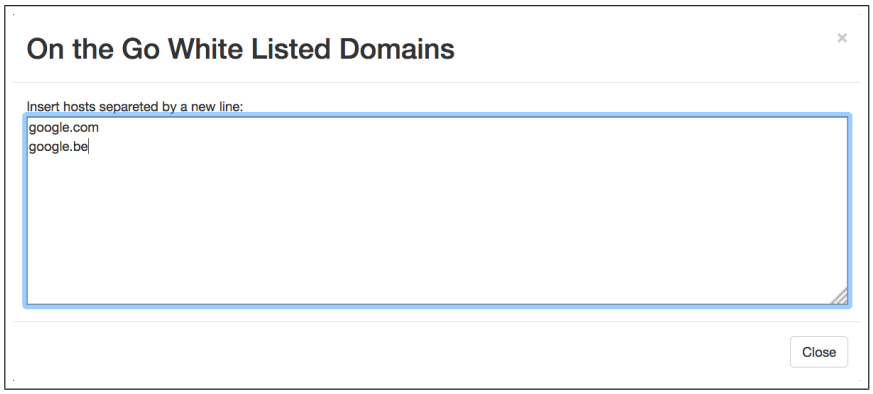


FIGURE 3.9 – Fenêtre permettant le white listing *on the go*.

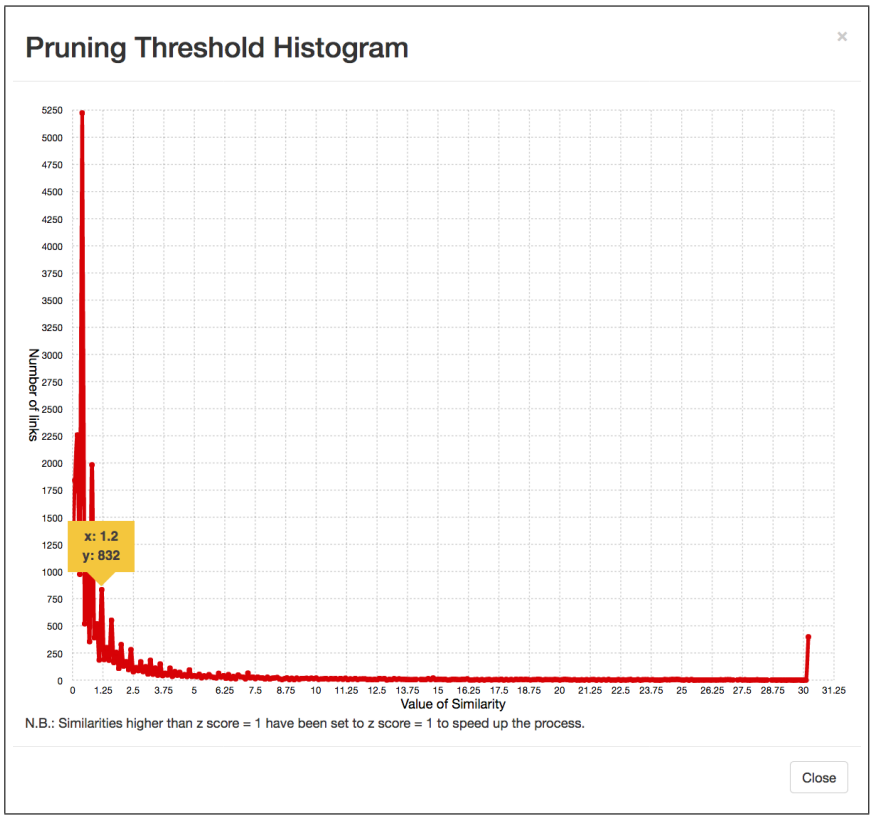


FIGURE 3.10 – Exemple d'histogramme des similarités (avec info-bulle [50])

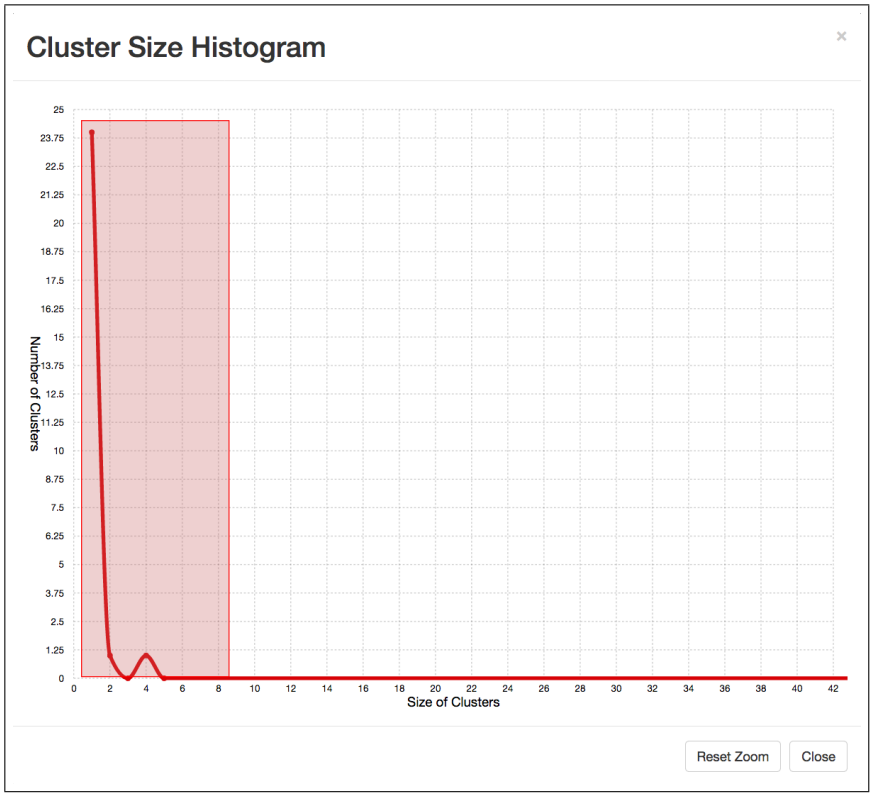


FIGURE 3.11 – Exemple d'histogramme des tailles de clusters (avec sélection d'une zone pour le zoom [30])

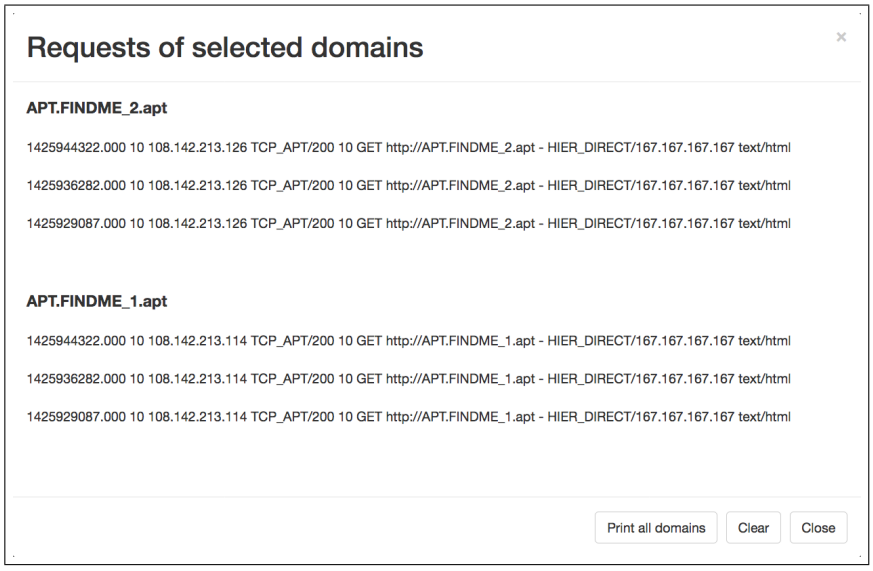


FIGURE 3.12 – Fenêtre permettant de visualiser les requêtes liées aux domaines sélectionnés au moyen de la souris sur la figure 3.7

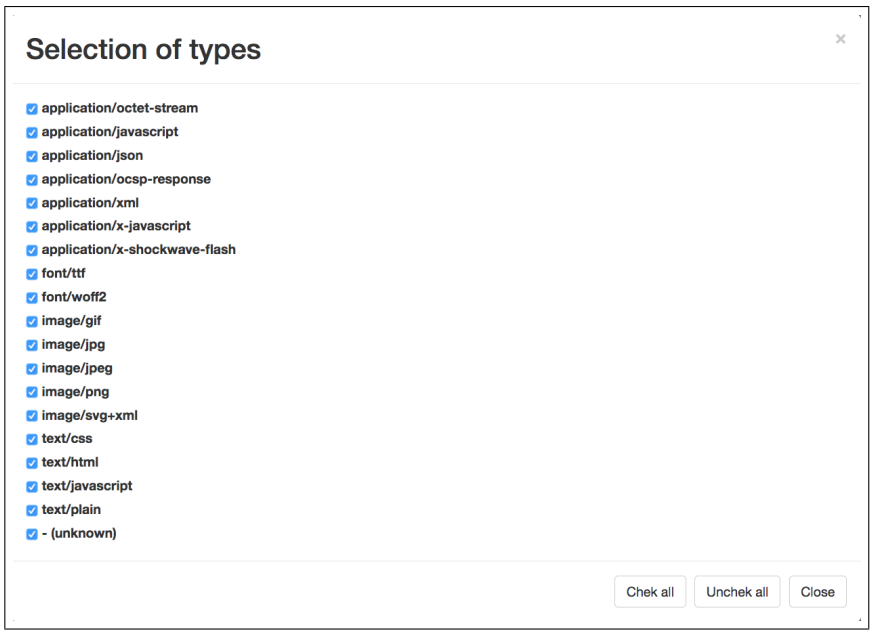


FIGURE 3.13 – Fenêtre permettant la sélection des types de fichiers affichés lors de l'impression des requêtes

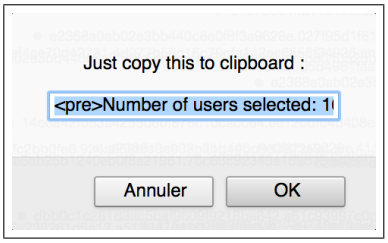


FIGURE 3.14 – Fenêtre permettant la copie, dans le presse-papiers, du dernier standard output

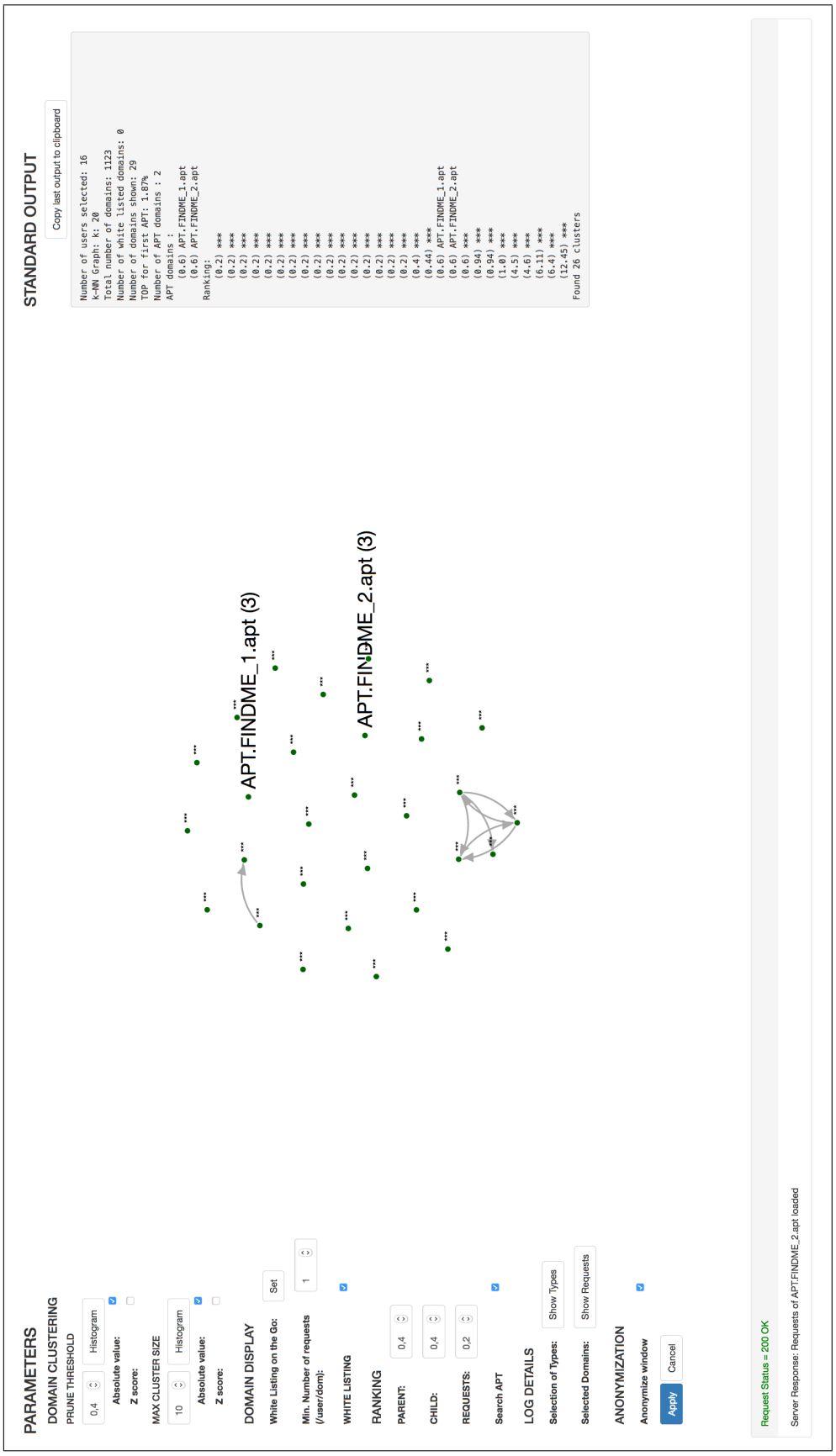


FIGURE 3.15 – Interface utilisateur anonymisée

3.7 Lien entre le phénomène physique et le processing

Afin d'implémenter l'algorithme, il a été nécessaire de se plier à différentes contraintes. Il faut maintenant s'assurer que, malgré ces contraintes, le graphe calculé soit cohérent avec le phénomène physique observé. De plus, il faut également vérifier que le comportement ciblé par l'algorithme soit bien celui d'une APT.

3.7.1 Graphe généré

Pour vérifier l'équivalence entre le graphe découlant du phénomène physique et le graphe calculé, chacun des deux graphes est d'abord présenté (en commençant par le graphe découlant du phénomène physique). Ensuite, les différences entre les deux graphes sont détaillées.

Il est souhaité d'avoir un graphe qui donne l'interdépendance entre les domaines présents dans le log étudié (Section 2.3). Il s'agit d'un graphe directif dont les similarités représentent l'intensité des relations parent/enfant. La répétition d'une relation pour un même utilisateur a pour caractéristique d'augmenter l'intensité de cette relation. Pour chaque utilisateur, un graphe modélisant l'ensemble des sites consultés est créé. Un graphe unique est alors obtenu par fusion de ces graphes individuels. La répétition d'une relation entre domaines pour deux utilisateurs a également la caractéristique d'augmenter l'intensité de cette relation. Le graphe idéal permettrait de distinguer clairement le C2 par des liens de faible intensité vers ses nombreux parents.

Le graphe calculé est un graphe directif de domaines. Pour chaque utilisateur, la similarité entre les domaines est définie comme une somme pondérée des similarités définies par l'analyste. Actuellement, l'intensité de la relation parent/enfant est évaluée sur base temporelle et du nom de domaine. Le renforcement d'un lien entre deux domaines est effectivement obtenu par répétition de ce lien chez un même utilisateur. Le graphe unique est obtenu par fusion des utilisateurs et la répétition d'un lien entre deux domaines chez deux utilisateurs renforce également ce lien.

Malgré les grandes similitudes entre le graphe découlant du phénomène physique et le graphe calculé, certains points peuvent être discutés. Ci-après sont discutées la perte des liens de faible intensité, le choix de mauvais voisins, la sélection de k requêtes voisines, la discrimination entre requêtes parentes et requêtes enfants, l'utilisation de graphes de domaines et la définition des similarités utilisées.

Perte des liens de faible intensité

Le fait d'utiliser des k-NN graphes pour les graphes des similarités au lieu de graphes complets est une simplification en soit. Les liens de parenté perdus sont les liens les plus faibles. Ceux-ci ne contribuent pas significativement à la somme permettant de calculer les similarités entre les domaines. Toutefois, l'algorithme a été construit pour augmenter la similarité entre deux domaines lorsqu'un lien est répété. Il faut donc être prudent en disant que l'impact de la perte des liens faibles est négligeable. Il est nécessaire de s'assurer une certaine masse critique de liens (donc, d'avoir un k minimum) afin de faire ressortir les structures des sites par répétition. Après un choix judicieux de k (ce choix est guidé dans la section 5.4.1), il peut être établi que cette hypothèse est admissible.

Choix de mauvais voisins

L'algorithme utilisé pour le calcul des k-NN graphes des similarités est l'algorithme *NN-Descent* [15] [18]. Cet algorithme construit de manière efficace une approximation du k-NN graphe idéal obtenu par force brute. Le voisin d'un noeud est correctement identifié si celui-ci est également

voisin de ce noeud dans le graphe obtenu par force brute. La précision avec laquelle les voisins sont correctement identifiés a été étudiée [18]. Il en ressort qu'il faut $k > 10$ pour obtenir de bons résultats et $k > 50$ pour obtenir plus de 90% de voisins corrects, en moyenne sur le graphe. Dans beaucoup de cas, ce résultat est obtenu bien avant $k = 50$. Finalement, il a été observé [18] que, au-delà d'un certain seuil, la précision n'augmente plus significativement avec un k croissant. Lors du choix de la valeur de k, il est donc nécessaire d'imposer une valeur minimum ($k > 10$).

Sélection de k requêtes voisines

Il est important de noter que le choix des k requêtes voisines se fait avant la sélection des requêtes enfants parmi ces mêmes requêtes voisines. Après cette sélection, les graphes ne sont donc plus des k-NN graphes. Il pourrait être souhaité de choisir k enfants plutôt que k voisins. Toutefois, il est admis que, si le k est judicieusement choisi, cette option n'est pas à préférer. En effet, le choix de k voisins assure des voisins appartenant probablement à la même page web. Ceux-ci proviennent probablement du même burst de données ou ont une grande similarité quant à leur nom de domaine.

Pour illustrer ces propos, l'expérience de la pensée suivante est proposée. Supposons une requête en fin d'un burst responsable du chargement d'une page web. En choisissant k voisins et en sélectionnant les enfants parmi ces voisins, il est probable de choisir des liens qui relient les domaines de la page web considérée. Puisque la requête est en fin de burst, ces liens sont ensuite éliminés lors de la sélection des liens enfants. Ce résultat est attendu, car cette requête n'a pas d'enfants. Par contre, en choisissant directement k enfants, de nombreux liens vers les requêtes de la page suivante, et donc vers les domaines du site suivant, sont créés. Ceci n'est pas souhaité, car cela crée des relations entre des domaines n'appartenant pas à la même page web.

Discrimination entre requêtes parentes et requêtes enfants

Comme déjà mentionnée dans la section 2.4.5, aucune information permettant la discrimination entre les requêtes parentes et enfants n'est disponible. Le choix a donc été fait de considérer chaque requête comme potentielle parente et de se baser exclusivement sur la répétition des liens pour faire ressortir les structures des sites. Cette déviation entraîne l'existence de liens parent/enfant entre des domaines qui ne devraient pas en avoir (ex. : un lien est forgé entre le dernier enfant d'une page et le parent de la page suivante). La conséquence la plus directe est que les sites risquent de ne pas pouvoir être clairement distingués. Le clustering risque donc de ne pas être capable de séparer tous les sites.

Utilisation de graphes de domaines

Le fait de réaliser un graphe de domaines plutôt qu'un graphe d'URL peut être discuté. Il est clair que, pour faciliter le rendu utilisateur, il est nécessaire de limiter le nombre de nœuds dans le graphe. Mais puisque l'APT utilise comme C2 un domaine compromis n'appartenant pas au trafic normal, cette simplification ne pose pas de problèmes. De plus, l'entièreté des données relatives aux requêtes est encore disponible dans le graphe final et il est possible de consulter les lignes du log relatives aux domaines affichés.

Définition des similarités utilisées

Afin de répondre aux contraintes réelles de chargement interrompu de pages et de dynamique des pages, l'utilisation d'une fusion de similarités s'impose de fait. En effet, une similarité temporelle

n'est pas suffisante pour distinguer APT et enfants légitimes, non permanents et faiblement liés. L'introduction d'une similarité basée sur les noms de domaine tente d'atténuer cette problématique.

Bien que l'algorithme ait été implémenté pour tirer profit d'un maximum de similarités, la définition de ces similarités est une difficulté en soi. Cette définition doit cibler une caractéristique intrinsèque du lien entre deux domaines appartenant à une même page web. Or, le problème est que ces caractéristiques sont peu nombreuses. Les similarités utilisées actuellement exploitent des caractéristiques évidentes du lien entre deux domaines d'une même page web : leur proximité temporelle et leur similarité de nom de domaine.

3.7.2 Comportement ciblé

Le C2 d'une APT est une anomalie dans le graphe, car celui-ci représente un enfant illégal, occasionnel et donc faiblement lié à ses parents. Idéalement, le C2 ne devrait même pas avoir de parents. En pratique, celui-ci a de nombreux liens faibles. Afin d'isoler ce C2 dans le graphe calculé, une succession de procédés est mise en œuvre. Chacun d'eux tend à exploiter la particularité de ces liens.

Le pruning coupe bon nombre des liens du C2, car ceux-ci sont faibles. Après pruning, le C2 peut techniquement encore avoir certains liens. Ceux-ci restent a priori faibles par rapport aux autres liens dans le graphe. Lors du clustering, cette faiblesse va avoir tendance à isoler le C2. Il y a un risque que le C2 soit absorbé dans un grand cluster, si les domaines de ce cluster apparaissent trop régulièrement comme parent ou enfant de l'APT.

L'élimination volontaire des grands clusters retire fondamentalement de l'information. Mais cette information n'a pas de plus-value. En effet, l'APT ne devrait pas se retrouver dans un grand cluster. Le résultat est donc nettoyé des clusters encombrants, qui correspondent aux modèles de sites. Ceci met le C2 en évidence. Pour permettre à l'analyste de se focaliser sur les domaines non reconnus avec certitude, le white listing a été mis en place. Cette étape renforce également la mise en évidence des domaines suspicieux.

Afin de faciliter l'étude du graphe, un classement des domaines est finalement effectué. Celui-ci cible clairement les domaines n'ayant que peu de voisins (parents et enfants) et n'ayant que peu de requêtes. Il cible donc un domaine isolé par le processing et relativement peu courant. Ce profil correspond au modèle d'une APT. La section 5.4.6 donne plus de détails quant à l'influence des poids du classement sur les caractéristiques ciblées.

3.8 Conclusion

Chacun des composants du processing a été passé en revue et a été détaillé. À chaque étape de la conception, les objectifs finaux ont été considérés : capacité de modéliser correctement les sites, capacité d'isoler un C2, interactivité et capacité de paramétriser l'algorithme. Il a été choisi d'effectuer l'ensemble des calculs lourds en amont dans le Batch Processor. Ce dernier calcule les graphes nécessaires sur base des similarités définies par le Core. Le Server exploite ces données et utilise l'interface graphique pour mettre à disposition un service interactif permettant le choix des paramètres et l'analyse des résultats. La perte d'informations a été réduite au minimum et l'analyste reste maître de l'ampleur de cette perte. Il peut être noté que la structure de la solution a l'avantage de permettre d'ajouter des similarités. Celles-ci ont pour objectif de renforcer les liens entre les parents et les enfants d'une même page web. Finalement, la cohérence du processing a été vérifiée. Il a été observé que certaines déviations existent. Afin d'assurer la meilleure modélisation possible, il est conclu qu'il est nécessaire d'avoir une valeur minimum de k ($k > 10$), d'avoir suffisamment de répétitions de liens entre domaines et, finalement, d'exploiter plusieurs similarités.

4. Environnement de développement

4.1 Introduction

Le processing a été détaillé au chapitre précédent. Ce chapitre-ci met en lumière certains éléments de l'environnement dans lequel la conception de ce processing a été faite. L'objectif est de présenter l'ensemble des outils utilisés et, ainsi, de faciliter la continuation du travail.

L'algorithme est implémenté en Java. L'ensemble des outils détaillés dans cette section le sont aussi. L'interface utilisateur est implémentée en HyperText Markup Language (HTML) et utilise du JavaScript et du Cascading Style Sheets (CSS). La page web utilise le framework Bootstrap [6]. Finalement, le Bash a été utilisé pour automatiser les tests. Le lecteur intéressé trouvera l'ensemble du code produit dans l'annexe B.

Le chapitre se divise en trois grandes parties. La représentation de certaines données fondamentales va d'abord être détaillée. L'objectif n'est pas de présenter de manière exhaustive l'entièreté des objets manipulés, mais plutôt, de souligner l'importance de certains objets et leur impact sur l'algorithme. Ensuite, les outils d'aide au développement vont être introduits. Ceux-ci sont présentés pour permettre la continuation du travail et non pour en détailler leur fonctionnement. La dernière partie présente des outils ayant été développés afin d'aider l'analyse de l'algorithme.

4.2 Représentation des données

Cette section cible certains objets fondamentaux et en décrit leur rôle. Seront détaillés les requêtes, les domaines, les graphes, les clusters, la mémoire du Server et les fichiers ".ser". Ces objets constituent le cœur de l'algorithme pour ce qui est des performances, de la paramétrisation et de la conservation de l'information lors du processing.

4.2.1 Requêtes

Lors de ce travail, deux types de fichiers log ont été traités. Ils sont référencés comme les types *Squid* et *JSON*. Chacun des deux types est brièvement présenté. L'objet `Request` est ensuite détaillé. Celui-ci contient l'entièreté de l'information d'une requête. Finalement, le mapping entre les champs des logs de chacun des deux types et l'objet `Request` sont présentés. L'objectif est de souligner les informations disponibles et les informations effectivement conservées dans la structure de données d'une requête.

Type *Squid*

Ce type sert de référence pour la construction de la structure de données des requêtes. Il est produit par un proxy implémenté sur base de *Squid* [9]. Ce type de proxy a été utilisé pour générer des logs simulés. Les détails des champs peuvent être trouvés sur le site [9]. Lorsque l'information d'un champ est indisponible, celle-ci est remplacée par " - ".

Une ligne de log se présente comme suit :

```
1425971539.000   1364 108.142.226.170 TCP_NC_MISS/200 342 GET
http://77efee5dcb3635e09435eb33a8351364.3da9819b747e806d78f83f22c703d178.an/
59a543c185f3330a33a47736d6879e16 - -/146.159.80.113 image/gif
```

Type *JSON*

Ce type est utilisé par les logs de données réelles étudiées. La quantité d'informations contenue dans ce type de log est plus importante. Les catégories nécessaires à l'utilisation de l'algorithme sont toutes présentes. Certaines informations supplémentaires sont ajoutées, par exemple, les informations de géolocalisation de l'adresse IP (Internet Protocol) du serveur. Ces informations sont dérivées des informations de base. Elles ne sont donc pas conservées dans la suite du processing. Une ligne de log prend typiquement la forme suivante :

```
{"@version":"1","@timestamp":"2014-10-10T23:12:24.000Z",
"type":"proxy_fwd_iwsva","timestamp":"Sat, 11 Oct 2014 01:12:24,CEST",
"tk_username":"192.168.2.167","tk_url":"http://weather.service.msn.com/
data.aspx?src=Windows7&amp;wealocations=wc:8040075&amp;weadegreetype=F&amp;
culture=en-US","tk_size":0,"tk_date_field":"2014-10-11 01:12:24+0200",
"tk_protocol":"http","tk_mime_content":"text/xml",
"tk_client_ip":"192.168.2.167","tk_server_ip":"92.122.122.162",
"tk_domain":"weather.service.msn.com","tk_path":"data.aspx",
"tk_file_name":"data.aspx","tk_operation":"GET",
"tk_uid":"0277354894-29fe6b562438c1f7e996","tk_category":"40",
"tk_category_type":"0","geoip":{"ip":"92.122.122.162","country_code2":"EU",
"country_code3":"EU","country_name":"Europe","continent_code":"EU",
"latitude":47.0,"longitude":8.0,"location":[8.0,47.0]},
"category":"Search Engines/Portals"}
```

Objet `Request`

La notion de requête a déjà été présentée. Celle-ci désigne une ligne de log. L'objet `Request` contient l'information d'une requête et a été formé sur base des entrées de type *Squid*. Ces informations sont également contenues dans les entrées de type *JSON*. Toutefois, le mapping n'est pas parfait et certaines catégories ne s'accordent pas avec celles de type *Squid*. L'entièreté des champs utilisés dans le type *Squid* se retrouve dans l'objet `Request`, ce qui n'est pas le cas pour le type *JSON*.

La structure d'une `Request` est commune au Batch Processor et au Server. Elle est donc définie dans le Core. Cet objet a les caractéristiques suivantes [29] :

- Time : Unix timestamp en millisecondes UTC (Coordinated Universal Time)
- Elapsed : Durée durant laquelle la transaction a occupé la cache du proxy
- Client : Adresse IP de l'instance effectuant la demande
- Code : Code de résultat Squid (décrit la réponse envoyée au client par différents tags)
- Status : Status code (HTTP result code avec quelques extensions spécifiques à Squid)
- Bytes : Quantité de données délivrée au client
- Method : Méthode de la requête
- URL : URL demandée
- Domain : Nom de domaine (champ généré localement à partir de l'URL)
- Peerstatus : Hierarchy codes (décrit comment la requête a été gérée par le proxy)
- Peerhost : Adresse IP à laquelle la requête a été envoyée
- Type : Type de contenu

JSON	⇒	Request
@timestamp		Time
/		Elapsed
tk_client_ip		Client
/		Code
/		Status
tk_size		Bytes
tk_operation		Method
tk_url		URL
Computed with tk_url		Domain
/		Peerstatus
tk_server_ip		Peerhost
tk_mime_content		Type

FIGURE 4.1 – Mapping du type *JSON* vers l'objet `Request`

La documentation de *Squid* définit un champ supplémentaire : *user*. Toutefois, celui-ci n'est pas utilisé. Il a donc été décidé de ne pas l'inclure dans l'objet `Request`.

Le fait que l'algorithme utilisé limite la perte d'information a déjà été souligné. L'objet `Request` est à la base de cette propriété. En effet, lors du parsing du fichier log, la majorité des informations contenues dans une ligne du fichier est stockée dans un objet de type `Request`. Cet objet assure donc une conservation maximale des informations.

Mapping des types de fichiers vers l'objet `Request`

En ce qui concerne le type *Squid*, le mapping est immédiat car le design de l'objet `Request` a été effectué sur base de ce type. Le mapping d'une requête de type *JSON* est plus délicat. Comme déjà mentionnés, les champs ne s'accordent pas parfaitement. Le mapping choisi est donné à la figure 4.1.

4.2.2 Domaines

La notion de requête a été décrite comme étant la base de la conservation de l'information lors du processing. La définition de la structure de données d'un domaine en est la continuité. L'objet `Domain` est défini dans le Core comme une liste de requêtes : `LinkedList<Request>`. Cette liste possède un nom : le nom du domaine lui-même.

Il est nécessaire d'être prudent lors de la manipulation des domaines. En effet, un domaine peut contenir les requêtes d'un utilisateur spécifique ou d'un ensemble d'utilisateurs. Ces deux domaines ont techniquement le même nom, mais leur contenu est lui différent. Afin de faciliter la manipulation des domaines, un inventaire des domaines est créé dans le Server. Celui-ci est de type `HashMap<String, HashMap<String, Domain>>`. Cet objet est référencé comme *all_ domains*. Il possède deux modes : *byUsers* et *all*. À chacun des modes est associé un `HashMap <String, Domain>`. En mode *byUsers*, chaque clé est spécifique à une combinaison d'un domaine et d'un utilisateur. La clé est définie comme suit : `user:domain` (ex. : `192.168.2.167: google.be`). Elle s'associe au domaine propre à l'utilisateur en question. Le mode *all* utilise la clé suivante : `domain` (ex. : `google.be`). Elle s'associe au domaine contenant l'ensemble des requêtes de ce domaine, tous utilisateurs confondus.

4.2.3 Graphes

Une fois les domaines définis, il est possible de construire des graphes. Ces graphes ont pour nœuds des domaines (`Graph<Domain>`) ou des requêtes (`Graph<Request>`). Chaque nœud possède une liste de voisins, et à chaque voisin est associée une similarité. L'implémentation des outils permettant la manipulation des graphes a été effectuée par Thibault Debatty [14]. Les principaux outils utilisés sont la création de k-NN graphes, le pruning et le clustering.

4.2.4 Clusters

Un graphe de clusters est un ensemble de sous-graphes. Chacun de ces sous-graphes représente un cluster. Ce dernier correspond à un `Graph<Domain>`. Un graphe de clusters est donc modélisé par une `ArrayList<Graph<Domain>>`.

4.2.5 Mémoire du Server

Un des objectifs du travail est de parvenir à créer un environnement interactif. Pour ce faire, un objet `Memory` a été créé. Celui-ci permet de conserver les résultats intermédiaires lors du traitement des graphes par le Server. Malgré la charge mémoire que cela représente, un avantage certain est obtenu quant à la réactivité. Le Server a été implémenté de telle façon qu'une nouvelle requête JSON n'entraîne que le strict minimum de calcul. Un maximum de données est récupéré du calcul précédent. Certaines parties du processing peuvent ainsi être passées, car le résultat intermédiaire est déjà connu. Il en résulte un clair avantage en matière de réactivité de la page web.

4.2.6 Fichiers ".ser"

Une fois le fichier parcouru et les objets `Request` créés, il est nécessaire de construire, sur base de ces requêtes, les graphes de domaines pour chacune des similarités. Cette opération s'effectue dans le Batch Processor (Section 3.4.1). Si le nombre d'utilisateurs est important, le nombre de graphes l'est d'autant plus. En effet, l'algorithme génère un graphe par similarité définie, et ce, pour chaque utilisateur. Pour alléger la charge mémoire lors du calcul de ces graphes, ceux-ci sont stockés sur le disque à chaque fois que les graphes liés à un utilisateur sont calculés. La mémoire est ainsi libérée pour les graphes de l'utilisateur suivant. Le fichier utilisé est un fichier ".ser" nommé par l'adresse IP de l'utilisateur en question (ex. : `192.168.2.167.ser`).

Le fait de stocker les graphes de chaque utilisateur séparément permet également de soulager la mémoire du Server. En effet, cette méthode de stockage permet de ne charger en mémoire que les graphes liés aux utilisateurs sélectionnés par l'analyste.

Du point de vue de l'analyse, cette séparation est également intéressante. Dans le cas où une attaque veut être simulée sur un utilisateur donné, seuls les graphes de l'utilisateur en question doivent être recalculés. L'algorithme du Batch Processor ne recalculera pas les données déjà présentent pour les autres utilisateurs (sauf si explicitement demandé au moyen de l'option **`-x`**).

4.3 Outils d'aide au développement

Un ensemble d'outils a été utilisé pour développer l'application : Maven, Grunt, des tests unitaires et GitHub. L'objectif est de présenter l'ensemble de ces outils afin de permettre une continuation aisée du travail. Les détails d'implémentation sont à retrouver dans le code lui-même (Annexe B).

4.3.1 Maven

Maven [57] est un outil de gestion de projets de développement. Celui-ci a été utilisé pour assister le développement du code Java lors de la phase de compilation. Il permet, entre autres, de vérifier le style du code Java écrit, d'inclure les dépendances dans le fichier compilé, de chercher de manière automatique des bugs dans le code ou encore d'exécuter automatiquement des tests unitaires (Section 4.3.3) [39] [40] [58]. Une certaine qualité de code est ainsi garantie. Le lecteur intéressé est invité à consulter les fichiers *pom.xml* des différents outils développés pour connaître l'ensemble des plug-ins utilisés.

4.3.2 Grunt

Grunt [27] est un outil permettant d'automatiser des tâches répétitives. Différents plug-ins Grunt ont été utilisés pour vérifier le code HTML et JavaScript [43] [44] [45] [46]. Ceci permet d'assurer une certaine qualité du code produit. Ainsi, ces plug-ins permettent de corriger les utilisations suspectes de l'HTML ou du JavaScript ainsi que le style du code. Le lecteur intéressé est invité à consulter le fichier de configuration *Gruntfile.js* pour connaître les détails sur les plug-ins utilisés.

4.3.3 Tests unitaires

Des tests unitaires ont été utilisés pour s'assurer que l'algorithme implémenté traite bien le graphe comme attendu. Ces tests vérifient que chacune des méthodes sensibles produit bien les résultats escomptés. L'intégrité des données est la principale cible des tests. Ceci signifie que les tests servent à vérifier que, entre deux étapes du traitement, il n'y a pas d'information perdue involontairement. Ainsi, il est vérifié que l'ensemble des requêtes manipulées sont toujours présentes et intactes, que les domaines possèdent bien l'ensemble de leurs requêtes, que les similarités évoluent comme souhaité, etc. Il n'est pas possible de valider directement l'algorithme, car aucun couple log / graphe n'est connu d'avance. Ces tests permettent donc de s'assurer d'un certain niveau de fiabilité.

Afin de connaître la couverture du code sur le plan des tests unitaires, l'outil Cobertura a été utilisé [10]. Celui-ci calcule la proportion du code qui est exécutée durant les tests et donne un rapport de couverture. Cet outil permet donc d'identifier les méthodes sensibles qui ne seraient pas couvertes par un test.

4.3.4 GitHub

GitHub [25] [61] a été utilisé pour assurer le contrôle de version et l'hébergement. Ce site donne accès à toutes les fonctionnalités de versioning mise à disposition par Git [24], mais offre également d'autres fonctionnalités qui lui sont propres. C'est un environnement permettant le développement de code Open Source. GitHub permet l'intégration de différents outils facilitant le développement. Les outils suivants ont été utilisés : Travis CI, Api123, Coveralls.

Travis CI [59] permet la vérification automatique des codes mis sur le dépôt. Ainsi, l'ensemble du code distribué est automatiquement vérifié par Maven et Grunt. Les résultats des tests sont affichés sur la page GitHub du projet [1].

Api123 [2] est un service permettant la distribution de documentation. La Javadoc est distribuée globalement grâce à ce service [2].

1. Les détails concernant les résultats des tests effectués par Maven et Grunt peuvent être retrouvés à l'adresse suivante : `https://travis-ci.org/RUCD/apt-graph`.

2. La documentation des outils disponibles sur GitHub peut être retrouvée à l'adresse suivante : `http://api123.io/api/apt-graph/head/index.html`.

Coveralls [11] distribue lui globalement les résultats produits par Cobertura. Celui-ci affiche le pourcentage de couverture sur la page GitHub de l'application. Il est important de noter ici que les tests unitaires ont été développés pour l'algorithme de traitement des graphes. C'est donc cet algorithme qui est testé. La couverture obtenue est de 76% [3]. Les outils additionnels présentés dans la section suivante (Section 4.4) ne sont eux pas pris en compte dans le calcul de la couverture des tests.

4.4 Outils d'aide à l'analyse

En plus de l'algorithme présenté au Chapitre 3, différents outils ont été écrits pour aider au développement de l'algorithme lui-même et pour aider à l'analyse du comportement de l'algorithme. L'objectif de la section est de présenter ces outils et leur fonctionnement. À nouveau, le lecteur intéressé est invité à consulter directement le code pour les détails d'implémentation (Annexe B). Ces outils n'ont pas vraiment d'intérêt d'un point de vue purement opérationnel, mais s'avèrent très utiles s'il est nécessaire de modifier ou d'étudier l'algorithme.

Le premier outil présenté est un outil permettant la simulation d'infection d'un log par une APT. Le second outil permet de visualiser au moyen d'un histogramme l'intensité du trafic contenu dans un log. Le troisième et le quatrième outil permettent ensemble de faire l'étude de l'influence d'un paramètre de l'algorithme sur la détection d'APT au moyen des ROC. Le cinquième et dernier outil est en fait un simple script permettant d'automatiser le processus d'analyse.

4.4.1 Infection de logs

Afin d'étudier les capacités de détection de l'algorithme, un outil d'infection de logs a été écrit (Fig.4.2). Celui-ci permet d'infecter un log avec deux types différents d'APT : une APT périodique et une APT basée sur les flux de données. Ces différents types ont déjà été présentés à la section 1.3.1. L'outil possède deux modes de fonctionnement, à savoir un par type d'APT, et offre un large éventail de paramètres. Ces modes vont être présentés et leurs paramètres expliqués.

APT périodique

Une APT périodique est caractérisée par un nom de domaine et un intervalle temporel entre les différentes contaminations. De plus, l'utilisateur infecté ainsi que le format des requêtes à injecter (*Squid* ou *JSON*) peuvent être choisis. L'outil prend les arguments suivants comme entrée (Fig.4.2) :

- **-i** : le fichier log à contaminer ;
- **-o** : le fichier dans lequel le résultat sera écrit (ceci évite de détruire les vraies données) ;
- **-t** : le type d'APT (dans ce cas-ci : *periodic*) ;
- **-f** : le format du fichier à infecter (*squid* ou *json*) (optionnel, par défaut : *squid*) ;
- **-d** : le nom du domaine de l'APT ;
- **-u** : l'utilisateur à infecter ;
- **-step** : le pas temporel entre les injections (en millisecondes).

Une commande typique est donnée ci-après :

```
./infect.sh -i <log file path> -o <output file path> -u <user ip> -d APT.FINDME.
    apt -t periodic -step 43200000
```

3. Les détails concernant la couverture du code par des tests unitaires peuvent être retrouvé à l'adresse : `https://coveralls.io/github/RUCD/apt-graph?branch=master`.

```
MBP-de-Thomas:infection Thomas$ ./infect.sh
usage: java -jar infection-<version>.jar
 -d <arg>            APT domain name (required)
 -delay <arg>        Delay between start of the burst and injection of APT (option for traffic APT,
                     default : middle of the burst)
 -delta <arg>        Maximal duration between two requests of the same burst (required for traffic
                     APT)
 -distance <arg>     Minimal time distance between two injections (option for traffic APT, default :
                     no limitation)
 -duration <arg>     Minimal duration of a burst to allow APT injection (required for traffic APT)
 -f <arg>            Specify format of input file (squid or json) (option, default : squid)
 -h                  Show this help
 -i <arg>            Input log file (required)
 -injection <arg>    Maximal daily number of injections (option for traffic APT, default : no
                     limitation)
 -o <arg>            Output log file (required)
 -proportion <arg>   Injection rate in the possible bursts (1 = inject in all possible bursts)
                     (option for traffic APT, default : 1)
 -step <arg>         Specify time step between periodic injections in milliseconds (required for
                     periodic APT)
 -t <arg>            Type (periodic or traffic) (required)
 -u <arg>            Targeted user or subnet (required)
MBP-de-Thomas:infection Thomas$
```

FIGURE 4.2 – Utilisation de l'outil d'infection de logs

APT basée sur les flux de données

Une APT basée sur les flux de données va tenter de se dissimuler au maximum en n'effectuant des requêtes qu'aux moments bien précis d'activité réseau de l'utilisateur. Cette APT n'apparaîtra donc jamais isolée dans les logs, c'est-à-dire à un moment où l'utilisateur, lui, n'est pas présent (en matière d'activité réseau).

Afin de choisir quand faire sortir l'APT, il est nécessaire de définir ce que l'APT considère comme étant une activité réseau. Une activité réseau est définie comme étant un flux continu de données pour un utilisateur donné pendant une durée donnée. Un flux continu de données est défini comme une succession de requêtes dont la distance temporelle est inférieure à une valeur prédéfinie donnée. L'outil prend les arguments suivants comme entrée (Fig.4.2) :

- **-i** : le fichier log à contaminer ;
- **-o** : le fichier dans lequel le résultat sera écrit (ceci évite de détruire les vraies données) ;
- **-t** : le type d'APT (dans ce cas-ci : *traffic*) ;
- **-f** : le format du fichier à infecter (*squid* ou *json*) (optionnel, par défaut : *squid*) ;
- **-d** : le nom du domaine de l'APT ;
- **-u** : l'utilisateur à infecter ;
- **-duration** : la durée minimale d'un burst de données autorisant l'injection d'une APT ;
- **-delta** : le délai maximal entre deux requêtes pour les considérer comme appartenant au même burst de données ;
- **-delay** : le délai entre le début du burst de données et l'injection de l'APT (optionnel, par défaut : injection au milieu du burst de données) ;
- **-distance** : la durée minimale entre deux injections d'APT (optionnel, par défaut : pas de limitation) ;
- **-injection** : le nombre maximal d'injections par jour (optionnel, par défaut : pas de limitation) ;
- **-proportion** : la proportion d'injections réellement faites par rapport au nombre total d'injections possibles (optionnel, par défaut : injection dès que possible).

```
MacBook-Pro-de-Thomas:traffic Thomas$ ./traffic.sh -h
usage: java -jar traffic-<version>.jar
 -f <arg>   Specify format of input file (squid or json) (option, default
            : squid)
 -h         Show this help
 -i <arg>   Input log file (required)
 -o <arg>   Output CSV file (required)
 -r <arg>   Time resolution in milliseconds (required)
MacBook-Pro-de-Thomas:traffic Thomas$
```

FIGURE 4.3 – Utilisation de l'outil d'analyse du trafic

Une commande typique est donnée ci-après :

```
./infect.sh -i <log file path> -o <output file path> -u <user ip> -d APT.FINDME.
    apt -t traffic -duration 5000 -delta 1000
```

L'option `delay` pourrait être remise en question. En effet, une APT ne peut pas prédire le trafic futur d'un utilisateur. Or, conceptuellement, c'est ce que cette option permet. En effet, l'APT est capable de connaître la durée d'un burst avant la fin de celui-ci et de s'y injecter n'importe où, si cette durée est supérieure à une valeur donnée. Dans un cas réel, l'APT ne peut pas savoir la durée du burst au moment de son injection. Elle s'injectera donc toujours après ce qu'elle considère comme la durée minimale pour un burst. En pratique, cette option donne la possibilité d'étudier l'influence sur la détection de la position temporelle de l'APT par rapport aux différents domaines parents. Il a été choisi d'injecter par défaut l'APT au centre du burst. Ceci permet une dissimulation maximale de l'APT dans les bursts de données. L'influence du délai pourrait faire l'objet d'une étude ultérieure (Section 6.2.3).

4.4.2 Étude du trafic

Il peut être intéressant d'étudier l'intensité du trafic d'un log donné. Pour ce faire, un outil d'analyse de trafic a été introduit. Celui-ci calcule l'histogramme du trafic pour une résolution donnée. Ceci permet de voir, par exemple, l'impact de l'injection des APT. Cet outil produit un fichier au format Comma-Separated Values (CSV) avec les données. L'histogramme peut alors être affiché par un programme comme Microsoft Excel ou Matlab (Code B.1 & Code B.2). L'outil prend les arguments suivants (Fig.4.3) :

- **-i** : le fichier log à analyser ;
- **-o** : le fichier CSV dans lequel seront écrites les données ;
- **-f** : le format du fichier à analyser (optionnel, par défaut : *squid* format) ;
- **-r** : la résolution de l'histogramme.

Une commande typique est donnée ci-après :

```
./traffic.sh -i <input log file> -o <output CSV file> -r 1000
```

Pour illustrer le fonctionnement de l'outil d'infection des logs et d'analyse du trafic, deux cas sont proposés : une infection périodique d'un utilisateur et une infection basée sur le flux de données du même utilisateur. Les histogrammes sont basés sur des données réelles et représentent 10 jours de trafic (Fig.1.1a & Fig.1.1b).

4.4.3 Étude des paramètres

Pour permettre une étude approfondie des paramètres disponibles pour l'analyste, deux outils complémentaires ont été mis en place. Le premier est un outil d'étude qui produit les Receiver

Operating Characteristics (ROC) correspondant aux paramètres du Server fournis dans un fichier de configuration. Le second outil permet, lui, de générer de manière automatique le fichier de configuration nécessaire au premier lors du balayage de la valeur d'un paramètre.

Dans la suite de cette section, les ROC sont introduites et définies. Ensuite, une ligne de configuration typique utilisée pour générer une ROC est présentée. Après, les outils permettant la production de ROC et le balayage d'un paramètre sont introduits. Finalement, l'attention du lecteur est attirée sur l'intérêt de l'utilisation de l'objet `Memory`.

Receiver Operating Characteristic

La notion de Receiver Operating Characteristic (ROC) est largement utilisée pour évaluer la performance d'un détecteur. Cette courbe représente la probabilité de détection attendue d'un détecteur donné, pour une probabilité de fausse alarme donnée. Il est évidemment souhaité d'obtenir la plus grande probabilité de détection, pour la plus faible probabilité de fausse alarme.

Dans le cadre de ce travail, les probabilités de détection et de fausses alarmes sont dérivées du classement des domaines produit par l'algorithme. Pour un rang donné dans le classement, il est possible d'estimer la probabilité de détection et de fausses alarmes en supposant l'étude des domaines du rang donné et des rangs précédents dans le classement.

La *probabilité de détection* est définie par le rapport entre le nombre de domaines C2 détectés au rang donné et aux rangs précédents, et le nombre total de domaines C2 présents dans le log.

La *probabilité de fausses alarmes* est définie par le rapport entre le nombre de domaines considérés au rang donné et aux rangs précédents (domaines C2 exclus), et le nombre total de domaines étudiés (domaines C2 exclus).

Il faut être prudent avec ces définitions pour deux raisons. Premièrement, ces notions sont nommées *probabilité* à tort. Leur évaluation n'est qu'une grossière approximation de la probabilité. Deuxièmement, les analyses effectuées sont faites sur des données réelles. Il ne peut être garanti que, dans ces données réelles, aucuns autres domaines que ceux intentionnellement injectés aient un comportement semblable à celui ciblé. Il est même possible que le log contienne une trace d'infection d'une APT non détectée. Les termes *détection* et *fausse alarme* doivent donc être employés avec précaution. Le terme *détection* est limité aux domaines C2 intentionnellement injectés dans le log et le terme *fausse alarme* étendu à tous les autres domaines.

La notion d'*aire sous la courbe* (Area Under the Curve, AUC) est, quant à elle, utilisée pour comparer plusieurs courbes entre elles. L'AUC mesure la puissance de discrimination du détecteur [21]. Une AUC égale à 1 correspond théoriquement à une discrimination parfaite alors qu'une AUC égale à 0,5 correspond à un détecteur n'étant pas discriminant. Ce dernier cas correspond à une ROC en ligne droite, partant du coin inférieur gauche et allant jusqu'au coin supérieur droit du graphe. Dans ce cas, la probabilité de fausse alarme évolue à la même vitesse que la probabilité de détection.

Il est souhaité de choisir la combinaison de paramètres donnant une AUC maximale. Ceci correspond au cas où la courbe est la plus proche possible du coin supérieur gauche du graphique. Ce cas assure la plus grande discrimination de la part de l'algorithme.

Finalement, il est à noter que les premières analyses effectuées auront des AUC inférieurs à 0,5. Ceci s'explique par le fait que ces courbes sont obtenues pour des cas clairement défavorables du point de vue de la combinaison de paramètres.

Configuration

L'outil permettant la production des courbes ROC utilise le Server pour produire des classements et, sur base de ces classements, calcule les ROC. Il a été expliqué à la section 3.5 que le Server

```
MacBook-Pro-de-Thomas:study Thomas$ ./study.sh -h
usage: java -jar study-<version>.jar
 -h         Show this help
 -i <arg>   Input configuration file (required)
 -x <arg>   Overwrite existing files (option, default : false)
MacBook-Pro-de-Thomas:study Thomas$
```

FIGURE 4.4 – Utilisation de l'outil permettant l'étude des paramètres du Server

prend une série de paramètres en entrée. Ceux-ci sont traditionnellement choisis dans l'interface utilisateur par l'analyste. Toutefois, afin d'automatiser les tests et la production des ROC, il est intéressant de pouvoir fournir ces paramètres autrement. Il a été choisi de pouvoir donner la configuration sous le format JSON.

Un fichier type de configuration est composé de plusieurs lignes. Chacune de ces lignes correspond à un ensemble de paramètres. Chaque ligne fournit donc les paramètres normalement donnés par l'interface utilisateur, mais fournit également le dossier à analyser (`input_dir`), le nom de fichier type dans lequel seront écrits les résultats (`output_dir`) et le nombre de domaines APT dans le fichier log (`n_apt_tot`). Ainsi, une ligne de configuration correspond typiquement à :


```
{"input_dir":"/Users/Thomas/Desktop/TFE_Data/anon/infected/",
"output_file":"/Users/Thomas/Desktop/TFE_Data/anon/CSV/ROC_anon.csv",
"n_apt_tot":"2","user":"108.142.213.0","feature_weights_time":"0.6",
"feature_weights_domain":"0.4","feature_weights_url":"0.0",
"feature_ordered_weights_1":"0.8","feature_ordered_weights_2":"0.2",
"prune_threshold":"0.01","max_cluster_size":"5","prune_z":"false",
"cluster_z":"false","whitelist":"true","white_ongo":"","number_requests":"1",
"ranking_weights_parents":"0.4","ranking_weights_children":"0.4",
"ranking_weights_requests":"0.2","apt_search":"true"}
```


Production de ROC

Sur base des paramètres fournis par une ligne du fichier de configuration, ce premier outil produit les courbes ROC correspondantes. Ces courbes sont ensuite sauvegardées en format CSV. Le résultat peut alors être affiché par un programme comme Microsoft Excel ou Matlab (Code B.3 & Code B.4). Le chapitre 5 utilise abondamment les courbes produites par cet outil. Le lecteur est donc invité à consulter ce chapitre pour voir les résultats produits. L'outil prend les arguments suivants en entrée (Fig.4.4) :

- **-i** : le fichier de configuration de l'analyse ;
- **-x** : le choix concernant le recalcul des données déjà présentes dans le dossier de sortie (optionnel, par défaut : false).

Une commande typique est donnée ci-après :

```
./study.sh -i <input configuration file>
```

Balayage de paramètres

Afin d'effectuer une analyse systématique des paramètres, il peut être intéressant de connaître l'influence de la variation d'un paramètre sur la ROC (gardant les autres paramètres constants). Pour ce faire, ce deuxième outil a été introduit. Celui-ci prend une ligne de configuration de base comme entrée et fait varier un paramètre cible sur une gamme de valeurs prédéfinies.

```
MacBook-Pro-de-Thomas:config Thomas$ ./config.sh -h
usage: java -jar config-<version>.jar
 -field <arg>   Configuration field to sweep (required)
 -h             Show this help
 -i <arg>       Input configuration file (default configuration line)
                (required)
 -multi <arg>   Sweep the given field as complement to stop value of the
                first field (option, default : no second field)
 -o <arg>       Output configuration file (required)
 -start <arg>   Start value of sweep (required)
 -step <arg>    Step of sweep (required)
 -stop <arg>    Stop value of sweep (required)
MacBook-Pro-de-Thomas:config Thomas$
```

FIGURE 4.5 – Utilisation de l'outil permettant la génération de fichiers de configuration

Dans certains cas, il peut être intéressant de faire varier un second paramètre en complément du premier. L'option `multi` a été introduite pour permettre ce type de tests. La valeur de ce second champ est calculée comme étant égale à la différence entre la valeur d'arrêt du balayage et la valeur actuelle du premier champ. L'outil prend les paramètres suivants comme entrée (Fig.4.5) :

- **-i** : le fichier contenant la ligne de configuration de base ;
- **-o** : le fichier dans lequel l'ensemble des lignes de configuration seront écrites ;
- **-field** : le champ de la configuration visée par le balayage ;
- **-start** : la valeur de départ du balayage ;
- **-stop** : la valeur d'arrêt du balayage ;
- **-step** : le pas du balayage ;
- **-multi** : le deuxième champ visé par le balayage, dans le cas d'un double balayage (optionnel, par défaut : pas de deuxième champ).

Une commande typique est donnée ci-après :

```
./config.sh -i <default configuration file> -o <output configuration file> -field
    prune_threshold -start 0.0 -stop 1.0 -step 0.1
```

Mémoire du Server

Par souci d'optimisation, l'entièreté des configurations d'un même fichier est calculée avec le même Server. Ceci implique que l'outil tire parti de l'objet `Memory` détaillé à la section 4.2.5. Il s'ensuit une efficacité accrue du calcul des ROC pour un balayage donné.

4.4.4 Bash scripting

Afin d'automatiser les procédures au maximum et de limiter les erreurs humaines lors de la manipulation des fichiers, le bash scripting est utilisé. Celui-ci implémente la succession d'opérations nécessaires à la production des résultats présentés au chapitre 5. L'utilisation de ce type de script permet la répétabilité des résultats, mais offre également une certaine facilité pour le recalcul des résultats en cas de mise à jour de l'algorithme. De plus, vu la durée d'exécution de certaines parties de l'algorithme, il est nécessaire d'être capable d'automatiser le processus. En effet, l'exécution complète de la procédure peut durer plus de deux jours. Afin de gagner du temps lors de l'exécution du script et pour utiliser au maximum la puissance de calcul disponible, certaines étapes sont exécutées en parallèle.

Le script principal peut être résumé comme suit (Code B.5) :

1. décompression des fichiers logs,
2. extraction des données du sous-réseau considéré (Code B.6),
3. infection des logs (Code B.7),
4. analyse du trafic (Code B.8),
5. calcul des graphes non infectés,
6. duplication des graphes non visés par l'infection dans le dossier des graphes infectés,
7. calcul des graphes infectés (uniquement les utilisateurs nécessaires),
8. création des fichiers de configuration,
9. calcul des ROC.

Les fichiers manipulés peuvent être extrêmement volumineux. Un log de 24h peut dépasser les 10GB de données. Il est donc nécessaire d'utiliser du Bash, même pour des opérations simples, car un éditeur de texte classique n'est pas capable d'ouvrir le fichier. Pour limiter la taille des fichiers à charger en mémoire et à manipuler durant les différentes étapes décrites précédemment, un script a été prévu (Code B.6). Celui-ci permet la sélection d'un sous-réseau parmi l'ensemble des sous-réseaux disponibles dans le log. Cette étape permet donc d'anticiper le choix laissé à l'analyste sur l'interface utilisateur. Le fait de cibler le sous-réseau plus tôt dans le processing offre un gain considérable quant à la charge de calcul.

4.5 Conclusion

Ce chapitre s'est consacré à la présentation de l'environnement de développement de l'algorithme. Il s'est attardé sur les points principaux permettant la continuation du travail. Une première section a été consacrée à la description des structures de données de certains éléments cruciaux de l'algorithme. Il a été expliqué que ces éléments impliquent directement la conservation d'un maximum d'information jusqu'au résultat final, mais également la réactivité de l'interface et la paramétrisation de l'algorithme. Les outils utilisés pour le développement ont ensuite été présentés. Ces outils sont mondialement utilisés dans le monde du développement Open Source. Il a été nécessaire de mentionner leur utilisation, sans pour autant devoir en faire une description très élaborée. La dernière section a présenté les outils développés pour faciliter l'analyse de l'algorithme, à savoir : un outil permettant la simulation d'infection d'un log par une APT, un outil calculant l'histogramme du trafic contenu dans un log, deux outils permettant ensemble l'étude de l'influence d'un paramètre de l'algorithme sur la détection d'APT et, finalement, un outil permettant d'automatiser l'analyse. Ceux-ci ont été utilisés pour obtenir les résultats du chapitre suivant.

5. Choix des paramètres de l'algorithme

5.1 Introduction

L'algorithme, ainsi que l'ensemble de l'environnement disponible pour l'analyste, sont connus. Ce chapitre a maintenant pour objectif de guider l'analyste dans le choix des nombreux paramètres proposés par cet algorithme. Les principaux paramètres vont être analysés et leur impact sur la détection d'APT étudié. Il est très difficile d'être exhaustif dans les cas étudiés. Certains cas clés ont donc été choisis et sont ciblés dans cette étude. Pour ce faire, un environnement de test a été choisi. Celui-ci est composé d'un sous-réseau particulier, à un moment particulier, et est infecté d'APT particulières. Ces éléments sont fixes pour l'étude. Sur base de cet environnement, l'influence des paramètres est étudiée et l'analyste conseillé(e).

Une fois l'ensemble de paramètres choisi, deux tests plus spécifiques sont effectués. Un premier cible particulièrement les APT basées sur le flux de données. Un second cible lui les APT périodiques. Bien que les APT périodiques ne soient pas la cible première pour cet algorithme, il est montré que celui-ci permet leur détection. D'autres méthodes existent pour détecter les APT périodiques, par exemple l'analyse fréquentielle du log [36].

Un dernier test est effectué pour confirmer la flexibilité des paramètres choisis. Ce test est effectué sur un nouveau sous-réseau avec une gamme plus large d'infections simultanées. L'objectif est de démontrer la capacité de détection de l'algorithme lorsque l'ensemble de paramètres déterminé est utilisé.

Une brève explication sur les domaines isolés par l'algorithme est ensuite fournie. Finalement, un résumé des résultats obtenus dans le chapitre est donné sous forme de conseils à l'analyste.

5.2 Choix d'un log

Il est choisi de travailler dans un environnement de données réelles. Le flux de données fait office de bruit de fond. L'algorithme a été conçu pour faire ressortir un comportement de type APT de ce bruit. Il n'est pas acquis que le trafic d'un sous-réseau ait des caractéristiques identiques avec celles d'un autre sous-réseau. Dans le cadre de ce travail, il est choisi de travailler avec un sous-réseau en particulier. Ce sous-réseau est choisi arbitrairement, mais le choix est guidé par la taille du sous-réseau. Il est clair qu'une étude plus approfondie des implications de ce choix pourrait être faite (Section 6.2.4).

Pour des raisons purement pratiques, il est souhaité de travailler avec un sous-réseau relativement restreint. La taille du sous-réseau a un impact direct sur la quantité de données à manipuler et donc sur le temps de calcul. Ce choix permet de faire un plus grand nombre de tests. Le sous-réseau choisi est composé de 26 utilisateurs.

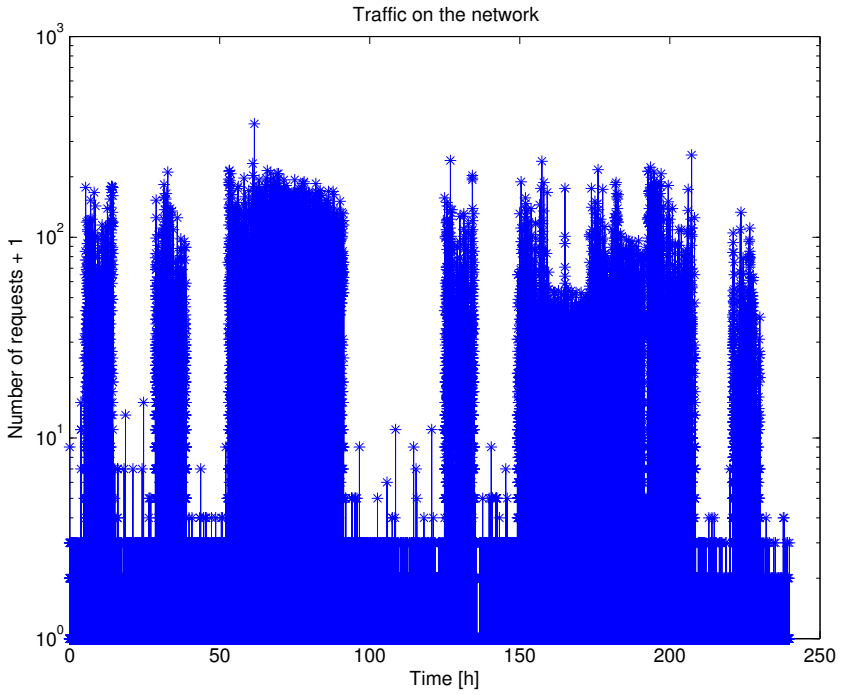


FIGURE 5.1 – Trafic du sous-réseau choisi (Résolution : 1s)

Le choix du sous-réseau n'est pas suffisant et il est également nécessaire de choisir une période particulière. Une période de 10 jours consécutifs est sélectionnée de manière arbitraire. Une étude plus approfondie de l'impact de ce choix pourrait être effectuée (Section 6.2.5). La période étudiée s'étale d'un mercredi au vendredi de la semaine suivante. Le trafic du sous-réseau étudié est présenté à la figure 5.1. Il peut être observé que certains jours sont clairement distincts. Toutefois, il semble que certains utilisateurs aient parfois une activité nocturne relativement intense. Au total, le fichier comptabilise 721 921 requêtes (infections comprises, voir Section 5.3) et 4 310 domaines.

5.3 Infection d'un log

Pour évaluer les capacités de détections de l'algorithme, il est nécessaire de connaître des domaines C2 pour pouvoir observer leur position dans le classement produit par l'algorithme. Or, aucun log ne peut être fourni avec une trace réelle d'APT connue. Afin d'évaluer les capacités de l'algorithme, des APT sont intentionnellement injectées dans le log. Comme déjà expliqué dans la section 4.4.3, il n'y a aucune garantie que les logs utilisés ne contiennent pas déjà des traces d'une infection. Toutefois, la notion de détection est limitée aux domaines intentionnellement injectés. Cette section introduit d'abord les cas d'infection traités. Elle détaille ensuite chacune des infections.

5.3.1 Cas d'infection traités

Pour que l'évaluation de l'algorithme soit la plus générale possible, des cas extrêmes sont considérés. Deux types d'APT ont déjà été définies (Section 4.4.1) et sont utilisées pour les infections, à savoir les APT périodiques et les APT basées sur les flux de données. Le sous-réseau utilisé contient 26 utilisateurs. Pour maintenir une proportion relativement faible d'infection, seulement

4 utilisateurs sont infectés en même temps. Si la proportion d'infections est trop importante, la probabilité qu'un domaine injecté ressorte relativement haut dans le classement est plus grande, et ce, même si l'algorithme n'est pas spécialement efficace à la détection. Deux cas extrêmes sont considérés par type d'APT, à savoir, une infection discrète et une infection agressive. Chacun des quatre utilisateurs est infecté par une de ces APT.

L'intérêt de ce scénario réside dans le large spectre de cas brassés. Il est attendu que, si l'algorithme est à la fois efficace pour la détection d'APT tant discrètes qu'agressives, il devrait également l'être pour des APT se situant entre ces deux extrêmes. De plus, ce scénario évalue la capacité de l'algorithme sur les deux types d'APT considérées. Finalement, une infection plus large du sous-réseau par une même APT est également simulée dans une certaine mesure. En effet, la manière dont a été définie la similarité entre deux noeuds du graphe final implique directement qu'une infection intensive d'un utilisateur unique soit comparable à une infection discrète d'un plus grand nombre d'utilisateurs.

La dernière affirmation peut être appuyée par le raisonnement suivant. Imaginons un cas très simple où l'APT ne sort que dans deux bursts de données sur la durée du log et pour un utilisateur. Pour chaque sortie, k similarités sont calculées et, parmi celles-ci, seuls les enfants temporels sont conservés. Supposons un enfant temporel présent dans les deux bursts de données et évaluons la similarité entre ce domaine et le C2. Celle-ci est égale à la somme des deux similarités individuelles, multipliées par le poids de la similarité temporelle. Supposés distincts, les deux domaines considérés n'ont pas de similarité basée sur leur nom de domaine.

Imaginons maintenant un cas où la deuxième sortie et les enfants temporels associés sont en fait liés à un autre utilisateur. La similarité entre les domaines est alors égale à la somme des produits des similarités individuelles et du poids de la similarité temporelle. Grâce à la propriété de distributivité de la multiplication par rapport à l'addition, les deux expressions sont mathématiquement équivalentes (Annexe A). Il est donc mathématiquement équivalent de concevoir un utilisateur fictif infecté par une APT agressive et un ensemble d'utilisateurs infectés par une APT discrète.

5.3.2 Détails des infections

Les quatre infections utilisées dans les tests sont maintenant détaillées (Code B.7).

La première infection est une infection périodique. Sa période est de 1h (Fig.5.2). Cette infection est relativement agressive. Au total, 239 sorties de l'APT peuvent être comptabilisées. Cette APT sera référencée comme *APT1*.

La seconde infection est également périodique, mais avec une période de 12h (Fig.5.3). Cette infection est moins agressive que la première et comptabilise 19 sorties de l'APT. Cette APT sera référencée comme *APT2*.

La troisième infection est une infection par une APT se basant sur le flux de données (Fig.5.4). Il a été choisi de définir comme burst acceptable pour l'injection d'une APT un burst constitué d'un ensemble de requêtes distantes temporellement de maximum 2 secondes, et ce, pendant 10 secondes. Ceci assure la présence de l'utilisateur. L'APT est injectée au milieu de ce burst, c'est-à-dire avec un délai de 5 secondes. Ceci garantit une certaine dissimulation de l'APT. Comme déjà mentionnée précédemment, l'influence du délai pourrait être plus largement étudiée (Section 6.2.3). Le nombre d'injections par jour est limité à 3 et la distance temporelle minimale entre deux injections est de 3h. L'ensemble de ces paramètres entraîne 13 injections dans le log pour cette APT. Celle-ci sera référencée comme *APT3*.

La quatrième et dernière infection est également une infection basée sur le flux de données (Fig.5.5). Il a été décidé d'être plus strict sur la définition du burst en n'autorisant une distance

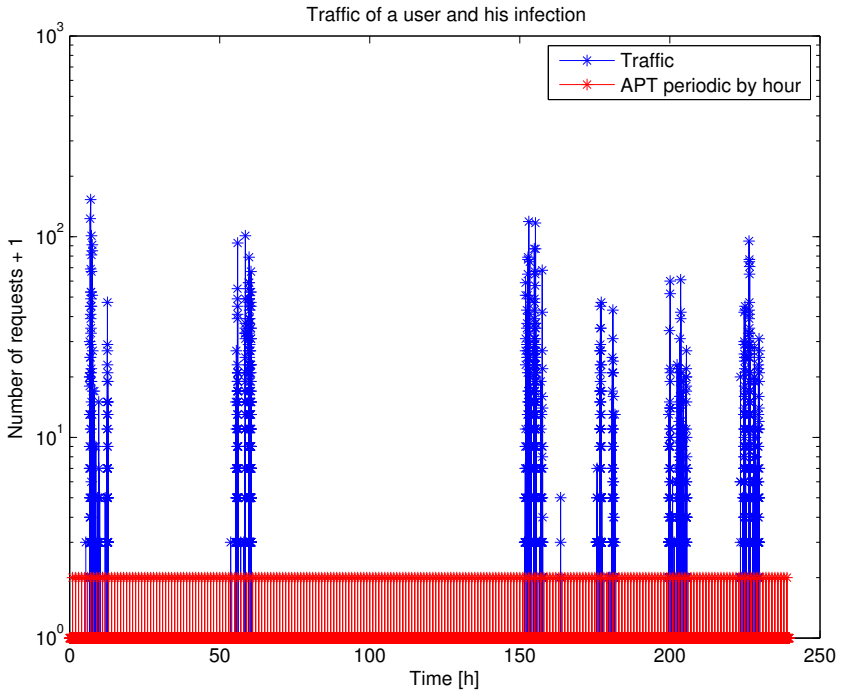


FIGURE 5.2 – Trafic de l'utilisateur infecté par une APT périodique ayant une période de 1h (Résolution : 1s)

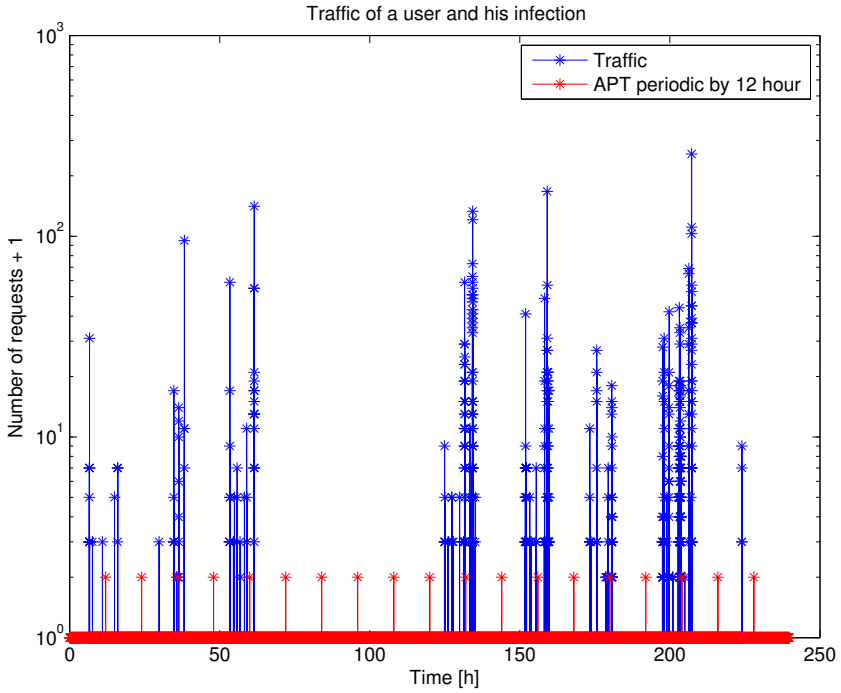


FIGURE 5.3 – Trafic de l'utilisateur infecté par une APT périodique ayant une période de 12h (Résolution : 1s)

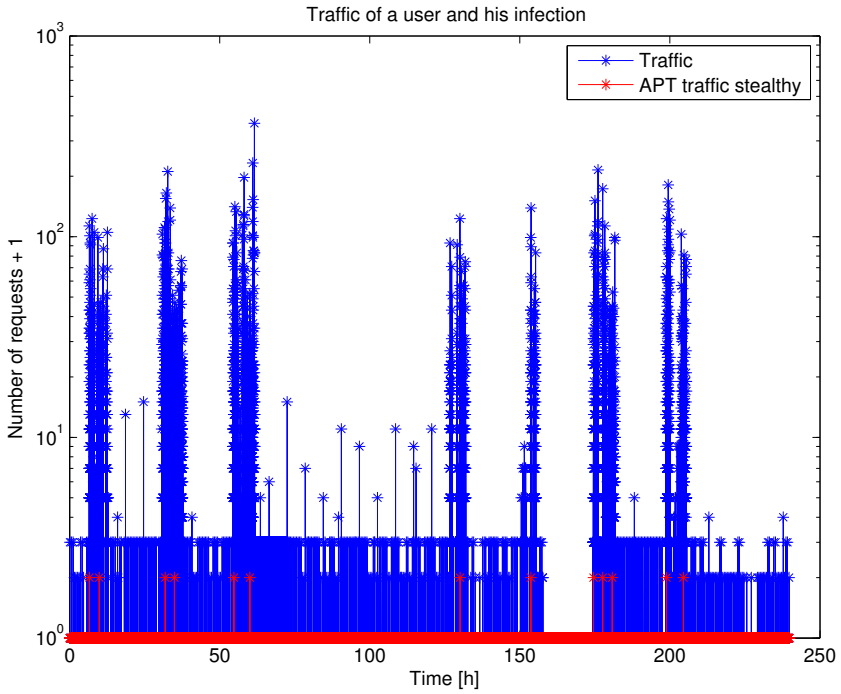


FIGURE 5.4 – Trafic de l'utilisateur infecté par une APT discrète basée sur le flux de données (Résolution : 1s)

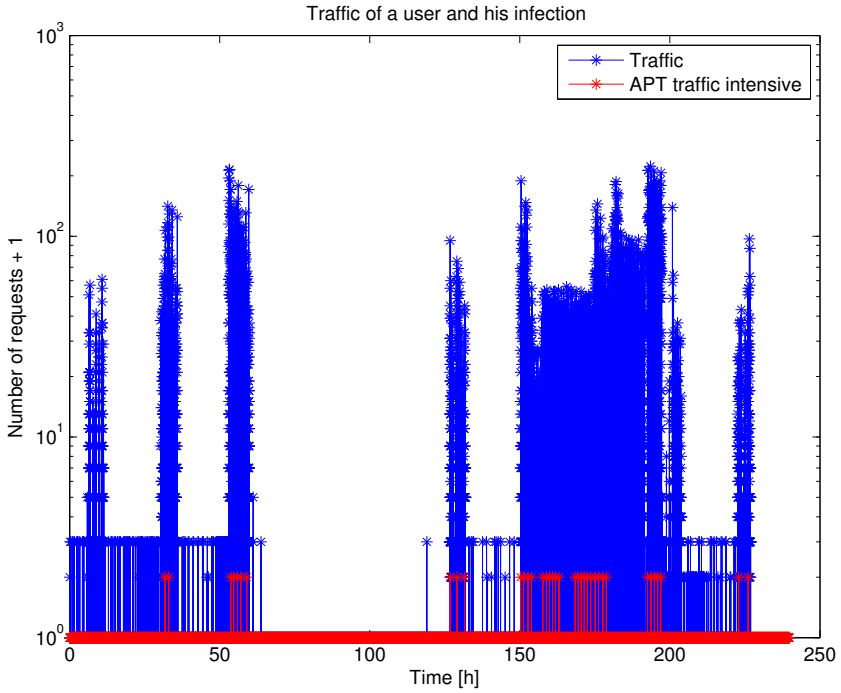


FIGURE 5.5 – Trafic de l'utilisateur infecté par une APT agressive basée sur le flux de données (Résolution : 1s)

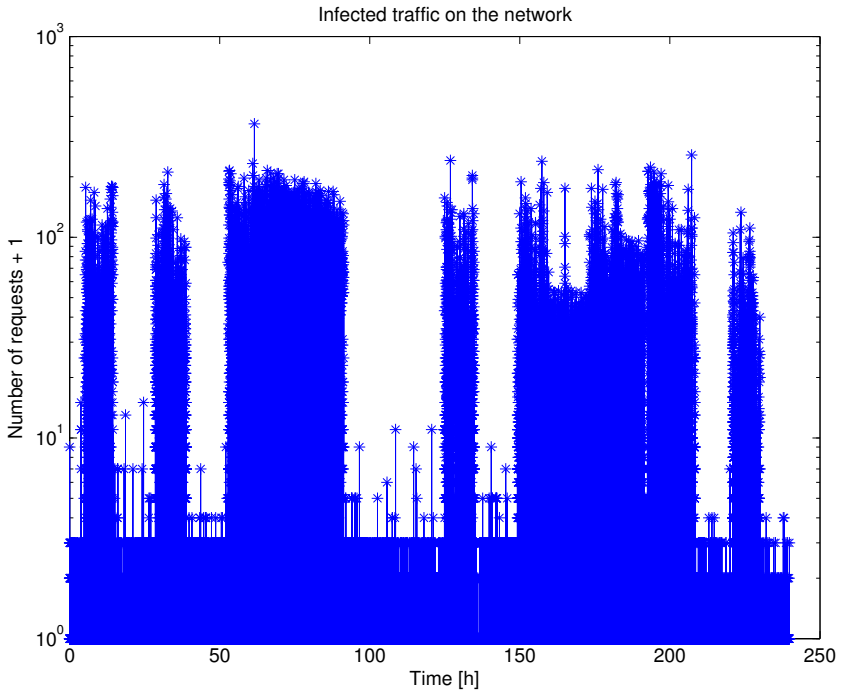


FIGURE 5.6 – Trafic infecté du sous-réseau choisi (Résolution : 1s)

temporelle entre deux requêtes que de 1 seconde. La durée totale du burst de données est, quant à elle, restée constante : 10 secondes. Pour les mêmes raisons de dissimulation, le délai reste également inchangé : 5 secondes. Le nombre d'injections pour cette APT est limité à 10 et la distance temporelle entre deux injections est, elle, diminuée à 1h. Le nombre total de sorties pour cette infection est de 37. Cette APT sera référencée comme *APT4*.

Vu l'agressivité de *APT4*, il est décidé de ne sélectionner que les bursts assurant une dissimulation maximale. Ceci explique le changement de distance temporelle entre les requêtes de *APT3* et *APT4*.

Le trafic complet du sous-réseau après infection est donné à la figure 5.6. À première vue, il n'est pas possible d'observer que ce trafic est infecté et la comparaison avec la figure 5.1 ne permet pas non plus d'identifier d'infections.

5.4 Étude des paramètres

Cette section a pour objectif de rechercher une combinaison optimale de paramètres. Cette combinaison doit permettre de placer les domaines injectés le plus haut possible dans le classement produit par le Server. La méthode choisie pour parvenir à identifier cette combinaison se base sur la maximalisation de l'AUC (Section 4.4.3).

Cette méthode commence l'étude avec une combinaison arbitraire (mais sensée) de paramètres et fait varier chacun d'eux successivement, dans l'ordre dans lequel ils apparaissent dans l'algorithme. Ci-après sont donnés les paramètres étudiés ainsi que les valeurs choisies a priori pour commencer l'étude.

Valeur de k - Le nombre de voisins dans les k-NN graphes initiaux est la première valeur balayée. Il n'est donc pas nécessaire de choisir une valeur initiale.

Poids de fusion des similarités - Afin de ne pas favoriser une des similarités étudiées par rapport à l'autre, les poids de fusion des similarités sont choisis équivalents. Ainsi, les poids sont fixés à 0,5 pour chacune des deux similarités.

Seuil de pruning - Le seuil de pruning est choisi égal à la moyenne, c'est-à-dire égal à un z-score de 0,0. Ce choix est guidé par la volonté de couper les liens faibles et moyens, tout en conservant les liens forts. Les liens forts sont supposés être les liens responsables de la modélisation des sites.

Taille maximale des clusters - Par défaut, il est souhaité de ne pas filtrer la taille des clusters. La taille maximale est donc fixée à une valeur très grande par rapport à la taille des clusters du graphe (ici, la valeur de 1 000 000 est choisie). Ceci permet d'observer dans le résultat final les domaines injectés, et ce, quelle que soit leur position dans le graphe de clusters.

Nombre minimal de requêtes par domaine et par utilisateur - Le nombre minimal de requêtes par domaine et par utilisateur n'est pas limité pour des raisons similaires à ce qui a été expliqué au point précédent. Le nombre minimal de requêtes est donc choisi égal à 1.

Poids des indices du classement - Les poids des indices sont choisis avec des valeurs presque égales. Toutefois, une légère préférence est donnée aux poids associés aux indices liés au nombre de liens vers des noeuds parents et enfants. Ce choix se justifie par la description du phénomène physique effectuée à la section 2.2. Cette description insiste sur l'isolation des APT grâce au pruning. La valeur de 0,35 est donc choisie pour les poids des indices liés au nombre de parents et d'enfants. Une valeur de 0,3 est choisie pour le poids de l'indice lié au nombre de requêtes.

Chacun des paramètres va maintenant être étudié. Sur base des résultats obtenus pour les paramètres précédents, les paramètres suivants sont étudiés. L'objectif final est de déterminer une combinaison optimale de paramètres.

5.4.1 Valeur de k

Le premier paramètre étudié est le nombre de voisins dans les k-NN graphes initiaux. Ce paramètre n'est pas un paramètre du Server, mais bien un paramètre du Batch Processor. C'est un choix qui doit donc être fait a priori. Vu que ce choix doit être fait avant le préprocessing, il est souhaité que la valeur choisie soit la plus compatible possible avec les différentes APT décrites à la section 5.3.

Un résumé de la configuration utilisée pour le balayage est donné ci-après [1] :

Valeur de k	[1 : 10 : 50 ; 100]	Clusters z-score	False
Similarité : Temps	0,5	Nbr. Req. (/dom./utilisateur)	1
Similarité : Domaine	0,5	Classement : Parents	0,35
Seuil de pruning	0,0	Classement : Enfants	0,35
Pruning z-score	True	Classement : Nbr. Req.	0,3
Taille maximale des clusters	1 000 000		

Le résultat du balayage est donné à la figure 5.7. Il peut être observé qu'une grande valeur de k donne une plus grande AUC. Certains doutes pourraient être émis quant à la valeur $k = 40$, car celle-ci a une AUC similaire à celle obtenue quand $k = 100$.

1. La notation choisie pour le balayage est la suivante : $[a : b : c\,;d]$, où a indique la valeur initiale, b le pas, c la valeur d'arrêt du balayage et d une valeur supplémentaire testée.

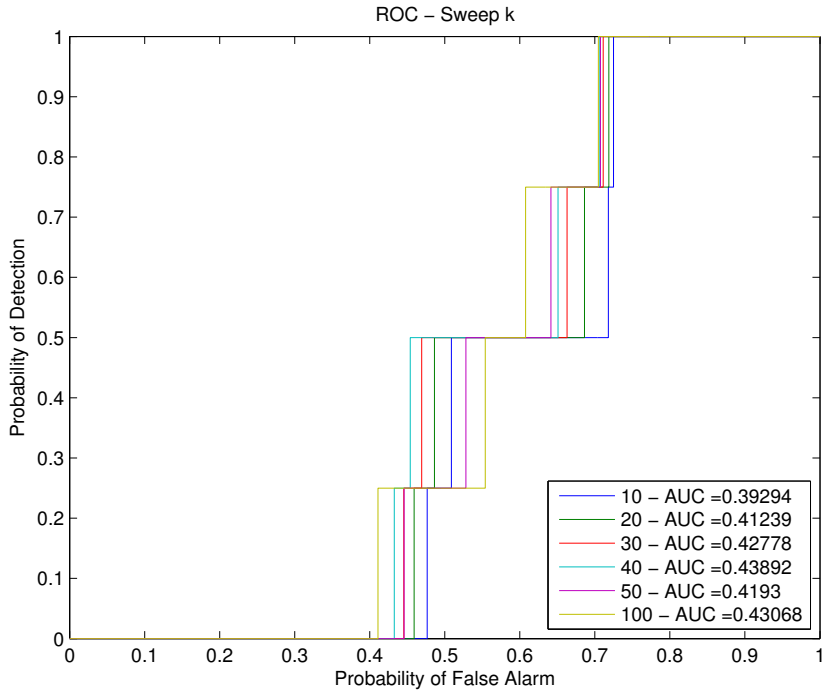


FIGURE 5.7 – Balayage du paramètre k

Étude secondaire

Pour juger de l'intérêt de la valeur de $k = 40$ par rapport à celle de $k = 100$, des tests supplémentaires sont effectués avec d'autres valeurs de paramètres que ceux utilisés jusqu'à présent. Ces valeurs sont en fait plus proches de l'ensemble de paramètres qui sera effectivement choisi grâce à l'étude effectuée dans ce chapitre. Les paramètres utilisés sont les suivants :

	1	2	3	4	5	6	7	8	9	10
Valeur de k	40	100	40	100	40	100	40	100	40	100
Similarité : Temps	0,1	0,1	0,4	0,4	0,1	0,1	0,1	0,1	0,1	0,1
Similarité : Domaine	0,9	0,9	0,6	0,6	0,9	0,9	0,9	0,9	0,9	0,9
Seuil de pruning	0,0	0,0	0,0	0,0	-0,1	-0,1	-0,5	-0,5	0,0	0,0
Pruning z-score	True									
Taille maximale des clusters	1 000 000									
Clusters z-score	False									
Nbr. Req. (/dom./utilisateur)	5									
Classement : Parents	0,4	0,4	0,4	0,4	0,4	0,4	0,4	0,4	0,5	0,5
Classement : Enfants	0,4	0,4	0,4	0,4	0,4	0,4	0,4	0,4	0,5	0,5
Classement : Nbr. Req.	0,2	0,2	0,2	0,2	0,2	0,2	0,2	0,2	0,0	0,0

Les résultats sont donnés à la figure 5.8 et il est conclu que la valeur de $k = 100$ donne de plus grandes AUC que celle de $k = 40$.

Implications du choix de la valeur du k

Le choix de la valeur du k a d'autres implications que celles liées à la détection. Ce choix influence également l'espace mémoire nécessaire au stockage des informations et la durée du préprocessing (Section 3.2).

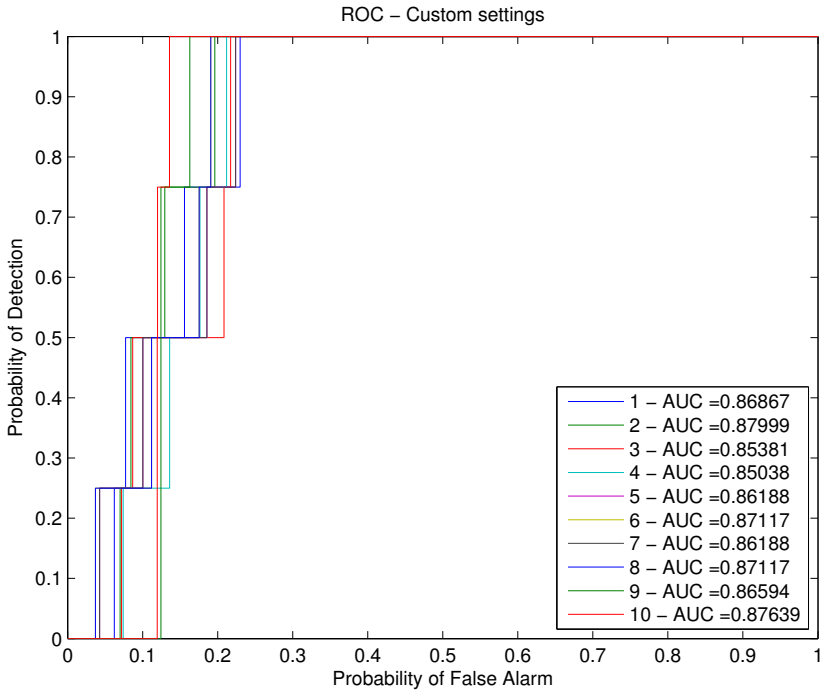


FIGURE 5.8 – Étude secondaire du paramètre k pour différents ensembles de paramètres

Concernant l'espace mémoire nécessaire, deux types d'information sont stockés : les noeuds et les liens. Les noeuds doivent de toute façon être enregistrés et constituent une charge mémoire fixe. Mais, le nombre de liens à stocker varie lui proportionnellement à k. La taille du dossier contenant les graphes a été mesurée et une évolution linéaire est observée (Fig.5.9). Les noeuds du graphe sont des domaines et chacun des domaines est une liste de requêtes. Il a été mesuré qu'une requête correspond à environ 200 Bytes. L'ordonnée à l'origine est donc interprétée comme la charge mémoire fixe nécessaire au stockage des noeuds (721 921 requêtes, c'est-à-dire environ 144 MB). À cause de la sélection des requêtes enfants et de la construction de graphes de domaines, il est difficile de prévoir le nombre de liens à stocker, et donc d'estimer la valeur du coefficient angulaire.

La durée d'exécution a également été mesurée (Fig.5.10). Pour rappel, le Batch Processor calcule un k-NN graphe par similarité et par utilisateur. Pour chaque graphe, la durée d'exécution est influencée par le nombre de requêtes de l'utilisateur. Il est observé expérimentalement que l'évolution de la durée d'exécution en fonction de k est plus quadratique que linéaire. Cette évolution non linéaire a déjà été observée par ailleurs [18]. Au vu de ces résultats, il est donc clair que le temps d'exécution est le paramètre limitant pour l'augmentation du k. En effet, le stockage de graphes dont la taille ne dépasse pas 200 MB n'est certainement pas un problème. Par contre, un préprocessing dépassant 24h peut, lui, commencer à être problématique.

Interprétation

Le fait qu'une augmentation de la valeur de k donne de meilleurs résultats peut être interprété comme suit (Section 3.7.1). Cette augmentation offre une meilleure approximation des k-NN graphes par l'algorithme NN-Descent [18]. De plus, elle permet de récolter beaucoup plus de liens faibles en faisant tendre les graphes vers des graphes complets. Ces liens faibles permettent de tisser des liens avec de nouveaux noeuds et permettent également de renforcer les liens existants. Les nouveaux liens tissés ne sont pas un problème en soi : leur similarité est faible, et ceux-ci

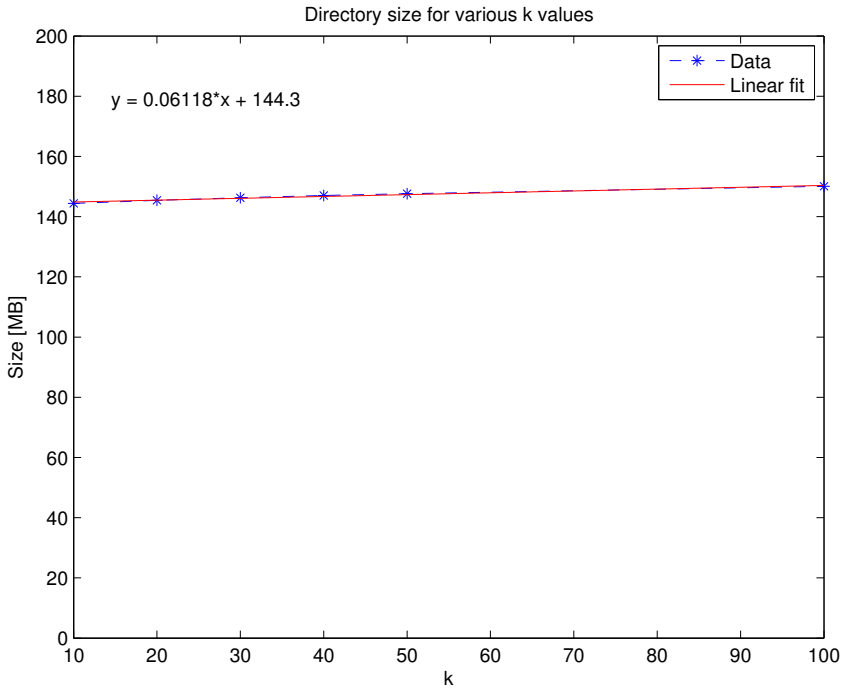


FIGURE 5.9 – Taille du dossier contenant les graphes du sous-réseau étudié pour différents k

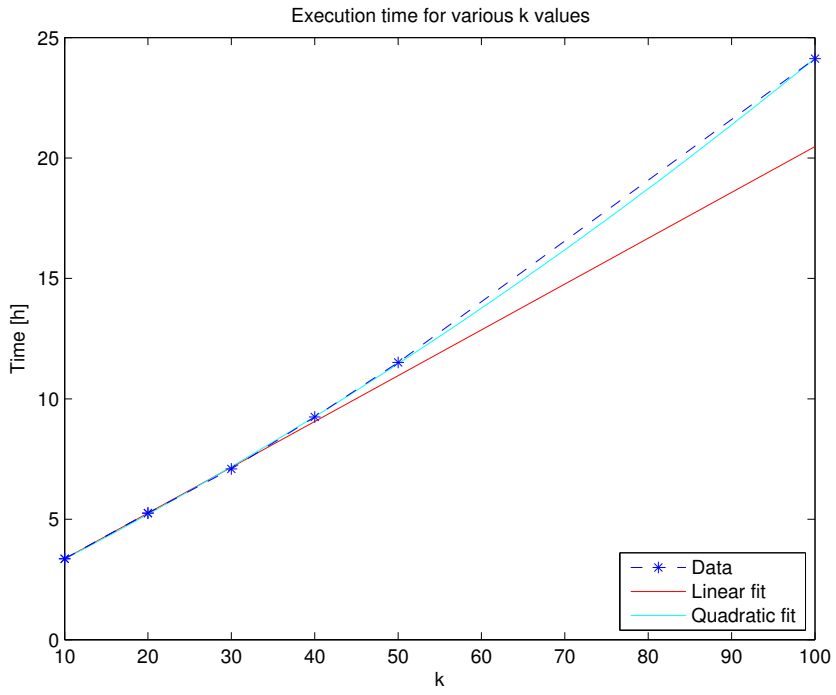


FIGURE 5.10 – Temps d'exécution pour différents k

sont coupés lors du pruning. Toutefois, les liens renforcés assurent eux une meilleure consistance de la modélisation d'un site. Le fait de renforcer la consistance des liens pour un même site permet de diminuer le nombre de domaines isolés. L'augmentation de la valeur de k permet donc de contribuer à limiter le taux de fausses alarmes. Ces dernières conclusions sont totalement en lignes avec la section 2.4.1 traitant de la capacité de l'algorithme à modéliser des pages chargées simultanément.

Sachant que les similarités des liens de l'APT n'ont comme contribution que celles provenant de la similarité temporelle, il est clair que l'augmentation de k ne rendra pas la tâche plus difficile pour isoler l'APT. En effet, les voisins rajoutés sont de plus en plus loin temporellement et engendrent donc l'ajout de très faibles liens. Par contre, cette augmentation permettra de relier d'autres domaines a priori plus éloignés temporellement, mais appartenant à la même page web. Ceci leur évite donc l'isolement. Il est clair qu'il y a une plus-value à augmenter le k. Il faudra toutefois trouver un compromis entre les grandes valeurs voulues et des valeurs raisonnables permettant le calcul des graphes. Pour la suite de l'étude, une valeur de $k = 100$ est choisie.

5.4.2 Poids de fusion des similarités

Après le premier balayage, il est décidé de travailler avec un $k = 100$ pour la suite de l'étude. Les paramètres suivants sont choisis pour l'étude des poids de fusion des similarités :

Valeur de k	**100**	Clusters z-score	False
Similarité : Temps	[0,0 : 0,1 : 1,0]	Nbr. Req. (/dom./utilisateur)	1
Similarité : Domaine	[1,0 : -0,1 : 0,0]	Classement : Parents	0,35
Seuil de pruning	0,0	Classement : Enfants	0,35
Pruning z-score	True	Classement : Nbr. Req.	0,3
Taille maximale des clusters	1 000 000		

Le résultat du balayage des poids de fusion des similarités est présenté à la figure 5.11 et fait ressortir une tendance claire en matière de détection. L'AUC augmente pour un poids décroissant de la similarité temporelle. L'AUC est maximale avec un poids de 0,1 pour cette similarité. Il est donc conclu que la contribution temporelle est nécessaire, mais que le poids doit être maintenu relativement petit par rapport au poids de la similarité basée sur les noms de domaine. Cette dernière similarité semble, quant à elle, apporter une contribution claire à la modélisation du trafic. Un poids de 0,1 est choisi pour la similarité temporelle et un poids de 0,9 pour la similarité basée sur les noms de domaine.

Avec ces poids, les APT sont détectées dans l'ordre suivant : *APT2*, *APT3*, *APT4*, et finalement, *APT1*. Il peut être constaté que les APT discrètes sont détectées en priorité. Sur la figure 5.11, il peut être observé que, avec ces poids, les deux APT discrètes sont détectées pour un taux de fausses alarmes largement plus faible que les APT agressives.

5.4.3 Seuil de pruning

Les paramètres suivants sont choisis pour l'étude du seuil de pruning :

Valeur de k	**100**	Clusters z-score	False
Similarité : Temps	**0,1**	Nbr. Req. (/dom./utilisateur)	1
Similarité : Domaine	**0,9**	Classement : Parents	0,35
Seuil de pruning	[-0,5 : 0,1 : 1,0]	Classement : Enfants	0,35
Pruning z-score	True	Classement : Nbr. Req.	0,3
Taille maximale des clusters	1 000 000		

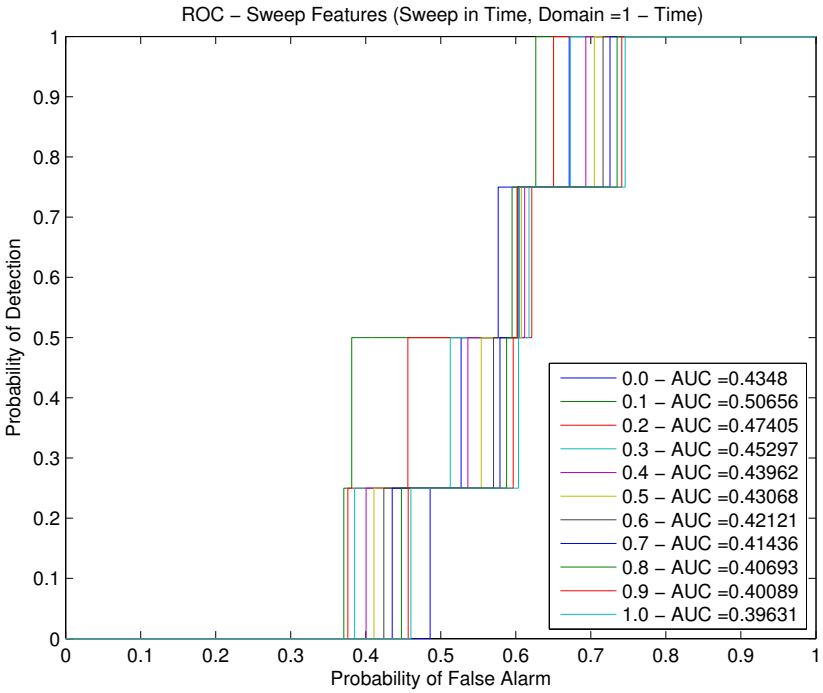


FIGURE 5.11 – Balayage des poids de fusion des similarités

Il a été observé, au moyen de l'histogramme des similarités, que la distribution des liens entre les domaines prend toujours la même forme. Pour l'ensemble de paramètres choisis, la distribution des liens est donnée à la figure 5.12. Un grand nombre de liens sont inférieurs à la moyenne. Ces liens faibles n'ont pas de plus value dans la modélisation du log. Des liens forts et très forts existent également, et permettent la création de structures dans le graphe.

Il aurait pu être attendu de retrouver des pics dans le graphe. En effet, il a été expliqué à la section 3.3 que les similarités prennent des valeurs discrètes. Ces valeurs discrètes se combinent avec une diversité telle que les pics ne ressortent pas. Toutefois, en zoomant suffisamment, des pics peuvent être observés (Fig.5.13), sans qu'aucun sens ne puisse leur être attribué.

La figure 5.14 donne les résultats du balayage. Il peut être observé que l'AUC augmente pour des seuils décroissants. Il semble qu'il ne soit plus possible d'améliorer l'AUC une fois le seuil inférieur à la moyenne. Il est tentant de choisir un seuil inférieur à la moyenne pour la suite de l'étude. Toutefois, une étude plus attentive des résultats est nécessaire.

La figure 5.15 illustre l'intérêt de ne pas choisir un seuil de pruning inférieur à la moyenne, malgré une valeur d'AUC plus importante. En effet, il est choisi de favoriser les APT basées sur le flux de données par rapport aux APT périodiques. D'autres méthodes beaucoup plus efficaces peuvent être mises en oeuvre pour détecter ces dernières (par exemple, l'étude fréquentielle du log [36]).

Lorsque le seuil de pruning choisi est inférieur à la moyenne, l'ordre de détection des APT est le suivant : *APT2*, *APT3*, *APT1*, et finalement *APT4*. Or, pour un seuil égal à la moyenne, l'ordre de détection est le suivant : *APT2*, *APT3*, *APT4*, et finalement, *APT1*. Les probabilités de fausses alarmes suivantes peuvent être associées à la détection de chacune des APT :

p_{fa}	Pruning = -0,1	Pruning = 0,0
APT1	0,61	0,63
APT2	0,24	0,37
APT3	0,44	0,38
APT4	0,61	0,59

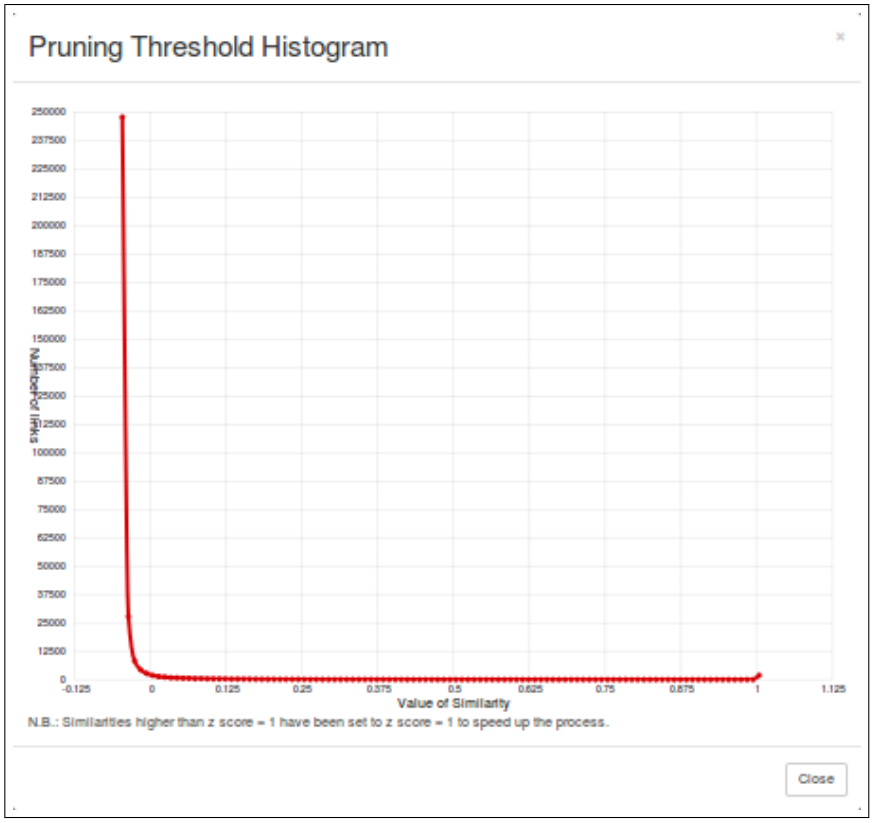


FIGURE 5.12 – Histogramme de la distribution des similarités entre les domaines du graphe (Seuil de pruning en z-score)

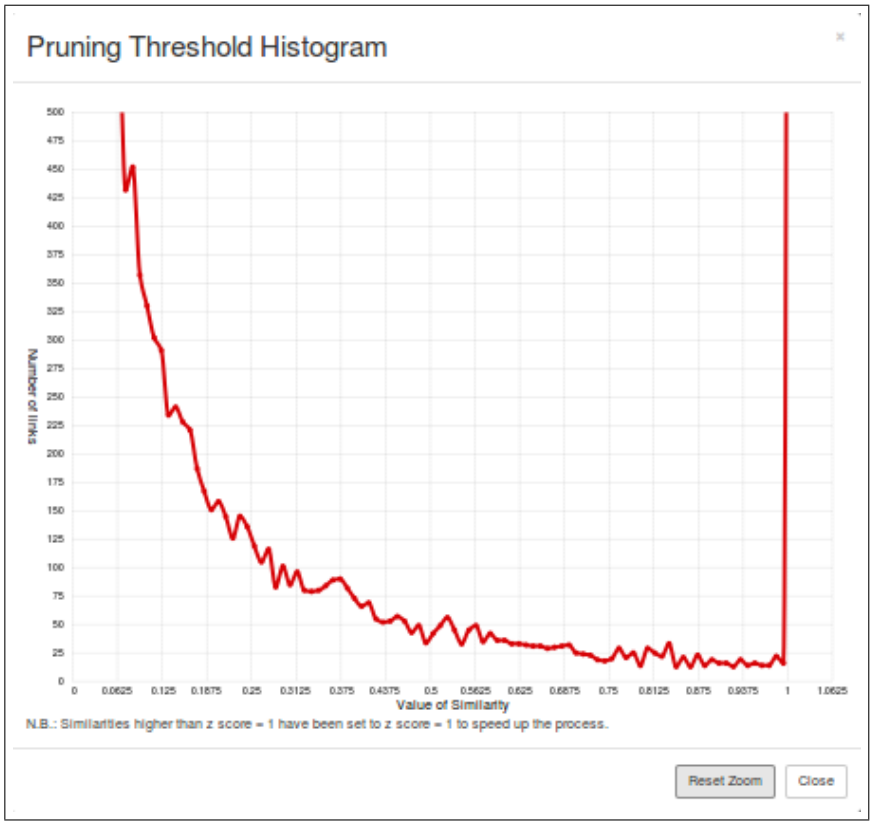


FIGURE 5.13 – Zoom sur les pics observables de l'histogramme des similarités entre les domaines du graphe

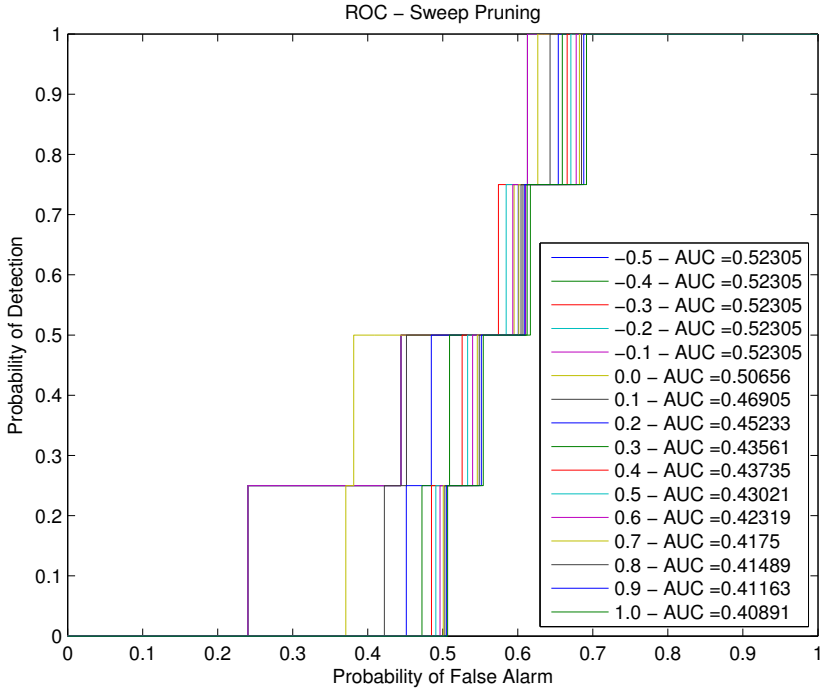


FIGURE 5.14 – Balayage du seuil de pruning

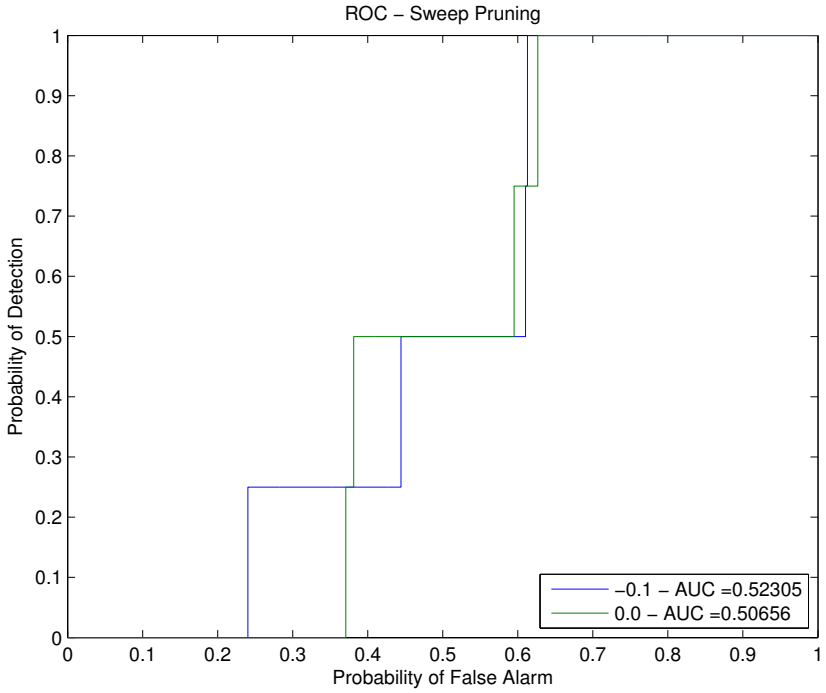


FIGURE 5.15 – Comparaison du pruning à la moyenne des similarités et à une valeur légèrement en dessous

Lorsque le seuil de pruning passe de -0,1 à 0,0, il peut être observé que la probabilité de fausse alarme augmente pour les APT périodiques et diminue pour les APT basées sur le flux des données. En fonction du type d'APT ciblée, il est donc conseillé de choisir un seuil égal ou inférieur à la moyenne des similarités.

Ce constat peut être interprété en se basant sur les histogrammes de trafic présentés à la section 5.3. Les APT périodiques ont une tendance naturelle à complètement s'isoler lorsque celles-ci font des requêtes hors du trafic de l'utilisateur. Les liens temporels qu'elles tissent alors avec d'autres domaines sont très faibles. Ceci entraîne leur isolement dès qu'un léger pruning est appliqué. Ce n'est pas le cas des APT basées sur le flux des données. Ces dernières sont, par définition, dissimulées dans le trafic et tissent des liens temporels avec d'autres domaines.

Pour détecter des APT, les liens faibles et moyens doivent donc être supprimés. Ceci ne laisse que les liens forts dans le graphe. Ces liens forts naissent par répétition de liens, soit pour un même utilisateur, soit pour plusieurs utilisateurs. Cette répétition assure donc la validité du lien. Il peut finalement être noté que l'APT ne peut pas tisser de liens au moyen de la similarité basée sur les noms de domaine puisque celle-ci a toujours un nom de domaine qui lui est entièrement propre.

Pour la suite de l'étude, et afin de favoriser les APT basées sur le flux des données, il est choisi de fixer le seuil de pruning égal à la moyenne des similarités. Il est finalement conclu que le seuil de pruning est un paramètre crucial. Celui influence considérablement le taux de fausse alarme et permet de cibler un certain type d'APT.

5.4.4 Taille maximale des clusters

Les paramètres suivants sont choisis pour l'étude de la taille maximale des clusters :

Valeur de k	**100**	Clusters z-score	False
Similarité : Temps	**0,1**	Nbr. Req. (/dom./utilisateur)	1
Similarité : Domaine	**0,9**	Classement : Parents	0,35
Seuil de pruning	**0,0**	Classement : Enfants	0,35
Pruning z-score	**True**	Classement : Nbr. Req.	0,3
Taille maximale des clusters	[1 : 10 : 100 ; 1 000 000]		

Il peut être observé sur la figure 5.16 que le fait de filtrer les clusters en se basant sur leur taille entraîne une détection partielle des APT. Les APT détectées lorsqu'un filtrage est appliqué sont les APT périodiques (*APT1* & *APT2*). Lorsqu'aucun filtrage n'est effectué (taille maximale des clusters fixée à une valeur très grande par rapport aux tailles observées de clusters), toutes les APT sont détectées, et elles le sont dans l'ordre *APT2*, *APT3*, *APT4* et *APT1*.

Ce résultat peut être compris en observant l'histogramme de la distribution des tailles de clusters (Fig.5.17). Il peut être observé que l'algorithme crée un grand nombre de clusters très petits (souvent, des domaines isolés) et un grand nombre de clusters très grands. Il est donc très difficile d'obtenir une gradation dans le filtrage effectué. Une origine possible de ce phénomène pourrait être la problématique liée à la discrimination des requêtes parentes et enfants (Section 3.7.1).

Les APT périodiques sont souvent isolées très facilement (voir section 5.4.3). Elles apparaissent donc systématiquement dans le lot des clusters de très petite taille (souvent, des clusters ne contenant qu'un domaine). Par contre, les APT basées sur le flux des données sont, elles, forcément connectées à des structures fortement liées et modélisant des sites. Il y a donc une forte probabilité qu'elles soient incluses dans un cluster, malgré leur faible lien. En filtrant les gros clusters, il y a donc un risque de filtrer également ce type d'APT. Ce risque a été annoncé lors de la critique du processing (Section 3.7.2).

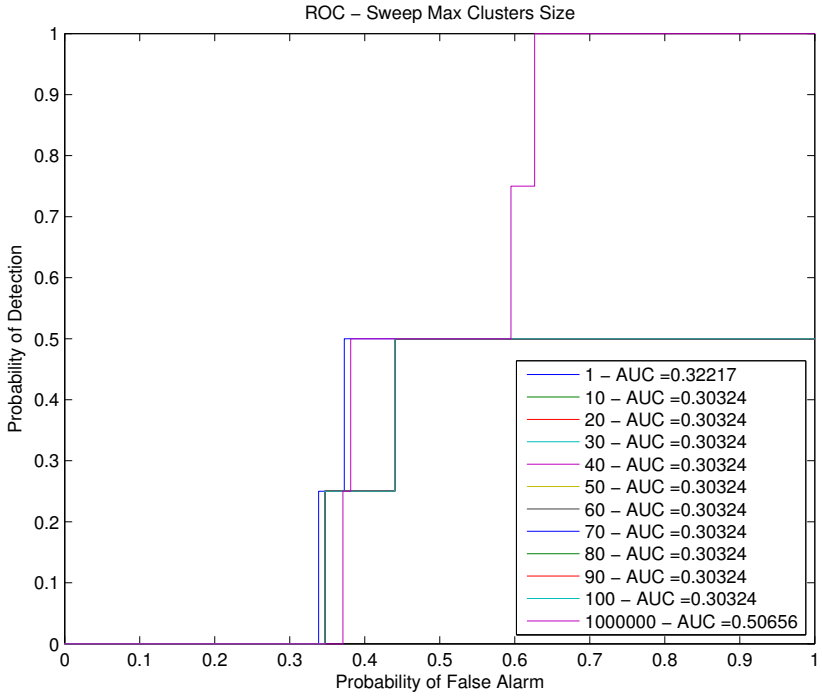


FIGURE 5.16 – Balayage de la taille maximale des clusters

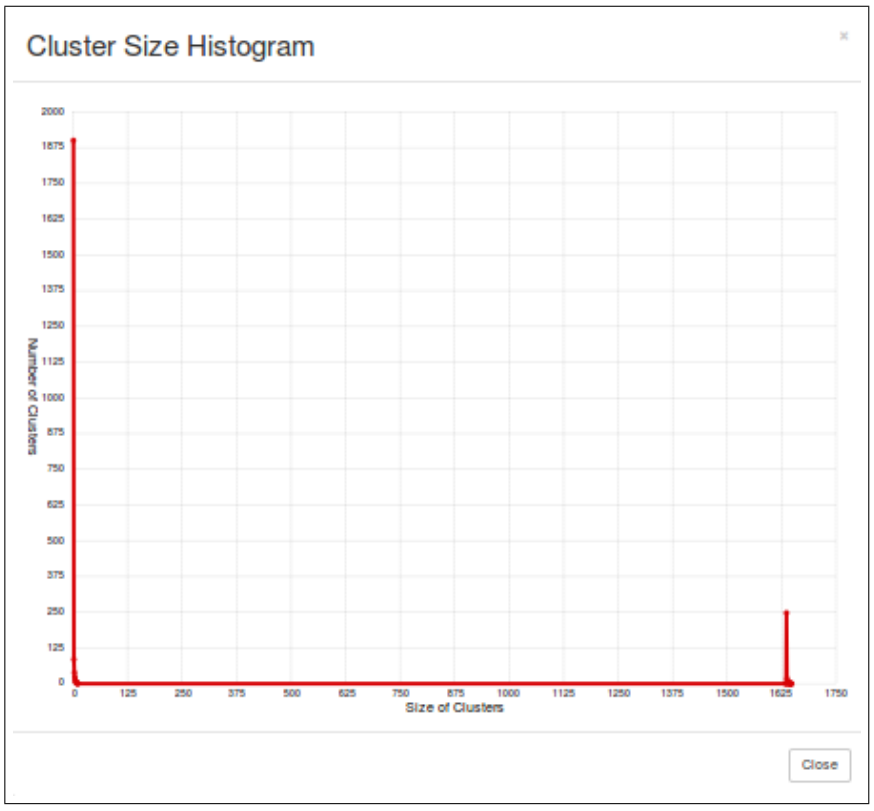


FIGURE 5.17 – Histogramme de la distribution des tailles de clusters

Bien que ce choix puisse sembler étonnant au premier abord, il semble donc intéressant de ne pas filtrer sur la taille des clusters. Ce choix implique deux conséquences pratiques. Premièrement, une grande part du temps de calcul est consacrée au clustering durant le processing des données au niveau du Server. De plus, une fois les clusters produits, la suite de l'algorithme doit les parcourir un par un, à chaque opération. Or, aucune plus-value n'est retirée de ces clusters. Il semble donc raisonnable d'envisager de modifier le coeur même du fonctionnement du Server (Section 6.2.1). Une autre solution serait d'améliorer le clustering afin de rentabiliser le temps de calcul consacré. Ce clustering pourrait être modifié pour éviter l'inclusion de ce type de domaines faiblement liés dans les clusters.

Un deuxième problème a été identifié au niveau de l'interface graphique lorsque le filtrage n'est pas effectué. L'interface n'est clairement pas assez robuste que pour être capable d'afficher l'entièreté des résultats modélisant un log de plusieurs jours, avec de nombreux utilisateurs. Lorsqu'il est demandé d'afficher de tels graphes, le JavaScript de l'interface a tendance à planter à cause de la quantité d'informations à afficher. Toutefois, rendre l'interface plus robuste n'est pas suffisant pour régler le problème. En effet, lorsque le graphe devient trop grand, celui-ci devient très difficile à consulter (Fig.5.18). La consultation est difficile, car la densité d'information affichée est très importante. Il faudrait donc travailler sur l'interface pour le rendre plus robuste, mais également plus explicite (Section 6.2.2). Sans ces mises à jour, l'affichage du graphe n'a pas de plus-value.

L'objectif étant de cibler en priorité les APT basées sur le flux de données, il est choisi de ne pas filtrer sur la taille des clusters en imposant une taille maximale de cluster égale à 1 000 000. Les limitations liées à l'interface ne posent pas de problèmes pour la suite de l'étude. En effet, le Server est capable de ne renvoyer que le standard output sur l'interface graphique, et ce, sans les données du graphe. Pour ce faire, il faut utiliser l'option **-study** du Server.

5.4.5 Nombre minimal de requêtes par domaine et par utilisateur

Les paramètres suivants sont choisis pour l'étude du nombre minimal de requêtes par domaine et par utilisateur :

Valeur de k	**100**	Clusters z-score	**False**
Similarité : Temps	**0,1**	Nbr. Req. (/dom./utilisateur)	[1 : 1 : 5 ; 10 ; 15 ; 20]
Similarité : Domaine	**0,9**	Classement : Parents	0,35
Seuil de pruning	**0,0**	Classement : Enfants	0,35
Pruning z-score	**True**	Classement : Nbr. Req.	0,3
Taille maximale des clusters	**1 000 000**		

Le résultat du balayage est présenté à la figure 5.19. Il peut être observé que ce paramètre permet de réguler de manière simple et efficace le taux de fausses alarmes. Il est également constaté qu'il y a une grosse amélioration de l'AUC en augmentant la valeur de ce paramètre. Il semble qu'il soit capable de sensiblement translater la ROC vers la gauche. Il serait tentant de le faire augmenter outre mesure et il est important de rester conscient de sa signification physique.

Le choix de la valeur de ce paramètre doit être fait en ayant une idée de ce que l'on cherche. En connaissant l'agressivité des APT recherchées, le nombre de requêtes par utilisateur et par domaine sur la période étudiée peut être estimé. Il est important de choisir un nombre qui ne soit pas trop élevé, car le risque d'éliminer l'APT est bien présent. Dans le scénario, le nombre de requêtes pour *APT3* est de 13. Il peut être observé que, pour les ROC associées à un paramètre supérieur à 13, l'AUC a largement chuté. Ceci s'explique par le fait que l'*APT3* n'est plus détectée. Un deuxième saut dans l'AUC se déroule à 19, lorsque *APT2* n'est plus détectée.

Il est important de bien comprendre les implications de ce paramètre. Si, dans un scénario donné, une même APT infecte deux utilisateurs, mais que le nombre de requêtes de l'APT pour ces deux

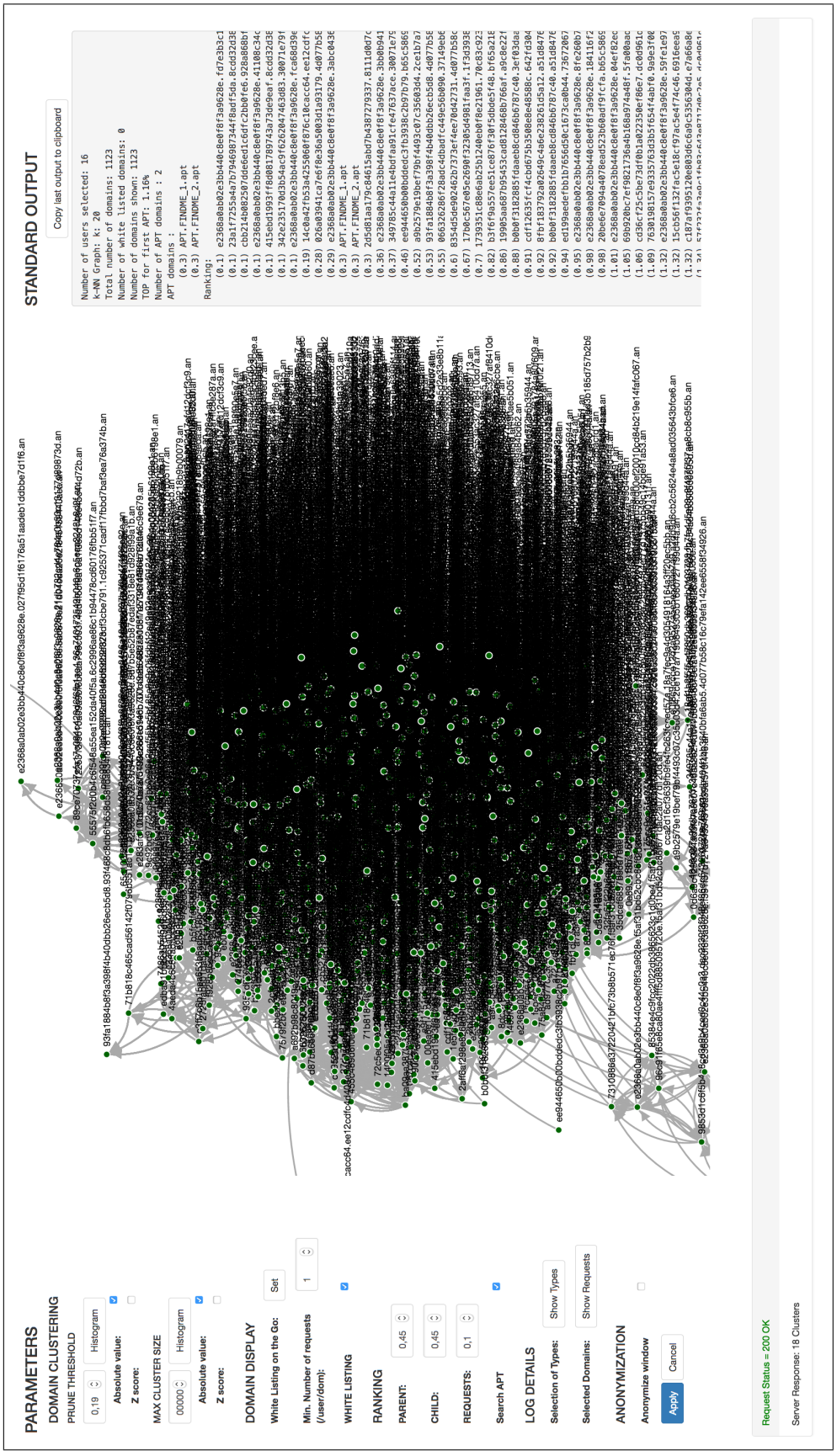


FIGURE 5.18 – Illustration de la difficulté de la consultation d'un graphe trop fourni

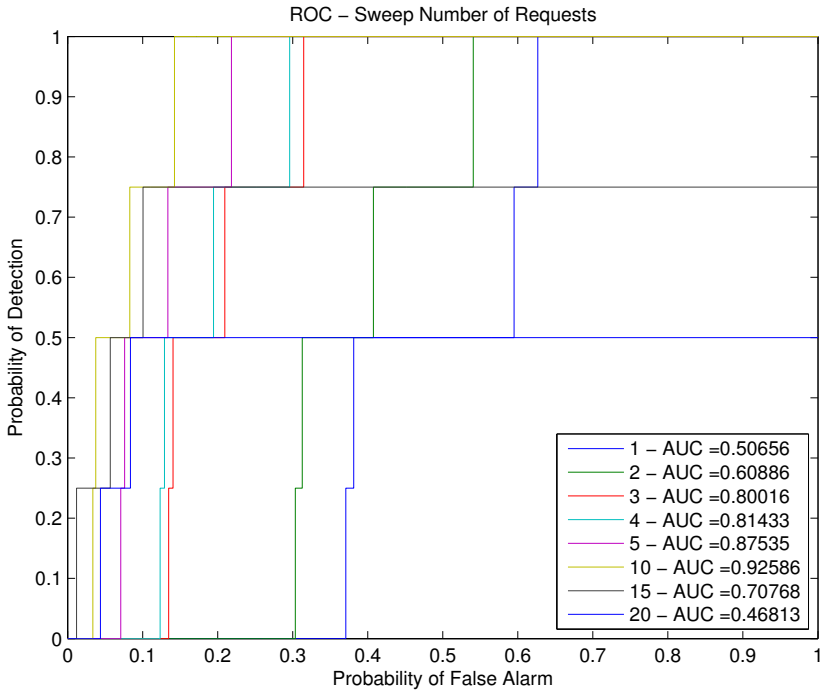


FIGURE 5.19 – Balayage du nombre minimum de requêtes par domaine et par utilisateur

utilisateurs est différent, il est possible que le domaine C2 soit éliminé pour ces deux utilisateurs. En effet, si le seuil est choisi entre le nombre de sorties de l'APT chez les deux utilisateurs, l'algorithme supprime complètement le domaine C2 du graphe (malgré le nombre de sorties supérieur au seuil choisi pour l'un des deux utilisateurs). Il est à noter que ce comportement est souhaité. En effet, il est attendu que l'analyste choisisse un seuil en prenant en compte le cas le plus défavorable. Cette contrainte est compensée par la capacité à nettoyer le résultat des domaines pouvant être très occasionnels (ex. : les domaines publicitaires).

Sans vouloir être trop spécifique, le choix d'une valeur de 5 pour le paramètre semble être un choix avisé. Ceci permet l'élimination d'énormément de bruit composé de domaines très peu fréquentés et qui ne semblent pas pouvoir constituer un C2 fiable. Pour cette valeur du paramètre, les APT sont détectées dans l'ordre suivant : *APT2*, *APT3*, *APT4* et *APT1*.

5.4.6 Poids des indices du classement

L'étude des poids des indices du classement est effectuée en deux parties. En effet, deux catégories peuvent être distinguées dans les indices. Dans un premier temps, le poids associé à l'indice lié au nombre de requêtes d'un domaine est étudié. Lors de cette étude, les poids des deux autres indices sont choisis égaux et tels que la somme des trois poids soit égale à 1. Dans un second temps, sur base du poids pour l'indice associé au nombre de requêtes d'un domaine, les poids associés aux indices liés au nombre de liens avec des domaines parents et avec des domaines enfants sont étudiés.

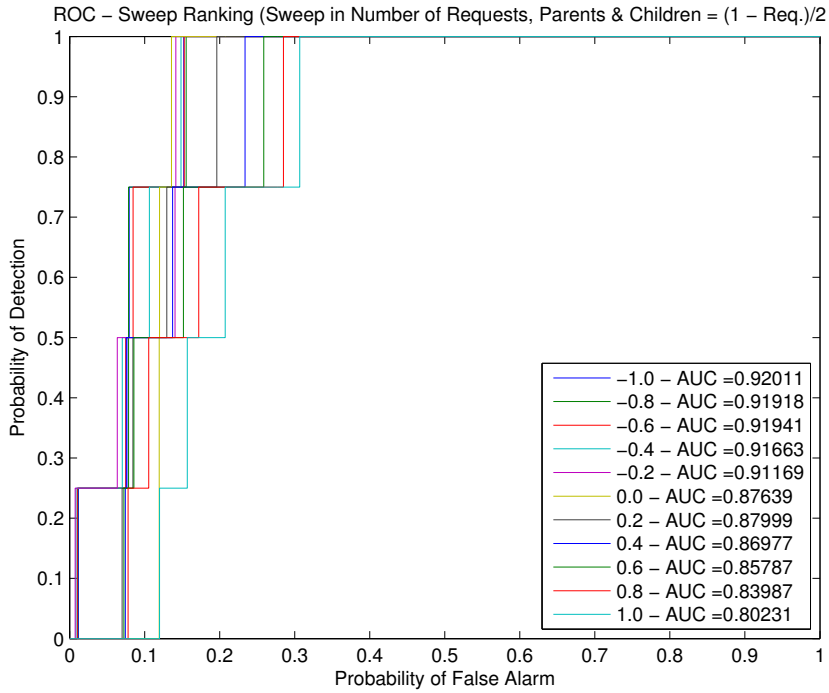


FIGURE 5.20 – Balayage du poids de l'indice lié au nombre de requêtes d'un domaine, en gardant les deux autres poids identiques et la somme des trois égale à 1

Poids de l'indice lié au nombre de requêtes d'un domaine

Les paramètres suivants sont choisis pour l'étude du poids de l'indice lié au nombre de requêtes d'un domaine :

Valeur de k	**100**	Clusters z-score	**False**
Similarité : Temps	**0,1**	Nbr. Req. (/dom./utilisateur)	**5**
Similarité : Domaine	**0,9**	Classement : Parents	$(1 - Nbr.Req.)/2$
Seuil de pruning	**0,0**	Classement : Enfants	$(1 - Nbr.Req.)/2$
Pruning z-score	**True**	Classement : Nbr. Req.	[-1,0 : 0,2 : 1,0]
Taille maximale des clusters	**1 000 000**		

Les résultats du premier balayage sont présentés à la figure 5.20. Il peut être observé qu'il y a un intérêt certain à ne pas donner une pondération trop importante à l'indice associé au nombre de requêtes d'un domaine. Il est tentant de préférer un poids inférieur à 0,0. Toutefois, une étude approfondie permet de mieux comprendre le phénomène caché derrière ces résultats.

Pour comprendre le phénomène, il faut s'intéresser à l'ordre dans lequel sont détectées les APT. Pour le poids de -0,2, cet ordre est le suivant : *APT1*, *APT2*, *APT3* et *APT4*. Pour le poids de 0,2, l'ordre est, quant à lui, le suivant : *APT2*, *APT3*, *APT4* et *APT1*. La figure 5.21 permet de visualiser l'intérêt de préférer un poids positif à un poids négatif. Les probabilités de fausses alarmes associées à chacune des détections sont données ci-après :

p_{fa}	Nbr. Req. = -0,2	Nbr. Req. = 0,2
APT1	0,01	0,20
APT2	0,06	0,07
APT3	0,14	0,08
APT4	0,14	0,13

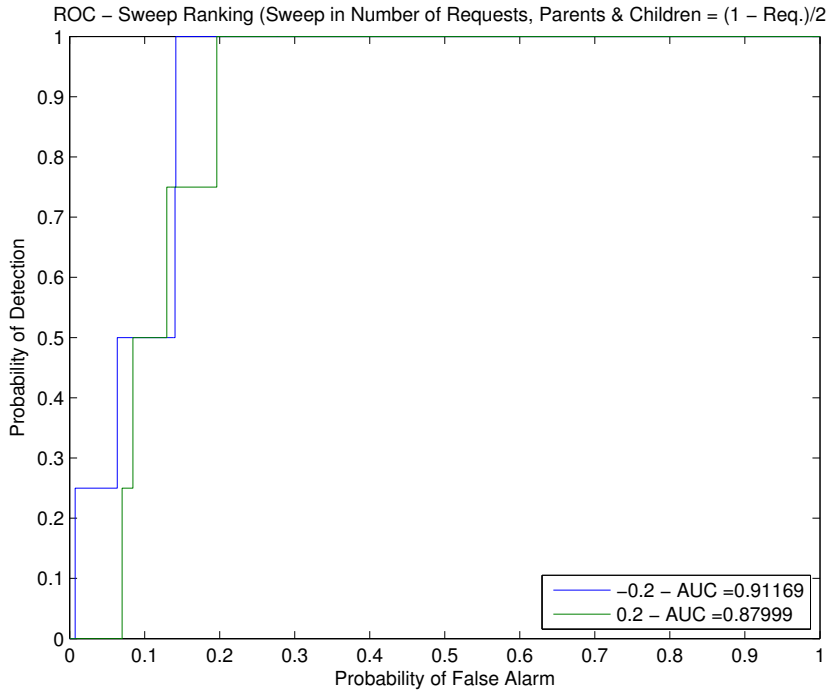


FIGURE 5.21 – Comparaison d'un poids positif et d'un poids négatif pour l'indice du nombre de requêtes d'un domaine

Il peut être observé que les APT périodiques sont associées à un taux de fausses alarmes plus important pour un poids positif que pour un poids négatif. En ce qui concerne les APT basées sur le flux de données, le phénomène inverse est observé et le taux de fausses alarmes diminue avec un poids positif. Comme déjà expliqué précédemment, le choix est fait de favoriser les APT basées sur le flux de données. Il est donc intéressant, pour faire ressortir ces APT, de favoriser un poids positif. Ainsi, il est choisi de travailler avec un poids de 0,2 pour la suite du développement.

Ces observations peuvent être interprétées de manière simple. Un poids négatif a tendance à favoriser les domaines isolés, mais ayant relativement beaucoup de requêtes. Inversement, un poids positif a lui tendance à favoriser les domaines isolés avec relativement peu de requêtes. Or, une APT périodique a tendance à effectuer plus de sorties qu'une APT basée sur les flux de données. Ceci explique donc pourquoi un poids négatif fait ressortir les APT périodiques et un poids positif fait ressortir les APT basées sur le flux de données.

Poids des indices liés au nombre de liens avec des domaines parents et enfants

Après l'étude du poids de l'indice lié au nombre de requêtes d'un domaine, il a été choisi de fixer celui-ci à 0,2. Les paramètres suivants sont alors choisis pour l'étude des poids des indices liés au nombre de liens avec des domaines parents et avec des domaines enfants :

Valeur de k	**100**	Clusters z-score	**False**
Similarité : Temps	**0,1**	Nbr. Req. (/dom./utilisateur)	**5**
Similarité : Domaine	**0,9**	Classement : Parents	[0,0 : 0,1 : 0,8]
Seuil de pruning	**0,0**	Classement : Enfants	[0,8 : -0,1 : 0,0]
Pruning z-score	**True**	Classement : Nbr. Req.	**0,2**
Taille maximale des clusters	**1 000 000**		

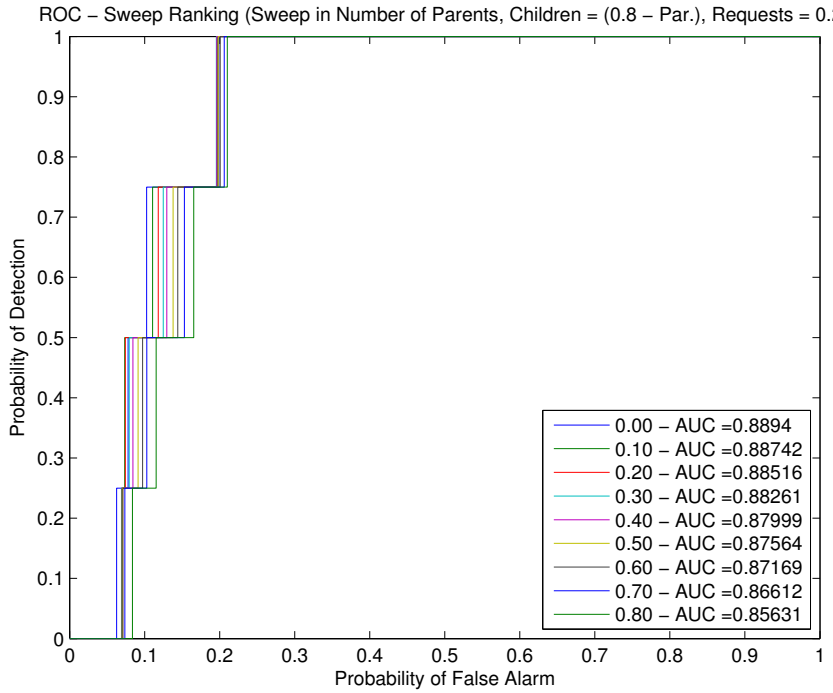


FIGURE 5.22 – Balayage des poids des indices liés au nombre de liens avec des domaines parents et avec des domaines enfants, en gardant le poids de l'indice lié au nombre de requêtes d'un domaine constant

Les résultats du balayage sont présentés à la figure 5.22. Il semble qu'il y ait une pondération optimale qui soit obtenue pour 0,0 de pondération pour l'indice lié au nombre de liens avec les domaines parents. Toutefois, ce n'est pas le choix qui sera conseillé à l'analyste. En effet, la position de l'injection de l'APT dans un burst (paramètre **-delay** de l'outil d'infection) pourrait avoir une influence importante sur les résultats. Afin de donner un ensemble de paramètres capable de cibler les APT basées sur les flux de données de manière générale, il est conseillé de choisir une pondération équivalente pour les deux indices. Une étude plus approfondie de l'influence de la position de l'APT dans un burst devrait être faite (Section 6.2.3). Pour une pondération de 0,4 pour chacun des deux poids, les APT sont détectées dans l'ordre : *APT2*, *APT3*, *APT4* et *APT1*.

5.5 Paramètres optimaux

L'étude de la section précédente a permis d'identifier un ensemble de paramètres permettant une détection maximale d'APT dans un log de proxy. Ces paramètres sont résumés ci-après :

Valeur de k	**100**	Clusters z-score	**False**
Similarité : Temps	**0,1**	Nbr. Req. (/dom./utilisateur)	**5**
Similarité : Domaine	**0,9**	Classement : Parents	**0,4**
Seuil de pruning	**0,0**	Classement : Enfants	**0,4**
Pruning z-score	**True**	Classement : Nbr. Req.	**0,2**
Taille maximale des clusters	**1 000 000**		

À ces paramètres est associé une ROC (Fig.5.23). Pour cette courbe, les APT sont détectées dans l'ordre suivant : *APT2*, *APT3*, *APT4* et *APT1*. Le temps nécessaire au Server pour obtenir

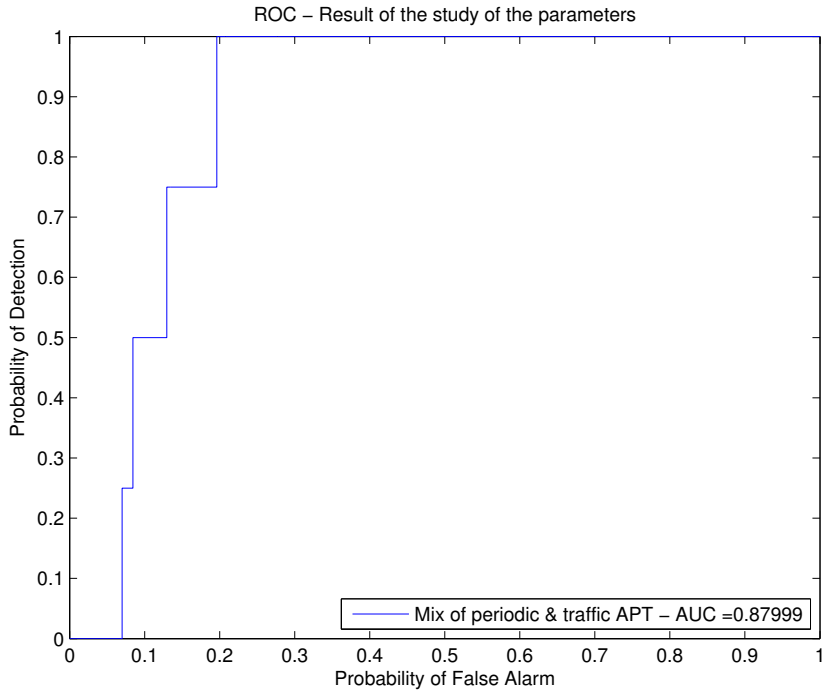


FIGURE 5.23 – ROC pour les paramètres optimaux

cette courbe est de l'ordre de grandeur de 23 minutes. Dans ces 23 minutes, 14 sont consacrées à la création des clusters. Or, il a été montré que le filtrage basé sur la taille des clusters n'est pas intéressant. Il est donc attendu de pouvoir rendre le Server encore plus interactif en supprimant le calcul de clusters lors du processing des données (Section 6.2.1).

Il est important de noter que l'algorithme obtient ces résultats en excluant bon nombre de domaines. Au total, sur les 4 310 domaines considérés, 65,3% ont été exclus (2 814 domaines). Ce chiffre met clairement en avant la notion de bruit de fond déjà évoquée précédemment. Ces domaines sont exclus par le simple fait qu'ils soient présents dans la white list ou par le fait qu'ils ne répondent pas aux critères imposés par l'analyste en ce qui concerne le nombre de requêtes par utilisateur et par domaine. Pour retrouver tous les domaines, 19,6% des domaines doivent être étudiés (849 domaines). Par contre, pour retrouver la première APT, seulement 7,0% d'entre eux doivent être vérifiés (302 domaines).

Il est finalement important de souligner que les paramètres ont été choisis de manière à rester relativement généraux. L'objectif de cette étude est bien de donner des valeurs repères pour un(e) analyste souhaitant utiliser l'algorithme. Il est clair qu'il est tout à fait possible d'ajuster les paramètres à ce cas simulé précis. Toutefois, l'objectif n'est pas de présenter un ensemble de paramètres parfaits pour ce cas-ci, mais bien une combinaison de paramètres permettant la détection d'APT dans une multitude de configurations. Parmi ces APT, les paramètres visent plus particulièrement les APT basées sur le flux de données. Les paramètres liés au nombre de requêtes (par domaine et par utilisateur) et aux poids du classement peuvent être ajustés de manière plus fine au cas simulé. Un léger gain pourrait être obtenu. Mais les paramètres utilisés perdent alors leur généralité. Il n'y a donc pas d'intérêt particulier à chercher cette combinaison optimale.

5.6 Étude spécifique d'APT basées sur le flux de données

L'étude des paramètres effectuée précédemment a permis d'identifier un ensemble de paramètres pouvant servir de valeurs par défaut lors de l'analyse d'un sous-réseau. Ces paramètres sont les valeurs par défaut de l'interface utilisateur et valent :

Valeur de k	**100**	Clusters z-score	**False**
Similarité : Temps	**0,1**	Nbr. Req. (/dom./utilisateur)	**5**
Similarité : Domaine	**0,9**	Classement : Parents	**0,4**
Seuil de pruning	**0,0**	Classement : Enfants	**0,4**
Pruning z-score	**True**	Classement : Nbr. Req.	**0,2**
Taille maximale des clusters	**1 000 000**		

Pour évaluer la capacité de cet ensemble de paramètres à détecter des APT basées sur le flux de données, un nouveau scénario est proposé. Celui-ci est simulé sur le même sous-réseau et la même période que précédemment, mais les infections sont différentes. Quatre utilisateurs sont infectés par des APT basées sur le flux de données. Ces APT ont une agressivité variable.

Pour les quatre APT injectées, le choix suivant a été fait concernant leurs caractéristiques (Code B.9). Le burst doit avoir une longueur minimale de 6 secondes et doit contenir des requêtes effectuées au moins toutes les 2 secondes. L'APT est injectée au centre des bursts, donc avec un délai de 3 secondes. Un maximum de 10 requêtes par jour est fixé. Finalement, les requêtes doivent être séparée de minimum 4h pour *APT5*, 2h pour *APT6*, 1h pour *APT7* et 30min pour *APT8*. Ce dernier paramètre permet de réguler l'agressivité de l'APT. Pour ces APT, le nombre de requêtes peut être déterminé : 9 requêtes pour *APT5*, 10 requêtes pour *APT6*, 21 requêtes pour *APT7* et, finalement, 30 requêtes pour *APT8*.

La ROC obtenue est donnée à la figure 5.24. Il peut être observé que, malgré les changements au niveau des infections, le résultat est totalement semblable à ce qui avait été obtenu après la séquence de tests (Fig.5.23). Il est également intéressant de remarquer que les APT sont détectées dans l'ordre croissant d'agressivité : *APT5*, *APT6*, *APT7* et *APT8*. Finalement, il est à noter que le temps d'exécution du Server pour l'obtention de cette courbe est de 22 minutes. Ceci est également semblable à la section 5.5. Dans ces 22 minutes, 14 sont consacrées au clustering. Comme déjà mentionné, le clustering pourrait être évité, vu qu'il n'est pas exploité.

Dans ce scénario-ci, 4 310 domaines sont étudiés et 65,3% d'entre eux exclus (2 814 domaines). Cette fois-ci, l'ensemble des APT injectées est retrouvé en étudiant 15,7% des domaines (677 domaines) et la première APT est, elle, trouvée avec 3,9% d'entre eux (168 domaines).

5.7 Étude spécifique d'APT périodiques

Sur base de l'étude effectuée précédemment, une combinaison de paramètres peut également être identifiée comme étant optimale à la détection d'APT périodiques. Bien que ce détecteur ne soit pas spécifiquement prévu pour ce type de détection, il est possible de le faire. Dans ce scénario-ci, il est choisi de mettre en évidence les APT périodiques à relativement faible période, c'est-à-dire les APT périodiques peu agressives. Les paramètres suivants sont donc utilisés :

Valeur de k	**100**	Clusters z-score	**False**
Similarité : Temps	**0,1**	Nbr. Req. (/dom./utilisateur)	**5**
Similarité : Domaine	**0,9**	Classement : Parents	**0,4**
Seuil de pruning	**-0,1**	Classement : Enfants	**0,4**
Pruning z-score	**True**	Classement : Nbr. Req.	**0,2**
Taille maximale des clusters	**1 000 000**		

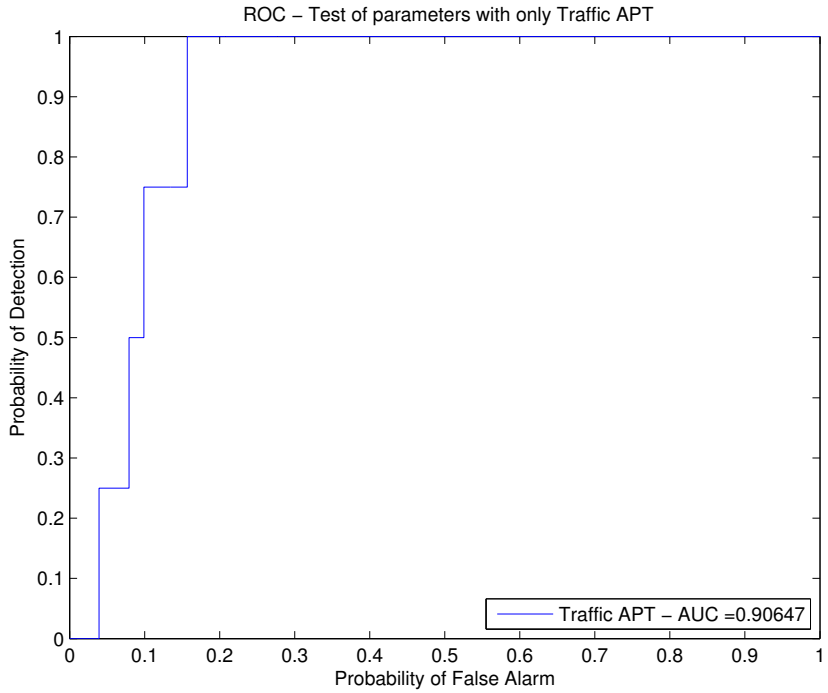


FIGURE 5.24 – ROC de l'étude spécifique d'APT basées sur le flux de données

Un scénario similaire à celui élaboré pour les APT basées sur le flux de données peut être mis en place. Le même sous-réseau et la même période sont utilisés. Quatre APT périodiques sont injectées (Code B.10). Ces dernières vont dans un ordre croissant d'agressivité. L'agressivité est ici régulée par la période d'injection. Les APT injectées ont les périodes suivantes : 24h pour *APT9*, 12h pour *APT10*, 6h pour *APT11* et 1h pour *APT12*. Pour ces APT, le nombre de requêtes peut être déterminé : 9 requêtes pour *APT9*, 19 requêtes pour *APT10*, 39 requêtes pour *APT11* et, finalement, 239 requêtes pour *APT12*.

Sur base de ce scénario, la ROC a pu être calculée et est donnée à la figure 5.25. Les résultats obtenus sont à nouveau très semblables à ceux obtenus précédemment. Comme dans le cas précédent, les APT sont détectées dans l'ordre croissant d'agressivité : *APT9*, *APT10*, *APT11* et *APT12*. Pour obtenir ce résultat, le Server a eu besoin de 9 minutes de calculs, dont 4 pour le clustering. Ce temps pourrait être amélioré en supprimant le clustering.

Il pourrait sembler étonnant qu'il y ait une si grande différence entre le temps d'exécution pour le scénario avec uniquement des APT périodiques (9 min) et le scénario avec uniquement des APT basées sur le flux de données (22 min). Ceci s'explique par la valeur du seuil de pruning. Le fait de couper moins de liens entraîne un clustering plus simple (608 secondes plus rapide). Les clusters ainsi formés sont beaucoup plus gros. Il y a donc moins de clusters à parcourir dans la suite de l'algorithme. Un gain de 157 secondes est à noter pour le white listing, et de 5 secondes pour générer le classement.

Dans ce scénario-ci, il est également question de 4 310 domaines étudiés et de 65,3% d'entre eux exclus (2 814 domaines). Pour retrouver l'ensemble des APT injectées, il est nécessaire d'étudier 18,8% des domaines (810 domaines). Toutefois, pour retrouver le premier d'entre eux, l'étude de 0,8% suffit (35 domaines).

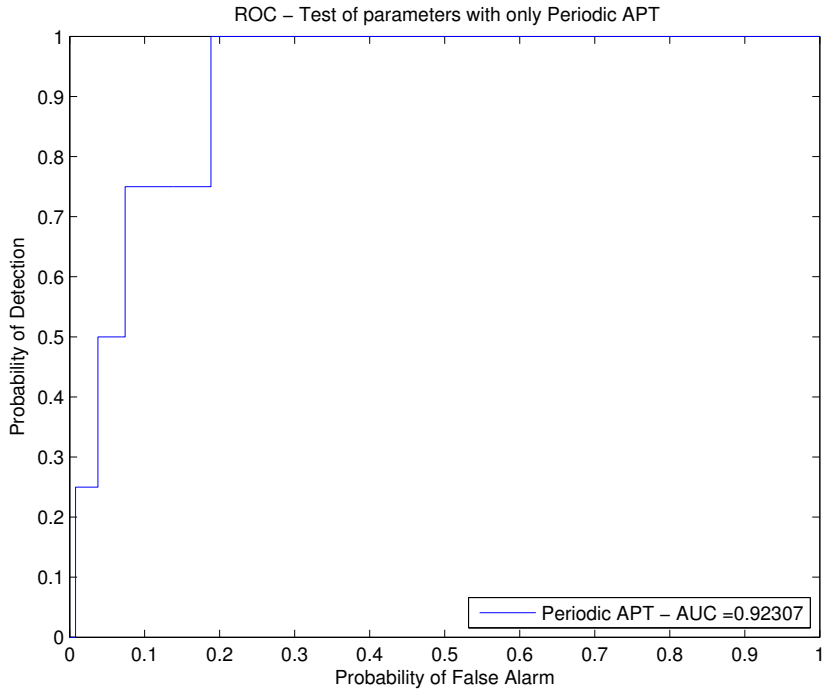


FIGURE 5.25 – ROC de l'étude spécifique d'APT périodiques

5.8 Validation des paramètres

L'ensemble de paramètres a été testé dans deux cas spécifiques et les résultats présentés sont cohérents. Il est maintenant intéressant de soumettre l'algorithme et l'ensemble de paramètres déterminés à un nouveau scénario. Ce scénario va récupérer la multitude d'APT décrites aux sections 5.6 et 5.7. Chacune des APT va être injectée à un utilisateur (Code B.11). La période considérée reste identique. Le sous-réseau choisi est lui nouveau. Il comporte 66 utilisateurs et n'a pas été étudié au préalable. Le nombre d'utilisateurs étant plus important, la probabilité qu'une APT ressorte par hasard relativement haut dans le classement est encore plus faible que dans les cas précédents. Au total, le fichier considéré contient 1 032 021 requêtes (infections comprises). Dans ce scénario, l'*APT5* effectue 11 requêtes, l'*APT6* en effectue 24, l'*APT7* 25, l'*APT8* 33, l'*APT9* 9, l'*APT10* 19, l'*APT11* 39 et, finalement, l'*APT12* 239. Les paramètres utilisés sont maintenant bien connus et sont les suivants :

Valeur de k	**100**	Clusters z-score	**False**
Similarité : Temps	**0,1**	Nbr. Req. (/dom./utilisateur)	**5**
Similarité : Domaine	**0,9**	Classement : Parents	**0,4**
Seuil de pruning	**0,0**	Classement : Enfants	**0,4**
Pruning z-score	**True**	Classement : Nbr. Req.	**0,2**
Taille maximale des clusters	**1 000 000**		

Le résultat est donné à la figure 5.26. L'AUC correspond à nouveau à tout ce qui a été produit avant en étant proche de 0,9. Ceci permet donc de confirmer l'ensemble de paramètres comme étant un bon ensemble de valeurs par défaut pour l'analyse d'un sous-réseau.

Pour ce scénario, le Batch Processor a besoin de presque 11h pour préparer les graphes. L'entièreté des données précalculées occupe 248,8 MB d'espace disque une fois stockée. Le Server, lui, n'a besoin que de 1h30 pour produire le résultat, dont 1h pour le clustering. Comme déjà expliquée plusieurs fois, cette durée peut être améliorée en supprimant le clustering.

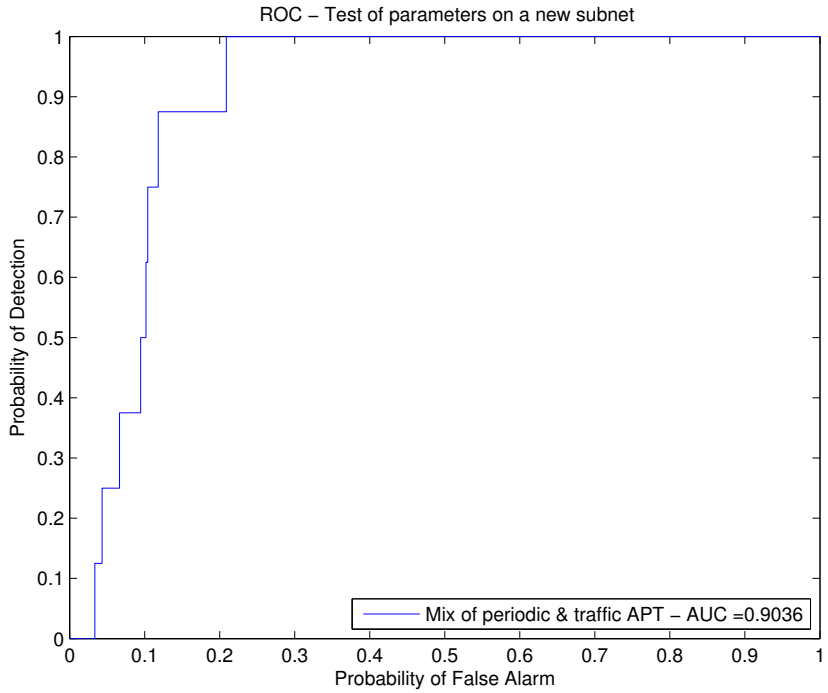


FIGURE 5.26 – ROC pour la combinaison de paramètres optimaux, pour un nouveau sous-réseau

Il peut ensuite être noté que, sur les 8 496 domaines présents dans le log, 66,9% d'entre eux ont été exclus (5 683 domaines) grâce à la white list et au nombre minimal de requêtes par utilisateur et par domaine. Pour retrouver l'ensemble des injections, 20,9% des domaines sont nécessaires (1 781 domaines). Le premier domaine est, quant à lui, retrouvé après l'étude de 3,4% des domaines (287 domaines). Finalement, l'ordre dans lequel sont détectées les APT est le suivant : *APT9*, *APT5*, *APT10*, *APT8*, *APT7*, *APT11*, *APT6* et *APT12*.

5.9 Domaines isolés par l'algorithme

Les domaines erronément isolés constituent la source de fausses alarmes et il est clair que l'amélioration de la détection est au prix de la diminution du taux de fausses alarmes. Pour ce faire, il est nécessaire d'étudier les domaines isolés pour en sortir des caractéristiques pouvant être ciblées. C'est grâce à ce type d'étude que la similarité basée sur les noms de domaine a été établie.

Actuellement, les domaines isolés sont de multiples origines :

- les Content Delivery Network (CDN),
- les domaines publicitaires,
- les Application Programming Interface (API),
- les sites très peu fréquentés,
- les sites dont l'entièreté des pages proviennent du même domaine.

Cette liste n'est pas exhaustive et il est très difficile de sortir un certain ordre des domaines isolés. Une étude approfondie des domaines isolés pourrait être réalisée afin d'essayer de réduire le taux de fausses alarmes (Section 6.2.1).

Il est finalement important de faire remarquer qu'il est très difficile de juger de la dangerosité d'un domaine apparaissant dans le classement. Dans le cas des données étudiées pour ce travail, certains domaines isolés n'existaient même plus. Il est donc encore plus difficile d'évaluer l'origine de l'isolement de ces domaines.

5.10 Conseils à l'analyste

Il est intéressant de résumer l'ensemble des conseils qui peuvent être donnés à un(e) analyste pour l'utilisation des paramètres de l'algorithme. Le k doit être déterminé a priori et doit être choisi relativement grand. Ceci permet de récolter un maximum de liens faibles qui contribuent à renforcer la modélisation des sites par répétition des liens. De plus, l'algorithme NN-Descent approxime mieux les k-NN graphes lorsque le k est grand [18]. Ce choix doit être guidé par le temps que l'analyste est prêt à consacrer au préprocessing et par l'espace de stockage qu'il souhaite consacrer aux graphes. Le temps a été identifié comme déterminant. Un k de 100 est un choix valable.

Concernant la fusion des similarités, le poids de la similarité basée sur les noms de domaines doit être dominant. En effet, cette similarité est très efficace dans la modélisation du log. Le poids de la similarité temporelle ne peut toutefois pas être nul. En effet, cette dernière assure un lien fort entre les domaines n'ayant pas de similarité basée sur les noms de domaine, mais apparaissant régulièrement comme parent et enfant l'un de l'autre. Un poids de 0,1 est conseillé pour la similarité temporelle et un poids de 0,9 pour la similarité basée sur les noms de domaine.

Le seuil de pruning doit être adapté en fonction du type d'APT. Une APT périodique a tendance à s'isoler en faisant des requêtes lorsque l'utilisateur n'est pas présent. Dans ce cas, un seuil de pruning inférieur à la moyenne est efficace pour isoler celle-ci. Lorsqu'une APT basée sur le flux de données est considérée, il est conseillé de choisir un seuil égal à la moyenne. En effet, il est nécessaire de couper tous les liens faibles et moyens, car l'APT a tendance à créer ce type de liens avec les modèles de sites en s'injectant dans les bursts de données de ces sites. Vu que ce travail cible principalement les APT basées sur le flux de données, il est donc conseillé de prendre un seuil égal à la moyenne.

Le filtrage basé sur la taille des clusters n'est pas conseillé. Comme expliqué précédemment, les APT (et particulièrement les APT basées sur le flux de données) ont tendance à rester liées à des modèles de sites. Ces APT sont donc incluses dans les clusters et éliminées lors du filtrage. Dans le cadre de l'algorithme actuellement implémenté, choisir une taille maximale plus grande que la plus grande taille de cluster permet de ne pas filtrer sur les tailles de clusters.

Concernant le nombre minimum de requêtes par domaine et par utilisateur, il est important d'exploiter l'expertise de l'analyste. En effet, il est nécessaire d'estimer l'agressivité de l'APT recherchée. Cette estimation permet de nettoyer les résultats d'énormément de domaines constituant du bruit (fausses alarmes). Si le nombre minimum de requêtes est surestimé, le risque est important d'éliminer les connexions d'APT. Il a été choisi de travailler avec un minimum de 5 requêtes par domaines et par utilisateur sur une période de 10 jours.

Les poids des indices du classement permettent de cibler de manière efficace soit des domaines agressifs, soit des domaines discrets. Il est conseillé d'utiliser un poids d'indice lié au nombre de requêtes qui soit positif et faible pour cibler les domaines peu agressifs. C'est a priori le comportement recherché dans le cadre de ce travail. Un poids négatif et faible permet, lui, de cibler les domaines ayant un nombre important de requêtes. Finalement, il est conseillé d'utiliser le même poids pour l'indice lié au nombre de liens avec des domaines parents et celui lié au nombre de domaines enfants. Il a été choisi de travailler avec un poids de 0,2 pour le poids de l'indice lié au nombre de requêtes et un poids de 0,4 pour les deux autres indices.

En résumé, les valeurs initiales conseillées à l'analyste sont les suivantes :

Valeur de k	**100**	Clusters z-score	**False**
Similarité : Temps	**0,1**	Nbr. Req. (/dom./utilisateur)	**5**
Similarité : Domaine	**0,9**	Classement : Parents	**0,4**
Seuil de pruning	**0,0**	Classement : Enfants	**0,4**
Pruning z-score	**True**	Classement : Nbr. Req.	**0,2**
Taille maximale des clusters	**1 000 000**		

5.11 Conclusion

Pour effectuer une étude complète des paramètres intervenant dans l'algorithme, un scénario a été choisi. Il comprend la sélection d'une période dans le temps, d'un sous-réseau et d'infections simulées. Pour juger au mieux des capacités de détection lors de l'étude des paramètres, des APT typiques ont été choisies : deux APT périodiques et deux APT basées sur le flux des données. Pour chacun des deux types, l'une des infections était agressive, et l'autre plutôt discrète.

L'influence des paramètres sur la capacité de détection a été étudiée au moyen de ROC et de la maximalisation de l'AUC. L'objectif était de déterminer un ensemble de paramètres donnant une AUC maximale (Section 5.10). Le seuil de pruning a été identifié comme un paramètre clé de la détection. Celui-ci permet facilement de cibler les APT périodiques ou les APT basées sur le flux de données. Les poids du classement permettent, eux, de cibler les APT plutôt agressives ou discrètes. La détection d'APT discrètes basées sur le flux de données a été favorisée.

L'étude des paramètres a permis d'identifier une faiblesse au niveau de l'algorithme. Celui-ci consacre un temps conséquent au calcul de clusters. Or, il a été montré que le filtrage basé sur la taille des clusters n'apporte pas de faculté supplémentaire en matière de détection. Il est même plus intéressant de ne pas filtrer sur la taille des clusters. Ceci s'explique par le fait que les APT basées sur le flux de données sont systématiquement connectées à des structures fortement liées modélisant un site. Les APT sont donc absorbées durant le clustering dans le modèle de ce site.

Une fois l'ensemble des paramètres déterminés, deux scénarios spécifiques ont été étudiés. Un scénario ne comprenant que des APT basées sur le flux de données et un scénario ne comprenant que des APT périodiques. Il a été montré que les résultats obtenus correspondent fortement à ce qui était attendu et observé lors de l'étude des paramètres.

Finalement, l'algorithme a été soumis à un dernier test ayant pour objectif de vérifier la généralité des paramètres déterminés. Pour ce faire, un nouveau sous-réseau a été infecté avec une multitude d'APT. L'objectif était d'évaluer la capacité de détection de l'algorithme en utilisant l'ensemble de paramètres déterminés dans ce chapitre. Il a été montré que les résultats correspondent à nouveau à ce qui était attendu et observé dans les tests précédents.

Ce chapitre fournit donc un ensemble de paramètres pouvant servir de valeur par défaut pour l'étude d'un sous-réseau. Différentes nuances sur ces paramètres ont été expliquées dans ce chapitre. Ces nuances permettent à l'analyste intéressé(e) d'adapter les paramètres pour cibler au mieux le type d'APT qu'il (elle) cherche à détecter. Il est finalement conclu que la modélisation par graphes d'un log de proxy est un outil efficace pour détecter des APT sur un réseau.

6. Conclusion

6.1 Introduction

Ce dernier chapitre est consacré au bilan du travail effectué. Une première section présente les améliorations potentielles du travail. Une deuxième section donne les détails concernant un bug connu dans le processing au moment de la finalisation de ce travail. La dernière section de ce chapitre et du mémoire est, elle, consacrée à la conclusion du travail.

6.2 Améliorations potentielles du travail

L'objectif de cette section est de donner au lecteur intéressé un aperçu des possibilités existantes pour continuer ce travail. Les deux premiers points abordent les améliorations possibles de l'algorithme lui-même et de l'interface utilisateur. Par la suite, trois autres points nécessitant une étude plus approfondie sont abordés, à savoir les caractéristiques des APT étudiées, la taille du sous-réseau étudié et la période étudiée.

6.2.1 Amélioration de l'algorithme

L'algorithme développé pourrait être amélioré sur plusieurs points. L'amélioration la plus importante est clairement liée à la problématique du clustering et du filtering. Ceux-ci ont été identifiés comme inefficaces dans l'état d'implémentation actuel. Il serait nécessaire de revoir les critères de clustering pour éviter de lier systématiquement les APT basées sur le flux de données aux modèles des sites. Une autre solution pourrait être d'abandonner définitivement le clustering et le filtering. Le gain en temps de calcul serait évidemment considérable.

Les techniques d'agrégation de données utilisées durant ce travail ont toujours été des sommes pondérées, mais des opérateurs plus généraux pourraient aussi être testés. Parmi les candidats, les opérateurs Ordered Weighted Average (OWA) et Weighted Ordered Weighted Average (WOWA) peuvent être cités. L'algorithme pourrait également être amélioré en proposant d'autres similarités pertinentes provenant de l'observation des domaines isolés. Une autre amélioration pourrait être apportée en trouvant une méthode efficace de discrimination entre les requêtes parentes et enfants. Finalement, la technique utilisée actuellement pour dériver un classement sur base d'un graphe pourrait également être améliorée. Toutes ces améliorations ont pour objectif de réduire le taux de fausses alarmes.

6.2.2 Amélioration de l'interface utilisateur

De manière générale, l'interface utilisateur est relativement rudimentaire et pourrait être améliorée. Mais, il a été constaté que l'affichage du graphe nécessite, lui, une mise à jour bien plus profonde. En effet, lorsque le nombre de liens devient trop important, l'interface plante. Or, il

a été remarqué que c'est en ne filtrant pas sur la taille des clusters que les résultats sont les meilleurs. Le nombre de liens est donc forcément important.

Admettons que l'affichage d'un nombre important de liens ne pose plus de problème. Il est toutefois encore nécessaire de retravailler les interactions possibles avec ce graphe. En effet, celles-ci sont actuellement très limitées. Il est possible de consulter la valeur d'un lien et de mettre un noeud en évidence. À l'avenir, il serait intéressant de pouvoir zoomer, se déplacer dans le graphe, rechercher un domaine ou encore mettre en évidence automatiquement les voisins d'un noeud. C'est au prix de ces améliorations que l'analyste pourra tirer une vraie plus-value de l'affichage du graphe.

6.2.3 Étude approfondie des caractéristiques des APT injectées

Afin d'évaluer les capacités de l'algorithme en matière de détection, différentes infections ont été simulées. Bien que ces simulations couvrent de nombreux scénarios, il est clair que l'entièreté des cas n'a pas pu être traitée.

Il est, par exemple, possible de faire une étude plus approfondie des infections de plusieurs utilisateurs par une même APT. Il a été montré mathématiquement que les scénarios choisis permettaient de cibler, dans une certaine mesure, ces cas-là également. Mais, des tests supplémentaires seraient nécessaires. De plus, il serait également intéressant d'étudier l'influence de l'intensité du trafic d'un utilisateur sur la détection d'une APT. L'APT a beau être agressive, si l'utilisateur n'est pas présent, une APT basée sur le flux de données ne sera pas détectée. Il est clair que ses capacités de communication avec un C2 seront également limitées.

En ce qui concerne les APT basées sur le flux de données, il serait intéressant d'étudier d'autres types de bursts et de voir leur impact sur la détection. Dans ce travail, le burst a été défini comme un flux continu de données pour un utilisateur donné pendant une durée donnée et certains paramètres ont été choisis. Il pourrait être intéressant de faire varier ces paramètres, par exemple faire varier le délai d'injection ou la distance temporelle entre deux requêtes du même burst. Il pourrait également être intéressant de voir l'impact d'une autre définition du burst. Une autre définition possible serait de baser l'injection d'APT sur le nombre de requêtes effectuées par un utilisateur durant une période donnée.

En ce qui concerne les APT périodiques, il pourrait être intéressant de voir l'effet sur la détection d'un masque sur les injections d'APT. Ce masque permettrait une infection de type périodique, mais uniquement en cas de présence de l'utilisateur.

Finalement, rien n'oblige de se limiter aux deux types d'APT choisis et il pourrait également être intéressant de voir la capacité de détection de l'algorithme pour des APT qui ne s'injecteraient, par exemple, que lorsqu'un domaine spécifique est consulté.

6.2.4 Influence de la taille du sous-réseau étudié

Il pourrait être intéressant d'étudier l'influence de la taille d'un sous-réseau sur les capacités de détection de l'algorithme. Ceci permettrait de donner des conseils concrets, lors de la phase de conception de l'architecture d'un réseau, sur la taille optimale d'un sous-réseau. L'intensité d'une similarité naît de la répétition d'un lien entre deux domaines du graphe. Il est donc attendu que la taille du sous-réseau étudié permette d'influencer le nombre de répétitions, et donc la capacité de modélisation des sites. Il peut également être attendu que cette taille soit un paramètre permettant d'influencer les caractéristiques du trafic de fond. Or, c'est justement de ce bruit de fond qu'il faut faire ressortir les connexions vers un C2. Finalement, cette étude permettrait de donner une idée concrète des capacités de l'algorithme en matière de traitement de données volumineuses.

6.2.5 Influence de la période étudiée

Il a été choisi de travailler sur une période de 10 jours et ce choix était arbitraire. Or, la répétition de liens entre domaines est à la base de la création de structures dans le graphe. Il peut donc être intéressant d'étudier l'influence de la durée de la période sur la détection. Ceci permettrait de trouver une durée minimale d'étude nécessaire pour être capable de détecter des APT. Cette étude permettrait de conseiller de manière concrète les analystes sur la manière d'aborder les logs pour leurs recherches. Elle permettrait également de donner une idée concrète des capacités de l'algorithme en matière de traitement de données volumineuses.

Une autre possibilité serait d'étudier l'influence du moment choisi dans l'année pour l'étude. Il serait intéressant de voir si les caractéristiques du réseau évoluent au cours de l'année, entraînant ainsi des variations dans les capacités de détection.

6.3 Bug connu

Il est finalement intéressant de noter que, lors de la finalisation de ce travail, un bug est connu et non résolu. Ce bug est lié à la version `v.40` de la libraire `info.debatty.java.graphs` [14]. Ce bug sera probablement résolu dans les versions ultérieures de la librairie.

Lorsque l'algorithme permettant la création de graphes (*ThreadedNNDescent* ou *NNDescent*) est confronté à de nombreux noeuds ayant la même similarité entre eux, l'algorithme n'est pas capable de choisir k noeuds parmi ces candidats. Un cas typique dans lequel ce problème est rencontré est un cas où un utilisateur génère plus de k requêtes simultanées. Dans ce cas, l'algorithme semble être en *dead lock* et bloque complètement le fonctionnement du Batch Processor.

Un test unitaire a été rédigé pour vérifier la résistance des algorithmes utilisés à ce bug. Toutefois, vu que ce test n'est actuellement pas réussi, il a été mis en commentaire.

6.4 Conclusion générale

Ce travail a été effectué dans le cadre des recherches menées par la cellule Research Unit Cyber Defence (RUCD) de l'École Royale Militaire. Son objectif était de continuer le travail du Lt Mattijs HERTSENS sur l'élaboration d'un algorithme permettant la détection d'Advanced Persistent Threat (APT). Les APT ciblées sont soit périodiques, soit basées sur le flux de données. Les Command and Control centers (C2) utilisés sont centralisés. L'algorithme modélise via des graphes le trafic contenu dans un log de proxy. Grâce à une modélisation relativement fiable du log par un graphe, une détection d'APT basée sur la recherche d'anomalies dans le graphe est possible. Cette approche a le grand avantage de permettre la détection d'APT encore inconnues des services de sécurité et diffère en cela de la détection basée sur une signature spécifique. Ce travail devait apporter une amélioration significative par rapport au travail du Lt Mattijs HERTSENS sur deux points. Les paramètres utilisés devaient être rendus plus physiques afin de faciliter leur manipulation par un(e) analyste. Ce travail devait également parvenir à augmenter l'interactivité entre l'analyste et l'algorithme.

Lors de ce travail, un algorithme permettant le calcul du graphe et l'isolement des APT a été conçu. Cet algorithme a le grand avantage de conserver jusqu'au rendu graphique un maximum des données contenues dans le log. L'analyste est le (la) seul(e) responsable de la suppression d'informations. L'algorithme est divisé entre trois segments. Le *Core* est responsable, entre autres, de la définition des similarités utilisées dans le graphe. Le *Batch Processor* est lui responsable du préprocessing. La solution choisie précalcule ainsi un maximum de données avant l'étude de l'analyste. Ces données sont alors utilisées par le troisième segment : le *Server*. Celui-ci traite

les données précalculées en utilisant les paramètres fournis par l'analyste. Il met également les résultats à disposition d'une interface graphique. Cette interface a été réalisée pour augmenter l'interaction entre l'analyste et les résultats. Celle-ci permet, entre autres, de choisir les paramètres, de visualiser le graphe des résultats et de consulter le classement des domaines.

Il a été nécessaire d'écrire des outils supplémentaires facilitant le travail de l'analyste. Au total, quatre outils ont été conçus. Le premier permet la simulation d'infections dans un log de proxy. Le second permet l'étude du trafic contenu dans ce log. Le troisième et le quatrième permettent l'étude de l'influence des paramètres intervenant dans l'algorithme sur les capacités de détection. Ils exploitent les Receiver Operating Characteristics (ROC) et la maximalisation d'Area Under Curve (AUC). L'ensemble du travail d'implémentation a été fait le plus rigoureusement possible en utilisant des outils standards et mondialement utilisés par la communauté de développeurs.

Le travail réalisé répond aux objectifs fixés, car l'algorithme proposé offre à l'analyste une large gamme de paramètres ayant tous un sens physique. Il est possible de développer une certaine intuition par rapport à leur influence en les manipulant. Une grande partie du travail a été consacrée à l'étude de ces paramètres afin de conseiller au mieux l'analyste dans leur utilisation (Section 5.10). De plus, il a été possible de réduire de manière importante le temps nécessaire au recalcul des résultats lors du changement d'un paramètre. En fonction du paramètre qui est adapté, la durée du recalcul peut varier d'une durée d'environ une heure à une réponse instantanée.

Le travail ouvre la porte à des études complémentaires sur différentes options prises lors de l'implémentation de l'algorithme et sur certaines améliorations devant y être apportées. Les résultats obtenus sont néanmoins prometteurs : la modélisation d'un log de proxy par des graphes s'avère être un outil efficace pour détecter des APT sur un réseau. Ce mémoire est donc conclu en espérant que des recherches ultérieures parviendront à rendre cet algorithme opérationnel.

Annexes

A. Infection de plusieurs utilisateurs, développement de la similarité

Cette annexe a pour objectif de donner le raisonnement mathématique amenant à considérer qu'une infection intensive d'un utilisateur permette également de simuler, dans une certaine mesure, une infection discrète d'un plus grand nombre d'utilisateurs.

Soit

- $\{u_k\}_{k\in[1,K]}$, l'ensemble des K utilisateurs ;
- A et B, deux domaines ;
- $\{a_i\}_{i\in[1,I]}$, l'ensemble des I requêtes vers le domaine A ;
- $\{a_i^k\}_{i\in[1,I^k]}$, l'ensemble des I^k requêtes provenant de l'utilisateur u_k vers le domaine A ;
- $\{b_j\}_{j\in[1,J]}$, l'ensemble des J requêtes vers le domaine B ;
- $\{b_j^k\}_{j\in[1,J^k]}$, l'ensemble des J^k requêtes provenant de l'utilisateur u_k vers le domaine B ;
- α, le poids associé à la similarité temporelle ;
- β, le poids associé à la similarité basée sur les noms de domaine.

Supposons que les domaines A et B aient L^k requêtes ayant une similarité définie entre elles pour un utilisateur u_k tel qu'il existe un ensemble de couples $\{a_l^k, b_l^k\}_{l\in[1,L^k]}$, avec $a_l^k \in \{a_i^k\}_{i\in[1,I^k]}$ et $b_l^k \in \{b_j^k\}_{j\in[1,J^k]}$.
Alors, les notations suivantes peuvent être définies :

- $\mu_{\Delta t}(a_l^k, b_l^k)$, la similarité temporelle entre les requêtes a_l^k et b_l^k de l'utilisateur u_k ;
- $\mu_{dom}(a_l^k, b_l^k)$, la similarité basée sur les noms de domaine entre les requêtes a_l^k et b_l^k de l'utilisateur u_k ;
- $\mu_{\Delta t}(A^k, B^k)$, la similarité temporelle entre les requêtes provenant de l'utilisateur u_k vers les domaines A et B ;
- $\mu_{dom}(A^k, B^k)$, la similarité basée sur les noms de domaine entre les requêtes provenant de l'utilisateur u_k vers les domaines A et B ;
- $\mu(A^k, B^k)$, la similarité entre les requêtes provenant de l'utilisateur u_k vers les domaines A et B, après fusion des similarités ;
- $\mu(A, B)$, la similarité entre les requêtes vers les domaines A et B, après fusion des utilisateurs.

Si $\mu_{\Delta t}(a_l^k, b_l^k)$ est définie entre a_l^k et b_l^k, mais pas $\mu_{dom}(a_l^k, b_l^k)$ (ou inversement), il peut être supposé que $\mu_{dom}(a_l^k, b_l^k) = 0$ (ou respectivement $\mu_{\Delta t}(a_l^k, b_l^k) = 0$).

Ainsi, les expressions suivantes peuvent être dérivées de l'algorithme implémenté (Section 3.4 et 3.5) :

$$\mu_{\Delta t}(A^k, B^k) = \sum_{l=1}^{L^k} \mu_{\Delta t}(a_l^k, b_l^k) \tag{A.1}$$

$$\mu_{dom}(A^k, B^k) = \sum_{l=1}^{L^k} \mu_{dom}(a_l^k, b_l^k) \tag{A.2}$$

Et,

$$\mu(A^k, B^k) = \alpha\ \mu_{\Delta t}(A^k, B^k) + \beta\ \mu_{dom}(A^k, B^k) \tag{A.3}$$

$$= \alpha \sum_{l=1}^{L^k} \mu_{\Delta t}(a_l^k, b_l^k) + \beta \sum_{l=1}^{L^k} \mu_{dom}(a_l^k, b_l^k) \tag{A.4}$$

Alors,

$$\mu(A, B) = \sum_{k=1}^{K} \mu(A^k, B^k) \tag{A.5}$$

$$= \sum_{k=1}^{K} \Big(\alpha \sum_{l=1}^{L^k} \mu_{\Delta t}(a_l^k, b_l^k) + \beta \sum_{l=1}^{L^k} \mu_{dom}(a_l^k, b_l^k) \Big) \tag{A.6}$$

Or, les APT considérées utilisent toutes un domaine qui leur est propre. La similarité $\mu_{dom}(a_l^k, b_l^k)$ pour une APT est donc toujours nulle. Ainsi, en supposant que A est le domaine associé au C2 d'une APT, l'expression A.6 se simplifie comme suit :

$$\mu(A, B) = \sum_{k=1}^{K} \Big(\alpha \sum_{l=1}^{L^k} \mu_{\Delta t}(a_l^k, b_l^k) \Big) \tag{A.7}$$

Afin de prouver l'équivalence mathématique entre les similarités d'une infection intensive d'un utilisateur et d'une infection discrète d'un plus grand nombre d'utilisateurs, deux cas sont maintenant considérés.

Dans le premier cas, il est supposé qu'un seul utilisateur u_m ($u_m \in \{u_k\}_{k\in[1,K]}$) est infecté par l'APT en question. L'expression A.7 se réduit alors comme suit :

$$\mu(A, B) = \alpha \sum_{l=1}^{L^m} \mu_{\Delta t}(a_l^m, b_l^m) \tag{A.8}$$

Dans le second cas, il est supposé que K' utilisateurs sont infectés ($K' \leq K$). Pour chacun d'entre eux, l'APT n'a effectué qu'une seule requête et celle-ci n'a qu'un lien temporel avec le domaine B. L'expression A.7 se réduit alors comme suit :

$$\mu(A, B) = \sum_{k=1}^{K'} \Big(\alpha\ \mu_{\Delta t}(a_1^k, b_1^k) \Big) \tag{A.9}$$

Afin de faire le lien entre les expressions A.8 et A.9, le trafic malicieux et les voisins temporels correspondant de l'expression A.9 sont alors attribués à un utilisateur fictif u_{k+1}. Son comportement sur le réseau est donc défini comme l'ensemble des connexions au C2 et des voisins temporels correspondant. Au total, K' injections seraient comptabilisées pour cet utilisateur fictif, car l'APT a établi une connexion par utilisateur infecté. Dans ce cas, il peut être constaté que $L^m = K'$ et que l'expression A.8 devient :

$$\mu(A, B) = \alpha \sum_{l=1}^{K'} \mu_{\Delta t}(a_l^{k+1}, b_l^{k+1}) \tag{A.10}$$

L'expression A.10 est équivalente à l'expression A.9 compte tenu de la propriété de la distributivité de $\cdot$ par rapport à $+$ et de la définition du trafic de l'utilisateur fictif u_{k+1}.

Le cas considéré se limite à un lien temporel par utilisateur entre la requête de l'APT et le domaine B pour établir la relation A.9. Toutefois, il est possible de faire un raisonnement semblable pour un nombre plus important de similarités par utilisateur.

Il a donc été montré qu'il est mathématiquement possible de faire le lien entre la similarité obtenue par une infection intensive d'un utilisateur unique et par une infection discrète d'un plus grand nombre d'utilisateurs.

B. Codes

L'entièreté des outils Java développés ainsi que l'interface utilisateur peuvent être retrouvés sur la page GitHub du projet : `https://github.com/RUCD/apt-graph` [53] [54]. D'autres codes utilisés lors du travail peuvent être retrouvés ci-après :

```
% This script produces multiple histograms of traffic.
% author : Thomas Gilon (2017)

close all
clc
clear all

% First part of the paths to the data
path_csv = '/Volumes/GILON/CSV/';

%% Traffic data loading
data = csvread(strcat(path_csv,'192.168.129.0.csv'));
data_infected_1 = csvread(strcat(path_csv,'192.168.129.0_infected_1.csv'));
data_infected_2 = csvread(strcat(path_csv,'192.168.129.0_infected_2.csv'));
data_infected_3 = csvread(strcat(path_csv,'192.168.129.0_infected_3.csv'));
data_infected = csvread(strcat(path_csv,'192.168.129.0_infected.csv'));
data_192_168_129_119 = csvread(strcat(path_csv,'192.168.129.119.csv'));
data_192_168_129_107 = csvread(strcat(path_csv,'192.168.129.107.csv'));
data_192_168_129_114 = csvread(strcat(path_csv,'192.168.129.114.csv'));
data_192_168_129_104 = csvread(strcat(path_csv,'192.168.129.104.csv'));
data_demo_periodic = csvread(strcat(path_csv,'demo_periodic_infected.csv'));
data_demo_traffic = csvread(strcat(path_csv,'demo_traffic_infected.csv'));

%% Traffic analyze
figure
title_string = 'Traffic on the network';
traffic_fun(data, [], data(1,1), title_string, {})

figure
title_string = 'Infected traffic on the network';
traffic_fun(data_infected, [], data(1,1), title_string, {})

figure
title_string = 'Traffic of a user and his infection';
traffic_fun(data_192_168_129_119, [data_infected_1(:,1) ...
    data_infected_1(:,2)-data(:,2)], data(1,1), title_string,...
    {'Traffic','APT periodic by hour'})

figure
title_string = 'Traffic of a user and his infection';
traffic_fun(data_192_168_129_107, [data_infected_2(:,1) ...
    data_infected_2(:,2)-data_infected_1(:,2)], data(1,1),...
    title_string, {'Traffic','APT periodic by 12 hour'})

figure
```

```
title_string = 'Traffic of a user and his infection';
traffic_fun(data_192_168_129_114, [data_infected_3(:,1) ...
    data_infected_3(:,2)-data_infected_2(:,2)], data(1,1) ,...
    title_string, {'Traffic','APT traffic stealthy'})

figure
title_string = 'Traffic of a user and his infection';
traffic_fun(data_192_168_129_104, [data_infected(:,1) ...
    data_infected(:,2)-data_infected_3(:,2)], data(1,1) ,...
    title_string, {'Traffic','APT traffic intensive'})

%% Demo APT traffic and periodic
figure
title_string = 'Traffic of a user and his infection';
traffic_fun(data_demo_periodic, [data_demo_periodic(:,1) ...
    data_demo_periodic(:,2)-data_192_168_129_119(:,2)], data(1,1) ,...
    title_string, {'Traffic','APT periodic by hour'})

figure
title_string = 'Traffic of a user and his infection';
traffic_fun(data_demo_traffic, [data_demo_traffic(:,1) ...
    data_demo_traffic(:,2)-data_192_168_129_119(:,2)], data(1,1) ,...
    title_string, {'Traffic','APT traffic'})
```

Listing B.1 – Script permettant l'affichage des histogrammes de trafic (Section 4.4.2) (Matlab)

```
function [ ] = traffic_fun( data_user, data_infection, time_rfc,...
    title_string, legends)
% Function plotting the traffic of a specific user and his infection,
% setting the time reference at 0h for the start of the traffic and setting
% the title and the legends provided.
% author : Thomas Gilon (2017)
%
% input : data_user, traffic of a specific user ; data_infection, traffic
% of the APT only ; time_rfc, 0h time reference ; title_string, title ;
% legends, name of the sets of data
%
% output : /

    semilogy((data_user(:,1)-time_rfc)/1000/3600,data_user(:,2)+1,'*-')
    if (~isempty(data_infection))
        hold on
        semilogy((data_infection(:,1)-time_rfc)/1000/3600,...
            data_infection(:,2)+1,'r*-')
        legend(legends)
    end
    title(title_string)
    xlabel('Time [h]')
    ylabel('Number of requests + 1')
    box on
end
```

Listing B.2 – Fonction créant l'histogramme du trafic d'un utilisateur et de l'infection liée à cet utilisateur (Section 4.4.2) (Matlab)

```
% Script plotting the ROC analyzing the influence of the different
% parameters. Theses ROC have been produced by the bash script.
% author : Thomas Gilon (2017)

close all
clc
clear all

% First part of the paths to the ROC
path_ROC = '/Volumes/GILON/CSV/ROC_';

%% K
figure
k_string = {};
stop = 50;
for i=[10:10:stop 100]
    k_string = cat(2,k_string,{sprintf('%.0f',i)});
end
paths = {};
for i=1:length(k_string)
    paths = cat(2, paths, strcat(path_ROC, 'k_', k_string{i:i}, '.csv'));
end
title_k = strcat('ROC - Sweep k');
ROC_fun(paths, title_k, k_string)

%% k = 40 vs k = 100 - Custom settings
figure
k_string = {};
for i=[1:1:10]
    k_string = cat(2,k_string,{sprintf('%.0f',i)});
end
paths = {};
for i=1:length(k_string)
    paths = cat(2, paths, strcat(path_ROC, 'custom_', k_string{i:i}, '.csv'));
end
title_k = 'ROC - Custom settings';
ROC_fun(paths, title_k, k_string)


%% Features Time & Domain
figure
feature_weights_time = {};
feature_weights_domain = {};
stop = 1;
for i=0.1:0.1:0.1
    feature_weights_time = cat(2,feature_weights_time,{sprintf('%.1f',i)});
    feature_weights_domain = cat(2,feature_weights_domain,...
        {sprintf('%.1f',stop-i)});
end
paths = {};
for i=1:length(feature_weights_time)
    paths = cat(2, paths, strcat(path_ROC, 'feature_weights_time_',...
        feature_weights_time{i:i}, '_feature_weights_domain_',...
        feature_weights_domain{i:i}, '.csv'));
end
title_features = strcat('ROC - Sweep Features (Sweep in Time,',...
    ' Domain = ', num2str(stop), ' - Time)');
ROC_fun(paths, title_features, feature_weights_time)

%% Pruning
figure
pruning_string = {};
for i=-0.1:0.1:0
```

```
    pruning_string = cat(2,pruning_string, {sprintf('%.1f',i)});
end
paths = {};
for i=1:length(pruning_string)
    paths = cat(2, paths, strcat(path_ROC, 'prune_threshold_',...
        pruning_string{i:i}, '.csv'));
end
title_pruning = 'ROC - Sweep Pruning';
ROC_fun(paths, title_pruning, pruning_string)

%% Max Clusters Size
figure
max_cluster_size = {};
for i=[1 10:10:100 1000000]
    max_cluster_size = cat(2,max_cluster_size,{sprintf('%.f',i)});
end
paths = {};
for i=1:length(max_cluster_size)
    paths = cat(2, paths, strcat(path_ROC, 'max_cluster_size_',...
        max_cluster_size{i:i}, '.csv'));
end
title_max_cluster_size = 'ROC - Sweep Max Clusters Size';
ROC_fun(paths, title_max_cluster_size, max_cluster_size)

%% Number of Requests
figure
number_requests = {};
for i=[1 2 3 4 5 10 15 20]
%for i=1:1:20
    number_requests = cat(2,number_requests,{sprintf('%.f',i)});
end
paths = {};
for i=1:length(number_requests)
    paths = cat(2, paths, strcat(path_ROC, 'number_requests_',...
        number_requests{i:i}, '.csv'));
end
title_number_requests = 'ROC - Sweep Number of Requests';
ROC_fun(paths, title_number_requests, number_requests)

%% Ranking Parents & Children & Number of requests (mode 1)
figure
ranking_parents = {};
ranking_children = {};
ranking_requests = {};
stop = 1;
%for i=[-0.2 0.2]
for i=-1:0.2:1
    ranking_parents = cat(2,ranking_parents,{sprintf('%.1f',(stop-i)/2)});
    ranking_children = cat(2,ranking_children,{sprintf('%.1f',...
        (stop-i)/2)});
    ranking_requests = cat(2,ranking_requests,{sprintf('%.1f',i)});
end
paths = {};
for i=1:length(ranking_parents)
    paths = cat(2, paths, strcat(path_ROC, 'ranking_weights_requests_',...
        ranking_requests{i:i}, '_ranking_weights_parents_',...
        ranking_parents{i:i}, '_ranking_weights_children_',...
        ranking_children{i:i},'.csv'));
end
title_ranking = strcat('ROC - Sweep Ranking (Sweep in Number of',...
    ' Requests, Parents & Children = (', num2str(stop), ' - Req.)/2 )');
ROC_fun(paths, title_ranking, ranking_requests)

```

```
%% Ranking Parents & Children & Number of requests (mode 2)
figure
ranking_parents = {};
ranking_children = {};
ranking_requests = {};
stop = 0.8;
for i=0:0.1:stop
    ranking_parents = cat(2,ranking_parents,{sprintf('%.2f',i)});
    ranking_children = cat(2,ranking_children,...
        {sprintf('%.2f',abs(stop-i))});
    ranking_requests = cat(2,ranking_requests,{sprintf('%.1f',0.2)});
end
paths = {};
for i=1:length(ranking_parents)
    paths = cat(2, paths, strcat(path_ROC, 'ranking_weights_requests_',...
        ranking_requests{i:i}, '_ranking_weights_parents_',...
        ranking_parents{i:i}, '_ranking_weights_children_',...
        ranking_children{i:i},'.csv'));
end
title_ranking = strcat('ROC - Sweep Ranking (Sweep in Number of ',...
    ' Parents, Children = (', num2str(stop), ' - Par.), Requests = 0.2 )');
ROC_fun(paths, title_ranking, ranking_parents)

%% Test of the optimal combinaison of parameters
figure
ROC_fun({strcat(path_ROC,'result.csv')}, strcat('ROC - Result of',...
    ' the study of the parameters'), {'Mix of periodic & traffic APT'})

%% Study of Traffic APT only
figure
ROC_fun({strcat(path_ROC,'traffic.csv')},strcat('ROC - Test of',...
    ' parameters with only Traffic APT'), {'Traffic APT'})

%% Study of Periodic APT only
figure
ROC_fun({strcat(path_ROC,'periodic.csv')},strcat('ROC - Test of',...
    ' parameters with only Periodic APT'), {'Periodic APT'})

%% Test with a new subnet
figure
ROC_fun({strcat(path_ROC,'ultime.csv')},strcat('ROC - Test of',...
    ' parameters on a new subnet'), {'Mix of periodic & traffic APT'})
```

Listing B.3 – Script produisant les ROC calculées par l'outil de balayage de paramètres (Section 4.4.3) (Matlab)

```
function [ ] = ROC_fun( paths, title_string, legends_string)
% Function plotting the ROC given in paths and setting the title
% and the legends provided.
% author : Thomas Gilon (2017)
%
% input : paths, paths to the ROC ; title_string, title ;
% legends_string, name of the sets of data
%
% output : /

    AUC = zeros(1,length(legends_string));
    for i=1:length(legends_string)
        data = csvread(paths{i:i});
        if data(end,1) < 1
            data = [data ; [1, data(end,2)]];
        end
```

```
        hold all
        [x_stairs, y_stairs] = stairs(data(:,1), data(:,2));
        plot(x_stairs, y_stairs)
        AUC(i) = trapz(x_stairs, y_stairs);
    end
    title(title_string)
    xlabel('Probability of False Alarm')
    ylabel('Probability of Detection')
    legends = {};
    for i=1:length(legends_string)
        legends = cat(2,legends,{strcat(legends_string{i:i},...
            ' - AUC = ',num2str(AUC(i)))});
    end
    legend(legends,'location','SouthEast')
    box on
end
```

Listing B.4 – Fonction créant le graphe des ROC données en paramètres (Section 4.4.3) (Matlab)

```
#!/bin/bash
# APT-GRAPH general study. This script completes a general study of the
# algorithm. It produces ROC allowing study of the influence of each
# parameter of the algorithm.
# author : Thomas Gilon (2017)

SUBNET_REGEX=192.168.129
SUBNET=192.168.129.0

YEAR=2014
MOUNTH=10
DAY_START=1
DAY_STOP=10
K_INPUT=(10 20 30 40 50 100)

PATH_IN="/home/cdn/http-log/"
PATH_LOG="/mnt/DATA/data/log_files/"$YEAR"_"$MOUNTH"_"$DAY_START"-"$DAY_STOP"/"
PATH_OUT="/mnt/DATA/data/graphs/"
PATH_CONFIG="/mnt/DATA/data/ROC/2014_10_1-10_192.168.129.0_multiple_infection/"

INIT_CONFIG="simple.conf"
K_CONFIG=$PATH_CONFIG"k_sweep.conf"
INFECTED_USERS=("192.168.129.104" "192.168.129.107" "192.168.129.114" "
    192.168.129.119")

#Generation of the subnet files
cd batch/
[ -f "$PATH_LOG$SUBNET.out" ]
EXIST_1=$?
if [ $EXIST_1 -ne 0 ]
then
  touch "$PATH_LOG$SUBNET.out"
  for (( i=$DAY_START; i<=$DAY_STOP; i++ ))
  do
    if [ "$i" -lt 10 ]
    then
      FILE="proxy_fwd_iwsva-$YEAR.$MOUNTH.0$i"
    else
      FILE="proxy_fwd_iwsva-$YEAR.$MOUNTH.$i"
    fi
    if [ -f "$PATH_IN$FILE.gz" ]
    then
      [ -f "$PATH_LOG$FILE.log" ]
```

```
      EXIST_2=$?
      if [ $EXIST_2 -ne 0 ]
      then
        echo "Decompression of file : $PATH_IN$FILE.gz"
        gunzip -c "$PATH_IN$FILE.gz" > "$PATH_LOG$FILE.log"
      fi
      echo "Parsing file : $PATH_LOG$FILE.log"
      ./subnet_selection.sh $SUBNET_REGEX "$PATH_LOG$FILE.log" "$PATH_LOG$SUBNET.out"
      if [ $EXIST_2 -ne 0 ]
      then
        echo "Removing file : $PATH_LOG$FILE.log"
        rm -v "$PATH_LOG$FILE.log"
      fi
    else
      echo "File not found : $PATH_IN$FILE.gz"
    fi
  done
fi

#Infection of the logs
cd ../infection/
echo "Infection..."
./multiple_infection.sh

#Generation of the histogram of traffic of infected users
cd ../traffic/
echo "Computing histogram of the traffic"
./multiple_traffic.sh

#Generation of the graphs
cd ../batch/
for i in "${K_INPUT[@]}"
do
  echo "Busy with k = $i"
  ./analyze.sh -i "$PATH_LOG$SUBNET"".out" -o "$PATH_OUT$YEAR"_"$MOUNTH"_"$DAY_START"-"$DAY_STOP"_"$SUBNET"_k_"$i" -f json -k $i
done

#Duplication of the non infected graphs (no need to recompute them)
for i in "${K_INPUT[@]}"
do
  cp "$PATH_OUT$YEAR"_"$MOUNTH"_"$DAY_START"-"$DAY_STOP"_"$SUBNET"_k_"$i"/*".ser" "$PATH_OUT$YEAR"_"$MOUNTH"_"$DAY_START"-"$DAY_STOP"_"$SUBNET"_k_"$i""_multiple_infection"
  for j in "${INFECTED_USERS[@]}"
  do
    echo "removing $j..."
    rm -v "$PATH_OUT$YEAR"_"$MOUNTH"_"$DAY_START"-"$DAY_STOP"_"$SUBNET"_k_"$i""_multiple_infection"/$j".ser"
  done
done

#Generation of the infected graphs
cd ../batch/
for i in "${K_INPUT[@]}"
do
  echo "Busy with k = $i (infected)"
  ./analyze.sh -i "$PATH_LOG$SUBNET""_infected.out" -o "$PATH_OUT$YEAR"_"$MOUNTH"_"$DAY_START"-"$DAY_STOP"_"$SUBNET"_k_"$i""_multiple_infection" -f json -k $i
done

#Generation of the config files
```

```
cd ../config/
./config.sh -i $INIT_CONFIG -o $PATH_CONFIG"prune_sweep.conf" -field
    prune_threshold -start -0.5 -stop 1.0 -step 0.1
./config.sh -i $INIT_CONFIG -o $PATH_CONFIG"features_sweep.conf" -field
    feature_weights_time -start 0.0 -stop 1.0 -step 0.1 -multi
    feature_weights_domain
./config.sh -i $INIT_CONFIG -o $PATH_CONFIG"max_cluster_size_sweep.conf" -field
    max_cluster_size -start 1 -stop 110 -step 10
./config.sh -i $INIT_CONFIG -o $PATH_CONFIG"number_requests_sweep.conf" -field
    number_requests -start 0 -stop 20 -step 1
./config.sh -i $INIT_CONFIG -o $PATH_CONFIG"ranking_sweep.conf" -field
    ranking_weights_parents -start -0.0 -stop 0.8 -step 0.1 -multi
    ranking_weights_children

#Generation of the ROC curves
cd ../server/
echo "Busy with k sweep"
./study.sh -i $PATH_CONFIG"k_sweep.conf" &
echo "Busy with prune sweep"
./study.sh -i $PATH_CONFIG"prune_sweep.conf" &
wait
echo "Busy with features sweep"
./study.sh -i $PATH_CONFIG"features_sweep.conf" &
echo "Busy with max cluster size sweep"
./study.sh -i $PATH_CONFIG"max_cluster_size_sweep.conf" &
wait
echo "Busy with number of requests sweep"
./study.sh -i $PATH_CONFIG"number_requests_sweep.conf" &
echo "Busy with ranking sweep"
./study.sh -i $PATH_CONFIG"ranking_sweep.conf" &
wait

echo 'END'
```

Listing B.5 – Script permettant l'analyse complète de l'algorithme (Section 4.4.4) (Bash)

```
#!/bin/bash
# Script allowing the selection of a given subnet in a
# given log file (format JSON)
# author : Thomas Gilon (2017)

if [ "$#" -ne 3 ]; then
  echo 'Illegal arguments !'
  echo 'Usage = ./subnet_selection.sh (subnet) (input file) (output file)'
  exit
fi

SUBNET=$1
FILE=$2
OUT=$3

echo 'Targeted subnet = '$SUBNET

grep -o '.*"tk_client_ip":"'$SUBNET'.*' $FILE >> $OUT
```

Listing B.6 – Script permettant la sélection d'un sous-réseau (Section 4.4.4) (Bash)

```
#!/bin/bash
# Script infecting the log with 4 APT
# author : Thomas Gilon (2017)

# periodic (by hour)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0.out -o /mnt/
    DATA/data/log_files/2014_10_1-10/192.168.129.0_infected_1.out -d APT.FINDME_1.
    apt -u 192.168.129.119 -t periodic -f json -step 3600000

# periodic (by 12h)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0_infected_1.out
    -o /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0_infected_2.out -d APT.
    FINDME_2.apt -u 192.168.129.107 -t periodic -f json -step 43200000

# traffic (stealthy)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0_infected_2.out
    -o /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0_infected_3.out -d APT.
    FINDME_3.apt -u 192.168.129.114 -t traffic -f json -delta 2000 -duration 10000
     -injection 3 -distance 10800000 -delay 5000

# traffic (intensive)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0_infected_3.out
    -o /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0_infected.out -d APT.
    FINDME_4.apt -u 192.168.129.104 -t traffic -f json -delta 1000 -duration 10000
     -injection 10 -distance 3600000 -delay 5000
```

Listing B.7 – Script utilisé dans le code B.5 permettant l'infection du log de proxy (Section 4.4.4 & 5.3.2) (Bash)

```
#!/bin/bash
# Script analyzing the traffic of the network
# author : Thomas Gilon (2017)

./traffic.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0.out -o /mnt/
    DATA/data/log_files/2014_10_1-10/192.168.129.0.csv -r 1000 -f json &
./traffic.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0_infected_1.out
     -o /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0_infected_1.csv -r 1000
     -f json &
./traffic.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0_infected_2.out
     -o /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0_infected_2.csv -r 1000
     -f json &
./traffic.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0_infected_3.out
     -o /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0_infected_3.csv -r 1000
     -f json &
./traffic.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0_infected.out -
    o /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0_infected.csv -r 1000 -f
    json &
wait

./traffic.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.104.out -o /mnt/
    DATA/data/log_files/2014_10_1-10/192.168.129.104.csv -r 1000 -f json &
./traffic.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.107.out -o /mnt/
    DATA/data/log_files/2014_10_1-10/192.168.129.107.csv -r 1000 -f json &
./traffic.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.114.out -o /mnt/
    DATA/data/log_files/2014_10_1-10/192.168.129.114.csv -r 1000 -f json &
./traffic.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.119.out -o /mnt/
    DATA/data/log_files/2014_10_1-10/192.168.129.119.csv -r 1000 -f json &
wait
```

Listing B.8 – Script utilisé dans le code B.5 permettant l'analyse du trafic (Section 4.4.4) (Bash)

```
#!/bin/bash
# Script infecting the log with 4 traffic APT
# author : Thomas Gilon (2017)

# traffic (by 4h)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0.out -o /mnt/
    DATA/data/log_files/2014_10_1-10/192.168.129.0_infected_traffic_1.out -d APT.
    FINDME_5.apt -u 192.168.129.116 -t traffic -f json -delta 2000 -duration 6000
    -delay 3000 -injection 10 -distance 14400000

# traffic (by 2h)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0
    _infected_traffic_1.out -o /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0
    _infected_traffic_2.out -d APT.FINDME_6.apt -u 192.168.129.112 -t traffic -f
    json -delta 2000 -duration 6000 -delay 3000 -injection 10 -distance 7200000

# traffic (by 1h)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0
    _infected_traffic_2.out -o /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0
    _infected_traffic_3.out -d APT.FINDME_7.apt -u 192.168.129.101 -t traffic -f
    json -delta 2000 -duration 6000 -delay 3000 -injection 10 -distance 3600000

# traffic (by 30min)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0
    _infected_traffic_3.out -o /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0
    _infected_traffic.out -d APT.FINDME_8.apt -u 192.168.129.109 -t traffic -f
    json -delta 2000 -duration 6000 -delay 3000 -injection 10 -distance 1800000
```

Listing B.9 – Script utilisé dans le code B.5 permettant l'infection du log de proxy par des APT basées sur le flux de données (Section 5.6) (Bash)

```
#!/bin/bash
# Script infecting the log with 4 periodic APT
# author : Thomas Gilon (2017)

# periodic (by 24h)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0.out -o /mnt/
    DATA/data/log_files/2014_10_1-10/192.168.129.0_infected_periodic_1.out -d APT.
    FINDME_9.apt -u 192.168.129.109 -t periodic -f json -step 86400000

# periodic (by 12h)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0
    _infected_periodic_1.out -o /mnt/DATA/data/log_files/2014_10_1
    -10/192.168.129.0_infected_periodic_2.out -d APT.FINDME_10.apt -u
    192.168.129.101 -t periodic -f json -step 43200000

# periodic (by 6h)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0
    _infected_periodic_2.out -o /mnt/DATA/data/log_files/2014_10_1
    -10/192.168.129.0_infected_periodic_3.out -d APT.FINDME_11.apt -u
    192.168.129.112 -t periodic -f json -step 21600000

# periodic (by 1h)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.129.0
    _infected_periodic_3.out -o /mnt/DATA/data/log_files/2014_10_1
    -10/192.168.129.0_infected_periodic.out -d APT.FINDME_12.apt -u
    192.168.129.116 -t periodic -f json -step 3600000
```

Listing B.10 – Script utilisé dans le code B.5 permettant l'infection du log de proxy par des APT périodiques (Section 5.7) (Bash)

```
#!/bin/bash
# Script infecting the log with 4 periodic APT and 4 traffic APT
# author : Thomas Gilon (2017)

# periodic (by 24h)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.120.0.out -o /mnt/
    DATA/data/log_files/2014_10_1-10/192.168.120.0_infected_1.out -d APT.FINDME_9.
    apt -u 192.168.120.145 -t periodic -f json -step 86400000

# periodic (by 12h)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.120.0_infected_1.out
    -o /mnt/DATA/data/log_files/2014_10_1-10/192.168.120.0_infected_2.out -d APT.
    FINDME_10.apt -u 192.168.120.129 -t periodic -f json -step 43200000

# periodic (by 6h)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.120.0_infected_2.out
    -o /mnt/DATA/data/log_files/2014_10_1-10/192.168.120.0_infected_3.out -d APT.
    FINDME_11.apt -u 192.168.120.131 -t periodic -f json -step 21600000

# periodic (by 1h)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.120.0_infected_3.out
    -o /mnt/DATA/data/log_files/2014_10_1-10/192.168.120.0_infected_4.out -d APT.
    FINDME_12.apt -u 192.168.120.116 -t periodic -f json -step 3600000

# traffic (by 4h)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.120.0_infected_4.out
    -o /mnt/DATA/data/log_files/2014_10_1-10/192.168.120.0_infected_5.out -d APT.
    FINDME_5.apt -u 192.168.120.103 -t traffic -f json -delta 2000 -duration 6000
    -delay 3000 -injection 10 -distance 14400000

# traffic (by 2h)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.120.0_infected_5.out
    -o /mnt/DATA/data/log_files/2014_10_1-10/192.168.120.0_infected_6.out -d APT.
    FINDME_6.apt -u 192.168.120.125 -t traffic -f json -delta 2000 -duration 6000
    -delay 3000 -injection 10 -distance 7200000

# traffic (by 1h)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.120.0_infected_6.out
    -o /mnt/DATA/data/log_files/2014_10_1-10/192.168.120.0_infected_7.out -d APT.
    FINDME_7.apt -u 192.168.120.150 -t traffic -f json -delta 2000 -duration 6000
    -delay 3000 -injection 10 -distance 3600000

# traffic (by 30min)
./infect.sh -i /mnt/DATA/data/log_files/2014_10_1-10/192.168.120.0_infected_7.out
    -o /mnt/DATA/data/log_files/2014_10_1-10/192.168.120.0_infected.out -d APT.
    FINDME_8.apt -u 192.168.120.148 -t traffic -f json -delta 2000 -duration 6000
    -delay 3000 -injection 10 -distance 1800000
```

Listing B.11 – Script utilisé dans le code B.5 permettant l'infection du log de proxy dans le nouveau scénario (Section 5.8) (Bash)

Bibliographie

[1] Alexstanbury. Plugin for Chartist.js allowing you to add a title to an axis. En ligne `https://github.com/alexstanbury/chartist-plugin-axistitle`, consulté le 09-05-2017.

[2] API123. API documentation as a service. En ligne `http://api123.io/`, consulté le 09-05-2017.

[3] Ariel E Bayá and Pablo M Granitto. Penalized K-Nearest-Neighbor-Graph Based Metrics for Clustering French Argentine International Center for Information and Systems Sciences. *Sciences-New York*, pages 1–16, 2000.

[4] Steven Black. Merged Collection of Hosts from Reputable Sources. En ligne `https://raw.githubusercontent.com/StevenBlack/hosts/master/hosts`, consulté le 26-02-2017.

[5] J. A. Bondy and U. S. R. Murty. *Graph Theory With Applications*. 1976.

[6] Bootstrap. Bootstrap. En ligne `https://getbootstrap.com/`, consulté le 09-05-2017.

[7] CERT.be. Cyber Emergency Team fédérale. En ligne `https://www.cert.be`, consulté le 08-05-2017.

[8] CERT.be. Rapport annuel de CERT.be. 2015.

[9] Francesco Chemolli. Squid-Cache Wiki. En ligne `http://wiki.squid-cache.org/`, consulté le 09-05-2017.

[10] Cobertura. Cobertura, A code coverage utility for Java. En ligne `https://cobertura.github.io/cobertura/`, consulté le 09-05-2017.

[11] Coveralls. Coveralls, Deliver better code. En ligne `https://coveralls.io/`, consulté le 09-05-2017.

[12] Cyber Security Belgium (CCB). Center for Cyber Security Belgium. En ligne `http://www.ccb.belgium.be`, consulté le 08-05-2017.

[13] Miguel de Bruycker. NIAS16 - LTC (rtd) Miguel de Bruycker. En ligne `https://www.youtube.com/watch?v=sKYVX3-ZHYI`, consulté le 09-05-2017.

[14] Thibault Debatty. java-graphs. En ligne `https://github.com/tdebatty/java-graphs`, consulté le 09-05-2017, 2017.

[15] Thibault Debatty, Pietro Michiardi, and Olivier Thonnard. Scalable Graph Building from Text Data. *Jmlr*, (36) :120–132, 2014.

[16] Dell SecureWorks. Advanced Threat Protection with Dell SecureWorks Security Services. 2012.

[17] Dell SecureWorks. Lifecycle of an Advanced Persistent Threat. 2012.

[18] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. *Proceedings of the 20th international conference on World wide web - WWW '11*, page 577, 2011.

[19] European Defence Agency (EDA). Cyber Defence. En ligne https://www.eda.europa.eu/what-we-do/activities/activities-search/cyber-defence, consulté le 10-05-2017, 2015.

[20] European Union Agency for Network and Information Security (ENISA). CERT-EU. En ligne https://www.enisa.europa.eu/topics/national-csirt-network/capacity-building/european-initiatives/cert-eu, consulté le 10-05-2017.

[21] Jerome Fan, Suneel Upadhye, and Andrew Worster. Understanding receiver operating characteristic (ROC) curves. 8(1) :19–20, 2006.

[22] Five Command. Advanced Persistent Threats : A Decade in Review. (June) :1–13, 2011.

[23] Joseph Gardiner, Marco Cova, and Shishir Nagaraja. Command & Control : Understanding, Denying and Detecting. *arXiv.org*, cs.CR(February) :1136, 2014.

[24] Git. git –fast-version-control. En ligne https://git-scm.com/, consulté le 09-05-2017.

[25] GitHub. Built for developers. En ligne https://github.com/, consulté le 09-05-2017.

[26] Google. About DNS. En ligne https://support.google.com/domains/answer/3251148?hl=en, consulté le 20-05-2017.

[27] Grunt. Grunt, The JavaScript Task Runner. En ligne https://gruntjs.com/, consulté le 09-05-2017.

[28] Mattijs Hertsens. *Grafen gebaseerde APT detectie*. Koninklijke Militaire School, 2015.

[29] Amos Jeffries. Feature : Customizable Log Formats. En ligne http://wiki.squid-cache.org/Features/LogFormat, consulté le 09-05-2017.

[30] Hannes Kamecke. Zoom Plugin for Chartist.js. En ligne https://github.com/hansmaad/chartist-plugin-zoom, consulté le 09-05-2017.

[31] Kaspersky Lab Global Research and Analysis Team (GReAT). ProjectSauron : top level cyber-espionage platform covertly extracts encrypted government comms. En ligne https://securelist.com/analysis/publications/75533/faq-the-projectsauron-apt/, consulté le 09-05-2017.

[32] Gion Kunz. Chartist - API Documentation. En ligne https://gionkunz.github.io/chartist-js/api-documentation.html, consulté le 09-05-2017.

[33] Gion Kunz. Simple responsive charts. En ligne https://github.com/gionkunz/chartist-js/, consulté le 09-05-2017.

[34] La Défense. La Défense recrute des experts en cyber sécurité. En ligne http://www.mil.be/fr/communiques-presse/la-defense-recrute-des-experts-en-cyber-securite, consulté le 08-05-2017.

[35] Wim Mees. *IN014 : Technology Solutions*. Royal Military Academy, 2016.

[36] Wim Mees and Thibault Debatty. Multi-agent system for APT detection. *Proceedings - IEEE 25th International Symposium on Software Reliability Engineering Workshops, ISSREW 2014*, pages 401–406, 2014.

[37] Wim Mees and Bart Scheers. *TE013 : Data Networks Part I*. Royal Military Academy, 2011.

[38] P. Mockapetris. RFC 1035 : Domain Names - Concepts and Facilities. En ligne https://tools.ietf.org/html/rfc1034, consulté le 11-05-2017, 1987.

[39] MojoHaus. Cobertura Maven Plugin. En ligne http://www.mojohaus.org/cobertura-maven-plugin/, consulté le 26-05-2017.

[40] MojoHaus. FindBugs Maven Plugin. En ligne https://gleclaire.github.io/findbugs-maven-plugin/index.html, consulté le 26-05-2017.

[41] NATO Communications and Information Agency (NCI). Cyber Security. En ligne https://www.ncia.nato.int/Our-Work/Pages/Cyber-Security.aspx, consulté le 10-05-2017.

[42] NATO Cooperative Cyber Defence Centre of Excellence (CCDCOE). About Cyber Defence Centre. En ligne https://ccdcoe.org/about-us.html, consulté le 10-05-2017.

[43] Npm. Grunt plugin for Eslint. En ligne https://www.npmjs.com/package/gruntify-eslint, consulté le 09-05-2017.

[44] Npm. Grunt task for checking JavaScript Code Style with jscs. En ligne https://www.npmjs.com/package/grunt-jscs, consulté le 09-05-2017.

[45] Npm. Lint html files with htlmlint. En ligne https://www.npmjs.com/package/grunt-htmlhint, consulté le 09-05-2017.

[46] Npm. Validate files with JSHint. En ligne https://www.npmjs.com/package/grunt-contrib-jshint, consulté le 09-05-2017.

[47] Oracle. Java HotSpot^{TM} Virtual Machine Performance Enhancements. En ligne https://docs.oracle.com/javase/7/docs/technotes/guides/vm/performance-enhancements-7.html, consulté le 09-05-2017.

[48] Oracle. Primitive Data Types. En ligne https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html, consulté le 09-05-2017.

[49] Organisation du Traité de l'Atlantique Nord (OTAN). La cyberdéfense. En ligne http://www.nato.int/cps/fr/natohq/topics{_}78170.htm, consulté le 10-05-2017.

[50] Markus Padourek. Tooltip plugin for Chartist.js. En ligne https://github.com/Globegitter/chartist-plugin-tooltip, consulté le 09-05-2017.

[51] Wolfgang Röhrig and Wg Cdr Rob Smeaton. Cyber Security And Cyber Defence In The European Union. (December) :23–27, 2013.

[52] Keijo Ruohonen. Graph Theory. 2013.

[53] StackOverflow. Iterating through a Collection, avoiding ConcurrentModificationException when removing in loop. En ligne https://stackoverflow.com/questions/223918/iterating-through-a-collection-avoiding-concurrentmodificationexception-when-re, consulté le 09-05-2017.

[54] StackOverflow. Sort a Map<Key, Value> by values (Java). En ligne https://stackoverflow.com/questions/109383/sort-a-mapkey-value-by-values-java, consulté le 09-05-2017.

[55] Symantec. What is a Zero-Day Vulnerability ? En ligne http://www.pctools.com/security-news/zero-day-vulnerability/, consulté le 08-05-2017.

[56] Symantec. Internet Security Threat Report. (April), 2017.

[57] The Apache Software Foundation. Apache Maven Project. En ligne https://maven.apache.org/, consulté le 09-05-2017.

[58] The Apache Software Foundation. Available Plugins. En ligne https://maven.apache.org/plugins/index.html, consulté le 26-05-2017.

[59] Travis CI. Test and Deploy with Confidence. En ligne https://travis-ci.org/, consulté le 09-05-2017.

[60] Steven Vandeput. La vision stratégique pour la Défense. 2016.

[61] Wikipédia. GitHub. En ligne https://en.wikipedia.org/wiki/GitHub, consulté le 09-05-2017.

[62] Robin J. Wilson. *Introduction to Graph Theory*. 1996.