Executive Certificate Big Data

Détection d'intrusion réseau à l'aide de l'apprentissage automatique

Ahmed Mekaouar

Octobre 2018

Résumé

Le réseau est de plus en plus la cible de cyberattaques avec des acteurs et des motivations diverses et variées. Les systèmes de détection d'intrusion réseau sont confrontés à d'énormes défis vu l'augmentation continue des débits et l'utilisation du chiffrement. De nouvelles approches, basées sur l'analyse du flux IP, ont alors vu le jour et cherchent à remédier aux difficultés des systèmes classiques.

Ce projet a comme objectif d'explorer la détection d'intrusion basée sur le flux IP à travers le jeu de données CICIDS2017 et ceci à l'aide de l'apprentissage automatique. L'apprentissage supervisé a été appliqué, en un premier temps, à l'aide d'un arbre de décision afin de concevoir un système de classification des intrusions réseau connues. Ce système permet de classifier les attaques connus avec une F-mesure de l'ordre de 97% mais il ne permet pas la détection de nouvelles attaques. L'apprentissage non supervisé a alors été employé, à travers l'algorithme SVM à une classe, afin de créer un système de détection d'anomalies. Ce système est particulièrement performant pour les attaques de type DOS. L'attaque *DoS Slowhttptest* peut être détectée à 98% mais avec un taux de faux positifs de 10%. Les résultats sont prometteurs même s'il y a encore du chemin pour que de cette approche soit adoptée à grande échelle.

Remerciements

Je tiens à remercier Madame Marie Aude Aufaure, la responsable de la formation Executive Certificate Big Data, ainsi que tous les intervenants de cette formation pour la qualité et la richesse du contenu proposé. Ils ont tous contribué à ce travail en me transmettant à la fois leur passion et leurs vastes connaissances des technologies liées au big data.

Table des matières

1 Introduction

Depuis son apparition et son adoption, la connectivité internet n'a pas cessé de prendre du terrain et de façonner la vie des individus. Son taux de pénétration est respectivement de 95% et 85,2 % en Amérique du nord et en Europe[1]. Le trafic IP mondial augmente également. Il était de l'ordre de 96 exaoctet (10^{18}) en 2016 et il est prévu que ce trafic augmente de 24%, tous les ans, les prochaines années (voir la figure 1).

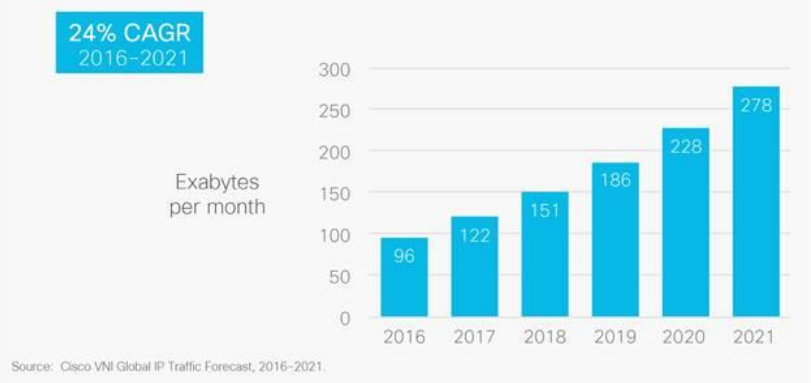

Figure 1- Croissance du trafic IP global

Les cyberattaques se sont en même temps multipliées et diversifiées. Dans un sondage réalisé en France par Bessé et PwC[2], 76% des ETI (Entreprise de Taille Intermédiaire) sondées déclarent avoir subi au moins une cyberattaque. L'évolution rapide du monde de la cybercriminalité est un enjeu majeur pour la plupart des entreprises. Celles-ci ont besoin de revoir sans cesse leur politique de sécurité et d'adopter de nouvelles approches, telles que la défense en profondeur (*Defense-In-Depth*), afin de mitiger les menaces. Cette partie introduit les différents types de cyberattaques, les systèmes de détection d'intrusion ainsi que le flux IP qui sert aux nouvelles approches de détection.

1.1 Les cyberattaques

Une cyberattaque vise à menacer l'un des trois piliers de la sécurité, à savoir : confidentialité, intégrité et disponibilité (connus sous le nom de triade CIA : *Confidentiality, Integrity, Availability*)

[1]https://www.internetworldstats.com/stats.htm

[2]https://www.pwc.fr/fr/espace-presse/communiques-de-presse/2018/mars/76-pourcent-des-eti-ont-deja-subi-une-attaque-cyber.html

Ces attaques peuvent être classées selon leurs modes opératoires ou leurs conséquences sur le système. Voici un aperçu des attaques connues:

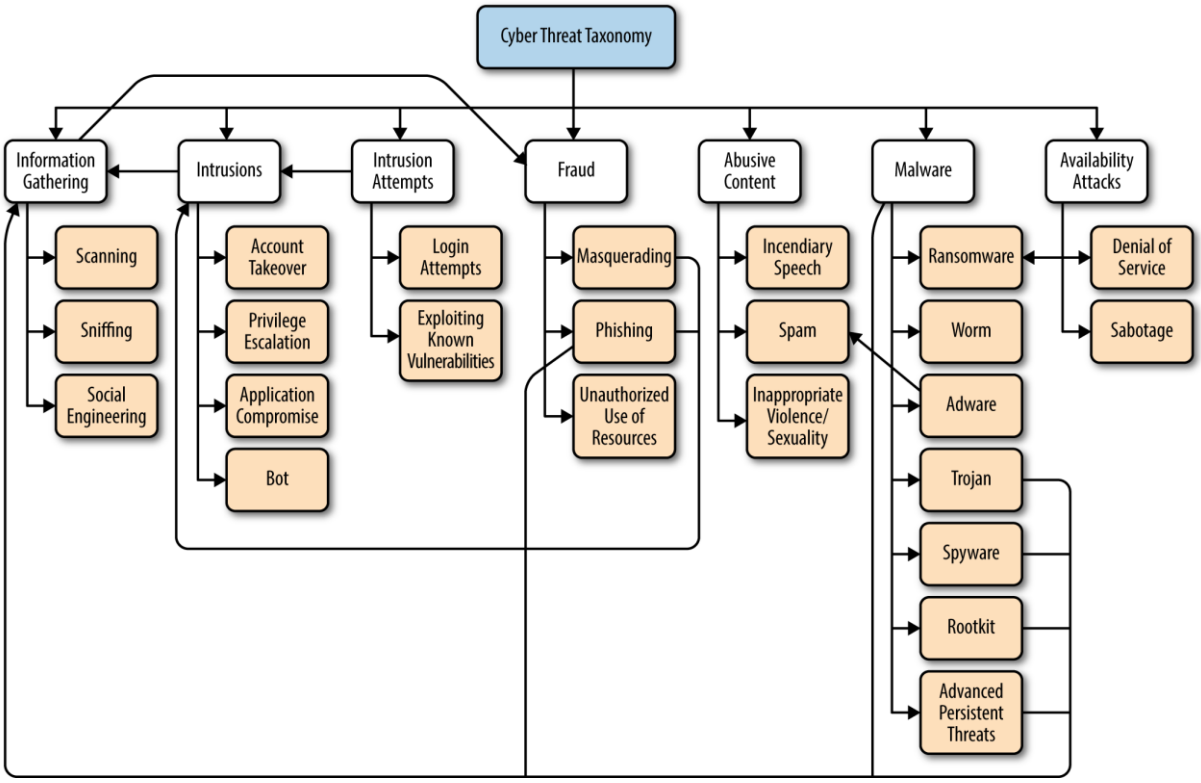

Figure 2-Classification des attaques (Freeman & Chio, 2018)

La classification des attaques est un sujet très large. Ce projet se limite à définir les catégories d'attaques présentes dans le jeu de données.

1.1.1 Attaques par déni de service

Un déni de service (DoS : *Denial of Service*) a lieu lorsque les données ou les applications sont inaccessibles aux utilisateurs légitimes. Le déni de service porte alors atteinte à la disponibilité, troisième élément de la triade CIA. Les attaques DoS exploitent généralement des vulnérabilités au niveau de TCP/IP. La plupart de ces attaques procèdent de manière à allouer énormément de ressources afin de saturer le système. On parle également de DDoS (*Distributed Denial of Service*) quand l'attaque provient de plusieurs sources en même temps. Les DDoS amplifient l'effet d'une attaque DoS par le nombre de sources qui lancent cette attaque. Le jeu de données du projet contient cinq attaques par déni de service : *DoS Hulk*, *DoS slowloris*, *DoS Slowhttptest*, *DoS GoldenEye* et *DDoS*.

1.1.2 Attaques web

Les attaques web exploitent des vulnérabilités au niveau applicatif. Elles sont alors dues à des défauts dans la conception et le développement des applications web. Ces attaques peuvent compromettre la confidentialité des données qui sont en clair au niveau de l'application. Le jeu de donnée du projet contient trois attaques de ce type: *Web Attack Brute Force*, *Web Attack XSS* et *Web Attack Sql Injection*.

1.1.3 Attaques de mots de passe

Les attaques de mots de passe ont pour objectif de trouver les mots de passe qui permettent d'accéder à des services. Ces attaques procèdent généralement par force brute en utilisant des dictionnaires ou des générateurs d'identifiants. Le jeu de donnée du projet contient 2 attaques de ce type : *FTP-Patator* et *SSH-Patator*.

1.1.4 Autres attaques

Voici un descriptif des autres attaques présentes dans le jeu de données.

- *PortScan* : C'est une attaque de reconnaissance du système et vise à connaitre les ports ouverts sur un serveur ou une machine donnée.
- *Heartbleed* : Cette attaque exploite une vulnérabilité de la bibliothèque openSSL et peut permettre à un attaquant de compromettre la confidentialité d'une communication chiffrée par TLS.
- *Infiltration* : Il s'agit d'une attaque qui vise à s'infiltrer dans le système attaqué.
- *Bot* : Cette attaque correspond à un réseau de machines infectées et contrôlées par un serveur central. Ce serveur commande les machines pour lancer tout type d'attaques.

1.2 Les systèmes de détection d'intrusion

Les systèmes de détection d'intrusion IDS (*Intrusion Detection System*) sont une ligne de défense importante afin de détecter les attaques exposées précédemment. Il est possible de classer ces systèmes selon plusieurs critères tels que la cible analysée ou le type de détection. On parle alors de HIDS (*Host-IDS*) pour les systèmes installés au niveau d'une machine et de NIDS (*Network-IDS*) lorsque le trafic est inspecté au niveau du réseau. La détection peut être basée sur la connaissance préalable des attaques, en faisant appel à une base de signatures. Elle peut aussi être basée sur la connaissance de l'état normal du système et procéder par détection d'anomalies.

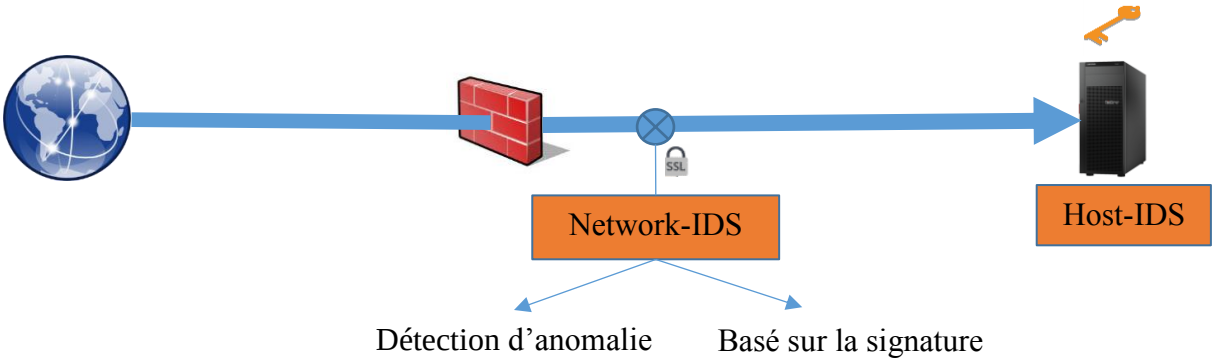

Figure 3- Système de détection d'intrusion

Les NIDS les plus connus, comme Snort, font de l'inspection de paquets afin d'identifier des signatures d'attaques répertoriées ou de détecter des anomalies. Cette analyse des paquets exige un niveau de performance de plus en plus élevé vu l'augmentation des débits, qui sont de l'ordre du Gb/s pour les connections à base de fibre optique. Les NIDS souffrent également de leur incapacité à analyser le trafic réseau chiffré, qui est délégué au HIDS (Host-based IDS). Ce trafic chiffré augmente d'année en année suite à la large adoption de HTTPS. Les chiffres d'utilisation de chrome[3] montrent que 80% des requêtes en France et 82% des requêtes aux États-Unis se font sont HTTPS en juin 2018 contre respectivement 49 % et 55 % deux ans auparavant. Afin de contourner ces difficultés, d'autres approches de détection d'intrusion ont vu le jour. Elles sont basées sur les caractéristiques du flux IP. Elles permettent ainsi de réduire considérablement la quantité d'information à analyser et ne nécessitent pas le déchiffrement des données. Voici par la suite la définition du flux IP qui servira de base pour comprendre les nouvelles approches d'IDS.

1.3 Le flux IP

En reprenant la définition qui figure dans la RFC 7011 : « Le flux est un ensemble de paquets ou de trames qui passent à travers un point d'observation dans le réseau pendant un intervalle de temps. Tous les paquets appartenant à un flux particulier ont des propriétés communes ». La RFC propose les propriétés suivantes: IP source, Port source, IP destination, Port destination et le Protocol. Voici, par la suite, l'approche adoptée dans ce projet afin de construire un système de détection d'intrusion à l'aide de ce flux IP et de ses caractéristiques.

[3] https://transparencyreport.google.com/https/overview

2 Construction et réalisation du projet

Le but de ce projet est d'explorer la détection d'intrusion basée sur le flux IP à l'aide de l'apprentissage automatique. Il essaie alors de répondre aux deux questions suivantes :

Question 1 : Dans quelle mesure, l'apprentissage **supervisé** peut permettre de détecter et de classifier des **intrusions réseau connues** du système ?

Question 2 : En cas, d'**intrusions inconnues**, quelle serait l'efficacité de l'apprentissage **non supervisé** afin de détecter ces attaques ?

Le premier enjeu de ce projet consiste alors à trouver un jeu de données labélisées afin de répondre à ces questions. Le choix s'est porté sur le jeu de données CICIDS2017 pour les raisons exposées ci-dessous.

2.1 Le jeu de données CICIDS2017

La recherche dans le domaine des systèmes de détection d'intrusion basés sur l'apprentissage automatique souffre de la rareté de jeux de données fiables afin de permettre de valider les performances dans les conditions réelles. Les données réseau, tels que les logs ou les exports de flux, sont extrêmement sensibles vu qu'elles dévoilent la topologie du réseau. Cette topologie est extrêmement utile pour les cybercriminels. Les rares structures, qui partagent leurs données réseau, ont recours à l'anonymisation afin de garantir leur sécurité. Parmi les jeux de données connus, on peut citer KDD'99, NSL_KDD (version améliorée de KDD'99) et DARPA 98. Ces jeux de données sont assez anciens et souffrent d'un manque diversité de trafic et d'attaques. Dans l'objectif de pallier aux faiblesses des jeux de données existants, une équipe de chercheurs, au sein du *Canadian Institute for Cybersecurity* (CIC), a réalisé le jeu de données CICIDS2017 (Sharafaldin, Lashkari, & Ghorbani, 2018). Ces chercheurs ont évalué ce jeu de données en classifiant le flux IP à l'aide de différents algorithmes d'apprentissage supervisé et ils ont obtenus de très bons résultats.

Ce jeu de données est composé d'une capture réseau complète sur 5 jours (entre le 3 juillet et le 7 juillet 2017) et comporte du trafic normal ainsi que 14 types d'attaques réseau. L'architecture du réseau utilisé ainsi que les différentes attaques sont détaillés sur cette page web du CIC: http://www.unb.ca/cic/datasets/ids-2017.html.

Ce trafic réseau est présent sous deux formats différents : des fichiers pcap (*packet capture*) qui font 48Go et des fichiers CSV (extraction des flux IP) labellisés qui font 1Go. Ce

sont ces fichiers CSV qui ont été utilisés dans le cadre de ce projet. Ils ont été extraits à partir des captures *pcap* en utilisant l'outil CICFlowMeter (Gerard Draper-Gil, Mamun, & Ghorbani, 2016). Cet outil génère un flux IP bidirectionnel où le premier paquet détermine la source et la destination. Le jeu de données contient 83 variables dont 75 qui caractérisent le flux IP (durée, nombre de paquets, nombre de bytes, longueur de paquets etc…). La plupart sont calculées séparément dans les deux directions et permettent ainsi de décrire complètement un flux IP bidirectionnel. La liste complète des variables est disponible en **Annexe 1**.

Ces variables ont déjà montré leur efficacité pour détecter et classifier différents flux VPN (Gerard Draper-Gil, Mamun, & Ghorbani, 2016). Ces mêmes variables ont également été utilisées afin de détecter et de classifier un flux de type TOR et ont montré des résultats satisfaisants (Lashkari, Draper-Gil, & Ghorbani, 2017).

Le paragraphe suivant détaille l'approche adoptée afin d'exploiter ce jeu de données pour répondre aux questions de ce projet.

2.2 Conception du projet

Le jeu de données labélisées CICIDS2017 distingue 15 classes : flux IP normal et flux IP correspondant à 14 types d'attaques. L'apprentissage supervisé a été appliqué, en un premier lieu, afin de produire un modèle qui détecte et classe les différentes attaques connues. L'apprentissage non supervisé a été par la suite utilisé afin produire un modèle qui détecte des attaques inconnues du système.

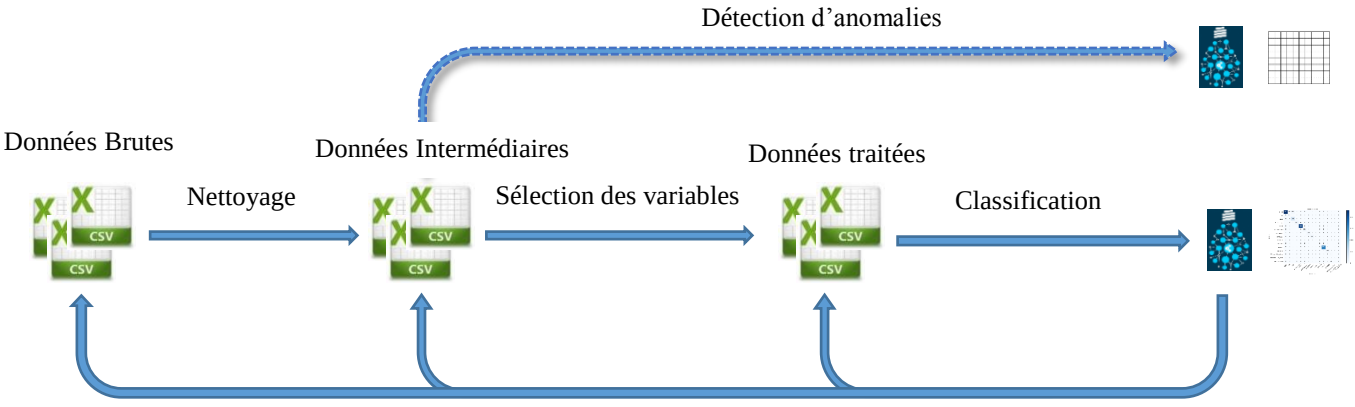

Figure 4- Conception du système de détection d'intrusion

Une première phase consiste à préparer les données. A l'issue de cette étape, les données intermédiaires sont sauvegardées sous format CSV. Les données subissent une autre transformation suite à la sélection des variables. Les données ainsi générées servent à

l'apprentissage automatique. L'apprentissage non supervisé se base directement sur les données intermédiaires. Il est à noter que les données de l'une des 14 attaques, a été séparé du reste et sauvegardée sous le fichier novelty.csv dès la fin de la première phase. L'attaque en question n'est pas du tout connue par les systèmes de classification entrainés.

2.3 Gestion du projet

Ce projet a été réalisé en mode agile, c'est-à-dire avec des ressources et temps fixes mais avec un périmètre variable. Un Produit Minimum Viable (MVP : *Minimum Viable Product*) a été défini dès la phase de planification du projet. Il correspond, dans le cas de ce projet, à la conception d'un modèle de classification des intrusions, à la réflexion autour du passage à l'échelle et à la rédaction du rapport. Ce MVP a été par la suite traduit en « backlog MPV » afin de planifier l'activité de développement. Un « backlog d'évolutions » a également été prévu afin d'alimenter de nouvelles activités une fois le MVP achevé.

Backlog MVP	**Backlog d'évolutions**
Exploration des données Nettoyage des données Classification en apprentissage supervisé Application temps réel sur PC Architecture pour le passage à l'échelle Rédaction du rapport	Détection d'anomalie en apprentissage non supervisé Préparation d'un environnement pour le passage à l'échelle Implémentation pour le passage à l'échelle

Tableau 1- Backlog d'activités

2.3.1 Planning Initial

Le backlog MVP a été planifié sur 4 sprints en 10 semaines avec l'objectif d'avoir un livrable à la fin de chaque sprint (sauf pour le sprint 1). Voici le planning en question :

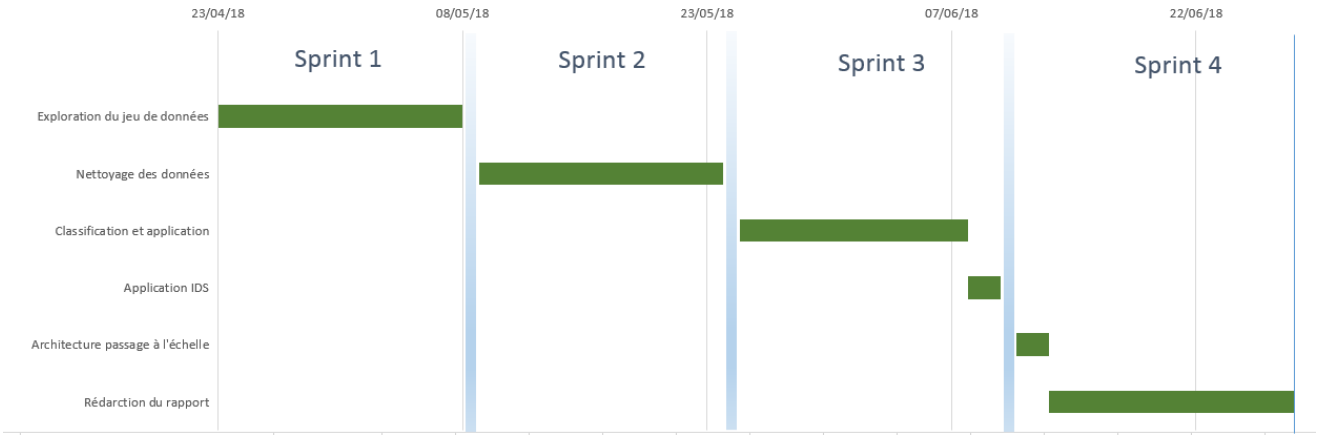

Figure 5-Planning Initial MVP

2.3.2 Planning effectif

Dans la pratique, les développements produits lors des 3 premiers sprints se sont construits de manière assez itérative. Le code, en sortie du sprint 1, a continué d'évoluer jusqu'à la fin du sprint 3 dans un esprit d'expérimentation et d'amélioration.

A l'issue du sprint 3, le choix a été fait d'explorer l'apprentissage non supervisé pour la détection d'anomalies. Le sprint 5 a été également rajouté et il a comme principal objectif de mettre en place un environnement pour les développements relatifs au passage à l'échelle.

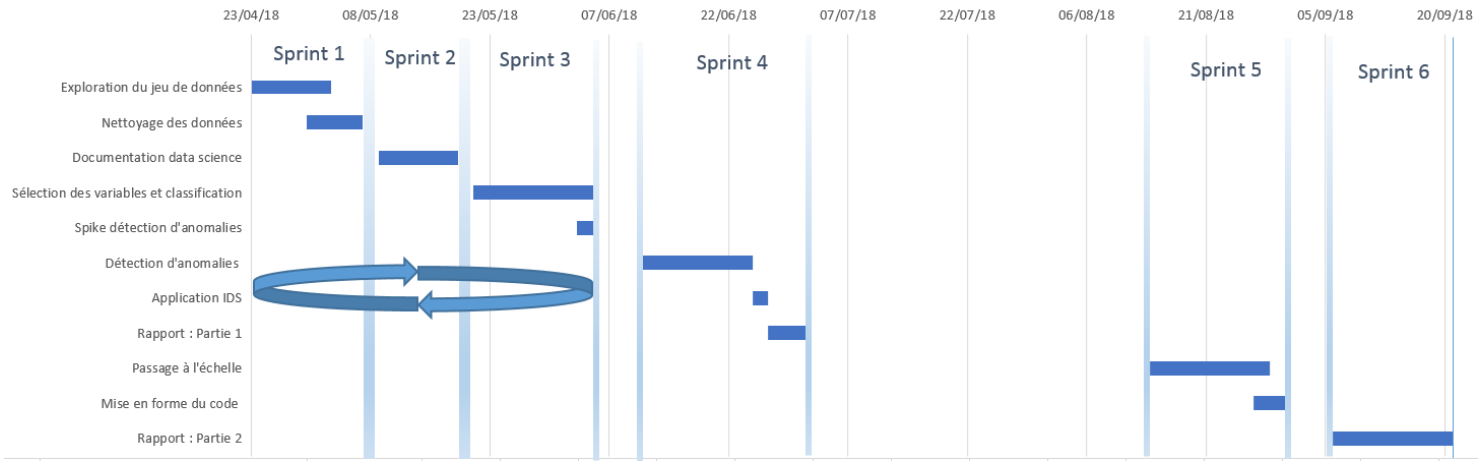

Figure 6- Planning effectif

Une description des livrables des sprints 1, 3, 4 et 5 est détaillée dans l'**Annexe 2**. Ces livrables incluent les jupyter notebooks, l'application IDS et les différents modèles.

2.4 Environnement de développement

Le développement a été réalisé à l'aide d'un PC windows7 avec 8Go de RAM et un processeur i7 quadricoeur à 2,6GHz. Cette configuration a permis d'atteindre la plupart des objectifs mais elle atteint vite ses limites avec certains algorithmes d'apprentissage automatique qui consomment beaucoup de RAM ou de CPU.

Le choix s'est porté sur Python vu la richesse des librairies existantes. Les développements des 4 premiers sprints ont été réalisés avec Pandas, Numpy et Scikit-learn. Le code développé pour le passage à l'échelle a été réalisé en pyspark en se basant sur les librairies Spark SQL et Spark MLlib. Spark a été installé sur une machine virtuelle Ubuntu qui tourne sur le PC mentionné ci-dessus. L'ensemble du logiciel a été réalisé en mode jupyter notebook afin de profiter du côté interactif de l'exécution.

La partie suivante décrit les étapes de la conception du système en allant de l'exploration des données à la création des modèles prédictifs.

3 Exploration et traitement des données

L'exploration du jeu de données est une étape incontournable qui permet d'analyser et de comprendre les données. Elle est réalisée essentiellement grâce à la librairie *pandas*. Parmi les fonctions intéressantes, on peut citer *info()* qui permet de ressortir les informations de base, comme le nombre d'échantillons ou les types de variables. Une fonction, non moins intéressante, est *describe()* qui fournit les indicateurs statistiques de bases qui décrivent les données. Le champ *std* (écart-type), par exemple, permet d'identifier les variables avec une variance nulle. De telles variables sont inutiles et sont donc éliminées des données.

Les paragraphes suivants détaillent les deux objectifs de cette phase, à savoir : le nettoyage et la compréhension des données.

3.1 Valeurs manquantes et aberrantes

L'exploration du jeu de données montre l'existence de certaines valeurs manquantes (NaN par exemple). De telles valeurs peuvent causer une mauvaise inférence des types des variables par *pandas* (la variable *FlowBytesPs* a été inférée comme « objects » par exemple). Ces valeurs ne sont également pas tolérées par les algorithmes implémentés dans Scikit-learn. Ces derniers exigent des valeurs numériques. Une analyse a été effectuée et tous les échantillons avec des valeurs manquantes ont été supprimées vu leur nombre faible et leur appartenance à la classe majoritaire.

L'exploration montre également l'existence de valeurs aberrantes. Le jeu de données est constitué majoritairement de valeurs numériques positives tels que : durées, tailles de paquets, nombres de paquets etc. Ceci dit, l'analyse du champs *min* de la fonction *describe()* montre l'existence de valeurs négatives dans certaines variables. Ces valeurs ont été analysées afin de déterminer leur nombre et leur classe d'appartenance. Les valeurs ont été traitées, soit en supprimant les échantillons, soit en remplaçant les valeurs aberrantes par des valeurs acceptables.

3.2 Analyse du flux IP

La phase d'exploration consiste aussi à analyser les données afin de les cerner. Il est utile, par exemple, de connaitre la distribution du flux IP entre les différents labels afin de savoir s'il

y a un déséquilibre éventuel entre les classes. Le tableau 2 montre que le jeu de donnée représente un déséquilibre fort entre le flux IP normal (*BENIGN*) et attaques. Ce déséquilibre doit être tenu en compte vu que la majorité des algorithmes de classification sont sensibles à un tel problème.

Label	Nombre	%
BENIGN	2359289	83.3452
DoS Hulk	231073	8.1630
PortScan	158930	5.6144
DDoS	41835	1.4779
DoS GoldenEye	10293	0.3636
FTP-Patator	7938	0.2804
SSH-Patator	5897	0.2083
DoS slowloris	5796	0.2048
DoS Slowhttptest	5499	0.1943
Bot	1966	0.0695
Web Attack Brute Force	1507	0.0532
Web Attack XSS	652	0.0230
Infiltration	36	0.0013
Web Attack Sql Injection	21	0.0007
Heartbleed	11	0.0004

Tableau 2-Répartition du flux IP dans le jeu de donnée

On peut visualiser cette répartition à l'aide matplotlib comme suivant :

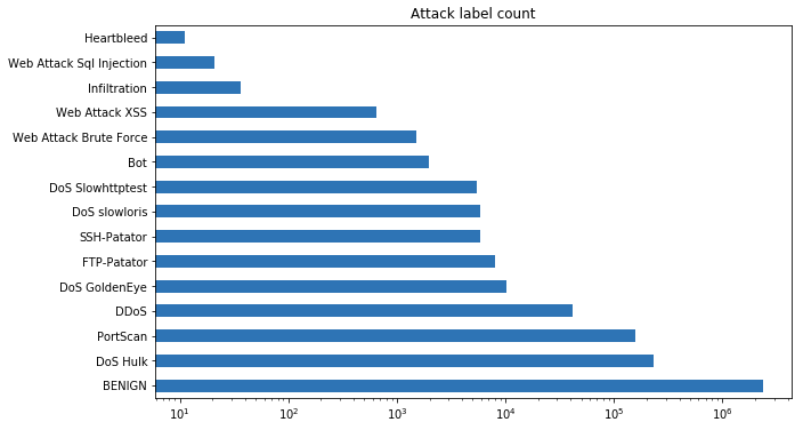

Figure 7- Répartition du flux IP - Echelle logarithmique

Il est également possible d'analyser les attaques afin d'observer, entre autres, l'IP source, l'IP destination, le port source et le port destination. On remarque, par exemple, que la plupart des attaques proviennent de l'adresse IP 172.16.0.1, qui est au final l'adresse IP du pare-feu en mode translation d'adresse (NAT). L'adresse IP source n'est alors pas une variable pertinente pour expliquer la nature du flux IP. Une réflexion a été menée pour toutes les autres variables

nominales et elles s'avèrent toutes inutiles pour la classification. Un tableau récapitulatif de l'analyse du flux IP des attaques est disponible en **Annexe 3**.

3.3 Corrélation des variables

Une fois les variables nominales éliminées, le jeu de données contient uniquement des variables numériques. Il est alors intéressant d'analyser leurs corrélations à l'aide de la fonction *corr()*. Une visualisation grâce à matplotlib, montre que certaines variables sont fortement corrélées.

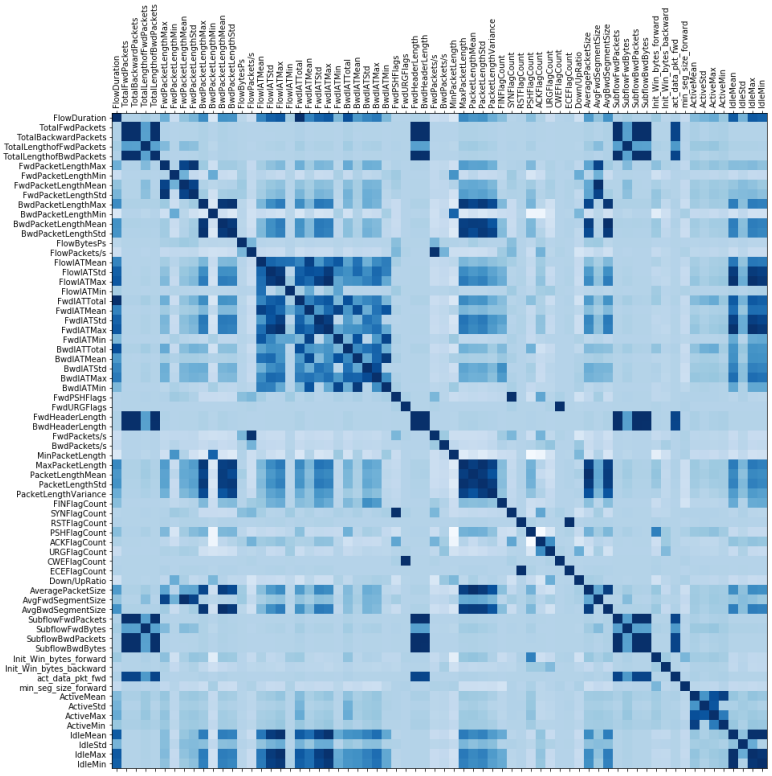

Figure 8-Matrice de corrélation des variables

Il aurait été possible, dès cette phase, de supprimer certaines variables en se basant sur les valeurs de corrélation. Ceci dit, le choix a été fait d'effectuer la sélection des variables uniquement dans le cadre de la classification et ceci via l'algorithme forêts aléatoires. C'est l'objet de la partie suivante.

4 Classification d'attaques à l'aide de l'apprentissage supervisé

A l'issue de la phase précédant, on se retrouve avec un jeu de données labélisées avec 68 variables. Certaines de ces variables sont hautement corrélées comme vu précédemment et il n'est pas sûr qu'elles aient toutes une utilité pour expliquer la nature du flux IP. La première étape consiste alors à procéder à une sélection des variables. Il est possible de faire appel à

plusieurs méthodes afin de remplir cet objectif. Ceci dit, le choix s'est porté sur la sélection via l'algorithme forêts aléatoires.

4.1 Arbres de décisions et forêts aléatoires

L'algorithme forêts aléatoires se base sur l'arbre de décision. Dans Scikit-learn, l'arbre de décision est lui-même basé sur l'algorithme CART (Classification And Regression Trees). CART construit un arbre binaire en séparant chaque nœud suivant une variable et un seuil. On choisit la variable et le seuil qui permettent de minimiser l'indice de diversité de Gini (par défaut). Cet algorithme a comme principal avantage d'être interprétable mais il peut être susceptible de sur-apprendre les données d'entrainement. L'algorithme forêts aléatoires (*Random Forests*) permet de résoudre ce problème en mettant au vote plusieurs arbres de décision entrainés. Il utilise des sous-ensembles aléatoires de données et un échantillon de variables. Une fois entrainé, l'algorithme forêts aléatoires permet d'avoir la liste des variables avec un score d'importance, ce qui explique le choix de cet algorithme pour la sélection des variables.

4.2 Sélection des variables

La sélection des variables permet à la fois d'éviter d'avoir un modèle très complexe et de gagner en temps d'apprentissage. Cette étape nécessite normalement l'avis d'un expert métier capable de juger de la pertinence des variables dans la construction du modèle. Ceci est d'autant plus vrai pour les sujets techniques comme c'est le cas de ce projet. Afin de choisir les variables caractérisant le flux IP, il est indispensable d'avoir une expérience assez pointue de TCP/IP et des attaques réseaux. La multitude des variables à analyser ainsi que des attaques potentielles rend cette tâche difficile. La sélection par algorithme peut aider à résoudre ce problème et permet d'automatiser cette étape et surtout de s'abstraire de l'expertise métier même si une bonne connaissance du sujet reste requise afin de pouvoir faire certains choix.

La sélection des variables à travers l'algorithme forêts aléatoires nécessite de faire une première classification avec cet algorithme. Cette première classification a permis de faire un cycle complet d'apprentissage automatique, à savoir : la séparation en données d'entrainement et de test, la gestion du déséquilibre des classes, l'entrainement du modèle et l'évaluation des performances. A l'issue de ces étapes, on peut récupérer l'attribut « *feature_importances_* » du modèle entrainé. Cet attribut permet de trier les variables par importance. Voici la liste des 10 premiers éléments obtenus :

1) Init_Win_bytes_backward 0.078754
2) Init_Win_bytes_forward 0.035087
3) FwdPacketLengthMax 0.034939
4) min_seg_size_forward 0.031537
5) PacketLengthMean 0.029018
6) SubflowFwdBytes 0.025572
7) BwdPacketLengthMean 0.024790
8) BwdPacketLengthMax 0.024562
9) AvgBwdSegmentSize 0.024307
10) BwdPackets/s 0.023839

La liste complète des variables classées est disponible en **Annexe 4**.

Voici également un graphe avec l'importance des différentes variables.

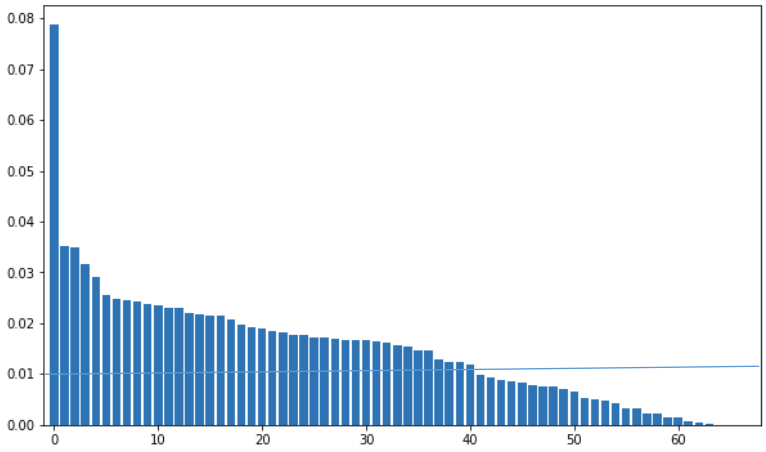

Figure 9- Importance des variables

Ce graphe permet de voir que les valeurs d'importances sont très proches (à l'exception de la 1ère variable). Le seuil d'importance de 0.01 a été choisi afin de sélectionner les variables. Cette sélection est automatisée à l'aide de la fonction *SelectFromModel* de scikit-learn. Ainsi 41 variables ont été retenues pour entrainer le modèle de classification. Mais avant de construire ce modèle, il est important de traiter le problème de déséquilibre de classes afin d'éviter que le modèle sur-apprenne la classe majoritaire, qui correspond au flux IP normal.

4.3 Déséquilibre des classes

Le jeu de donnée présente un fort déséquilibre (voir tableau 2) : 83% des données appartiennent à la classe *BENIGN* et les 14 attaques se partagent les 17% restantes. Il est alors nécessaire de rétablir un équilibre afin d'éviter que le modèle ne soit biaisé par cette répartition des données. Afin de résoudre ce problème, il a fallu faire appel à la fois au sous-échantillonnage de la classe majoritaire et au sur-échantillonnage de toutes les classes minoritaires. En tenant compte de la taille de la RAM, le choix a été fait de sous-échantillonner,

en premier lieu, la classe *BENIGN* à 10% (à l'aide d'un *sample* en pandas) puis de ramener, en second lieu, les classes minoritaires au même niveau. Pour le sur-échantillonnage, le choix s'est porté sur SMOTE : Synthetic Minority Over-sampling Technique. SMOTE (Chawla, Bowyer, Hall, & Kegelmeyer, 2002) est une technique assez populaire pour le sur-échantillonnage des données minoritaires. Elle génère des échantillons artificiels en multipliant la différence entre un vecteur et l'un de ses k plus proche voisins par un nombre aléatoire compris entre 0 et 1.

Il est à noter, que le sur-échantillonnage en question ne doit être appliqué que sur les données d'entrainement. Il faut alors que les données soient séparées au préalable.

4.4 Algorithmes de classification

A l'issue du traitement qui permet d'équilibrer le nombre d'échantillons de chaque classe, les données sont enfin prêtes pour entrainer les modèles. Même si l'algorithme forêts aléatoires présente des résultats satisfaisants, à l'issue de la sélection des variables, le choix a été fait de tester le modèle bayésien naïf et d'arbre de décision afin de pouvoir comparer différents modèles. Dans le cas du modèle bayésien naïf, les données ont été transformées pour qu'elles soient centrés réduites à l'aide du *StandardScaler()*.Cette transformation n'est pas nécessaire pour les algorithme basés sur CART. Le modèle bayésien naïf n'est pas très complexe et présente un temps d'entrainement de l'ordre de quelques secondes. L'arbre de décision(CART) a également un temps d'entrainement assez court et permet une prédiction quasi-instantanée. L'algorithme forêts aléatoires est le plus complexe, aussi bien en entrainement qu'on prédiction. Pour chacun des trois modèles testés, le tableau suivant fournit la complexité moyenne de la phase d'entrainement ainsi que la durée réellement observée lors des tests effectués avec scikit-learn.

	Complexité Entrainement	Durée (sec) entrainement	Complexité prédiction	Durée (sec) prédiction
Naïve Bayes	$O(pN)$	7	$O(p)$	3.5
CART	$O(pNlog^2N)$	132	$O(logN)$	0.37
RF	$O(MK\tilde{N}log^2\tilde{N})$	1245	$O(MlogN)$	13

Figure 10- Complexités des modèles[4]

p : Nombres de variables
M : Nombre d'arbres dans la forêt (*n_estimators* = 200)
K : La taille de l'échantillon des variables dans RF (*max_features* = $\sqrt{p}$)

[4] (Louppe, 2015) : Complexité pour CART et RF
https://www.thekerneltrip.com/machine/learning/computational-complexity-learning-algorithms/ pour Naïve Bayes

N : Nombre d'échantillons dans le jeu de données d'entrainement
Ñ = 0.632N : Taille du sous échantillon utilisé dans RF
RF a été exécuté avec l'option *n_jobs* = -1 afin paralléliser le calcul

Les durées d'entrainement et de prédiction sont très importantes pour le choix du modèle. Un système en production peut exiger un entrainement régulier du modèle et une prédiction avec des débits élevés des données. Il faut alors que les durées correspondantes soient les plus courtes que possible. Il reste alors à évaluer les performances de ces modèles afin de pouvoir choisir celui qui sera retenu comme modèle de classification.

4.5 Indicateurs de performance et résultats

Scikit-Lean implémente un module qui calcule différents indicateurs permettant d'évaluer les performances des modèles. Voici l'ensemble des indicateurs utilisés pour ce projet :

- Exactitude (*Accuracy*) : cet indicateur mesure la proportion des échantillons qui sont bien classifiés :

$$Acc = \frac{TP + TN}{TP + FP + TN + FN}$$

- Précision (*Precision*) : cet indicateur mesure l'aptitude du détecteur à ne détecter que les vraies attaques (donc minimiser les faux positifs)

$$Pr = \frac{TP}{TP + FP}$$

- Rappel (*Recall*) : cet indicateur mesure l'aptitude du détecteur à détecter toutes les attaques (donc minimiser les faux négatifs)

$$Rc = \frac{TP}{TP + FN}$$

- F-mesure (*F1*) : cet indicateur est une combinaison de la précision et du rappel.

$$F1 = \frac{2}{\frac{1}{Pr} + \frac{1}{Rc}}$$

Vu le déséquilibre dans la répartition des échantillons dans les classes, l'exactitude n'est pas du tout suffisante afin d'évaluer le modèle. Il faut alors se baser également sur la précision, le rappel et la F-mesure. Ces derniers indicateurs sont nativement conçus pour les classifications binaires avec une classe « positive » et une classe « négative ». Pour les classifications multi-classe, comme c'est le cas de ce projet, Scikit-Learn calcule ces indicateurs sur toutes les classes

puis effectue une moyenne selon différentes méthodes. C'est la moyenne « *weighted* » (qui tient compte du déséquilibre entre classes) qui a été retenu pour les tests.

	Bayésien Naïf	Arbre de decision	Forêts aléatoires
Exactitude	0.5697	0.9741	0.9715
Précision « *weighted* »	0.9165	0.9873	0.9881
Rappel « *weighted* »	0.5697	0.9741	0.9715
F-mesure « *weighted* »	0.6670	0.9789	0.9774

Tableau 3- Indicateurs Scikit-Learn

Le choix a été fait de de construire ces mêmes indicateurs différemment afin d'avoir une classe positive (les attaques) et une classe négative (flux IP normal). Cette méthode de calcul s'intéresse plutôt à l'aptitude à détecter l'intrusion. Elle ne pénalise alors pas une attaque mal classée du moment qu'elle ne soit pas prédite comme flux IP normal.

	Bayésien Naïf	Arbre de decision	Forêts aléatoires
Exactitude	0.5927	0.9745	0.9719
Précision	0.2822	0.8688	0.8524
Rappel	0.9986	0.9902	0.9970
F-mesure	0.4400	0.9255	0.9191

Tableau 4-Indicateurs personnalisés

Ces différents indicateurs fournissent une bonne idée des performances de chaque modèle. Pour le modèle bayésien naïf, les performances ne sont pas assez satisfaisantes notamment en ce qui concerne l'exactitude et la précision. Il parvient ceci dit à détecter les attaques même s'il ne les assigne pas à la bonne classe. Ce modèle fait l'hypothèse que les variables sont indépendantes, ce qui est faux vu que les variables présentent une corrélation importante. L'arbre de décision et les forêts aléatoires ont tous les deux d'excellentes performances et la différence entre les deux est vraiment marginale. Ceci dit l'arbre de décision est plus facile à paramétrer, plus rapide à entrainer et extrêmement rapide pour la prédiction. L'arbre de décision est alors le meilleur des trois algorithmes pour la classification du flux IP.

S'agissant d'une classification multi-classe, il est également intéressant de calculer la matrice de confusion qui détaille les résultats par classe. On peut ainsi avoir une idée plus précise des forces et des faiblesses du modèle. Comme le montre la figure 12, le modèle est très performant pour détecter la plupart des attaques. Cependant, il ne semble pas très adapté contre les attaques de type web qui, par nature, visent l'application et n'ont à priori pas de comportement typique en termes de trafic réseau. On peut également remarquer que le modèle a beaucoup de difficultés avec la classe DDoS. Cette classe est responsable de la quasi-totalité des faux positifs et des faux négatifs.

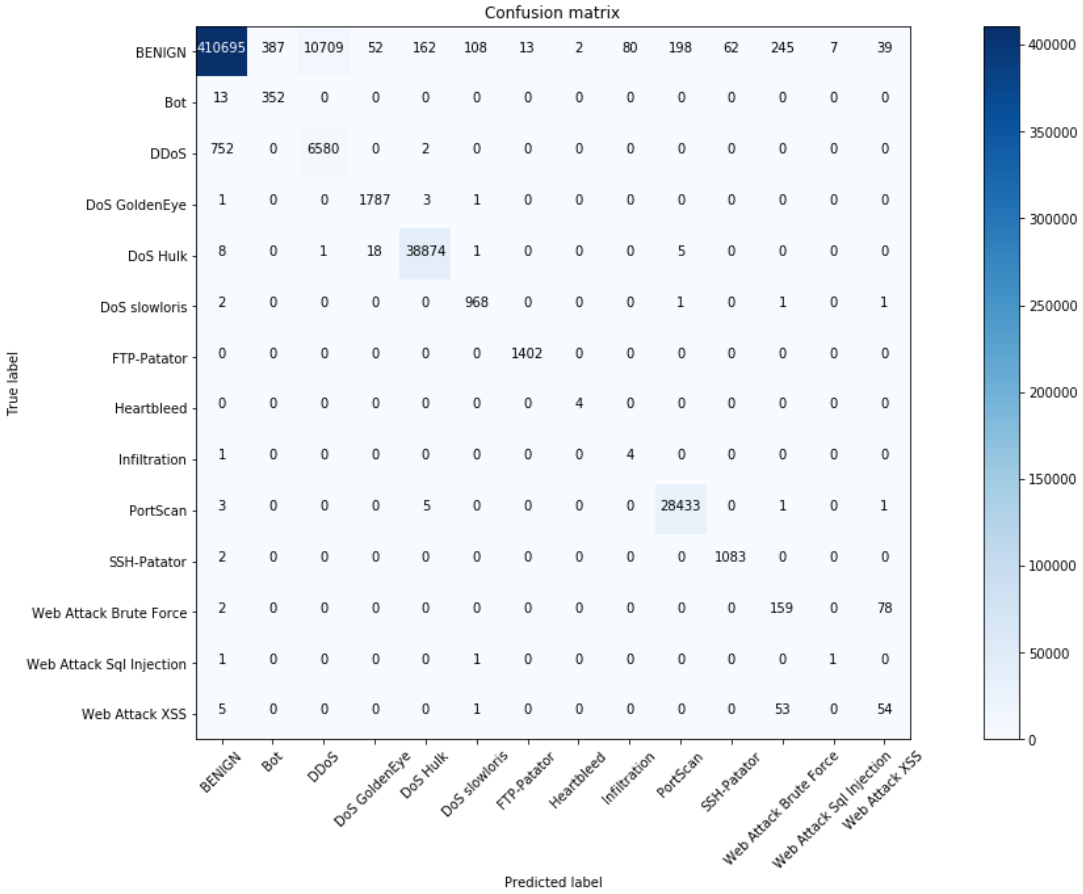

Figure 11-Matrice de confusion Arbre de décision

Même si les performances de l'algorithme sont satisfaisantes, il est intéressant de comprendre et d'expliquer les choix effectués par le modèle pour classifier le flux IP. Le paragraphe suivant explore l'interprétabilité des résultats.

4.6 Interprétation des résultats de classification

La difficulté d'interpréter les résultats des prédictions des algorithmes d'apprentissage automatique présente un défaut majeur et un frein à l'adoption de ces techniques. L'arbre de décision est cependant un des meilleurs algorithmes en termes d'interprétabilité. Il est, par exemple, possible de traduire le modèle obtenu en code source ou de visualiser l'arbre de décision en utilisant la librairie graphviz. Ceci dit, vu la dimension du jeu de données d'entrainement (2367806 échantillons et 41 variables), ces techniques ne permettent pas d'atteindre le résultat escompté. Tout comme les forêts aléatoires, l'arbre de décision permet également de classer l'importance des variables dans le modèle mais ceci ne permet pas d'expliquer le rôle de chaque variable dans une prédiction. Le choix s'est porté sur

treeinterpreter[5] qui permet de décomposer une prédiction donnée en somme de contributions des différentes variables. L'outil calcule la prédiction réalisée à l'aide d'un arbre de décision en utilisant l'approximation suivante[6] :

$$f(x) = c_{full} + \sum_{k=1}^{K} contrib(x,k)$$

c_{full} est le biais de prédiction apporté par la phase d'entrainement.
$contrib(x,k)$ est la contribution de chaque variable parmi les K variables.

Cette approche peut être étendue à l'algorithme forêts aléatoires :

$$F(x) = \frac{1}{J}\sum_{j=1}^{J} c_{j\ full} + \sum_{k=1}^{K}(\frac{1}{J}\sum_{j=1}^{J} contrib_j(x,k))$$

J est le nombre d'arbres de décisions dans forêts aléatoires.

treeinterpreter a été utilisé afin de comprendre la prédiction d'une attaque en tant que *DoS GoldenEye*. Voici les résultats :

- Le biais est de 0.07142 ce qui correspond à 1/14, 14 étant le nombre de classes.
- La contribution de la variable FlowBytesPs est de 0.57.
- La contribution de la variable FlowPackets/s est de 0.21.

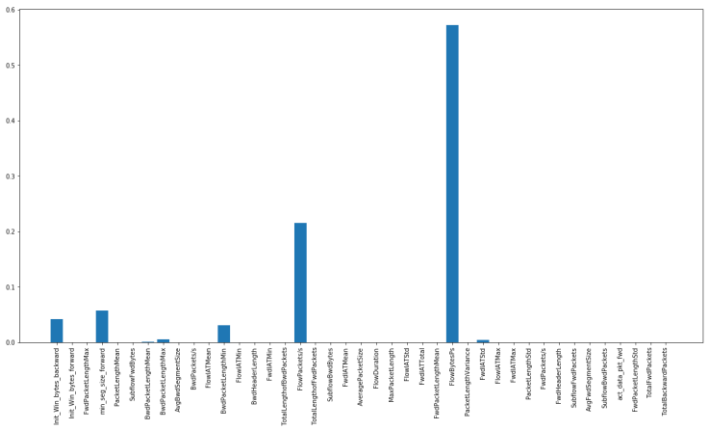

Figure 12 - Contribution des variables avec treeinterpreter

4.7 Détection d'anomalies par les modèles de classification

Le modèle a de très bonnes performances pour détecter et classifier les attaques connus. Ceci dit, de nouvelles attaques apparaissent tous les jours dans le réseau et il n'est pas facile de

[5] https://github.com/andosa/treeinterpreter
[6] http://blog.datadive.net/interpreting-random-forests/

maintenir une base de données de flux IP labélisées avec les dernières attaques découvertes. Il serait alors intéressant de savoir si le modèle arrive à détecter une attaque inconnue. C'est là qu'interviennent les données novelty.csv qui correspondent à une attaque inconnue par le modèle (attaque *DoSSlowhttptest*). L'algorithme RF classifie cette attaque comme flux IP normal pour 76% des échantillons. L'arbre de décision fait un peu mieux en faisant la prédiction d'un flux IP normal dans 53%. Les deux modèles échouent alors à détecter cette nouvelle attaque et il est fort probable que ces modèles ne soient pas capables de détecter d'autres nouvelles attaques. L'apprentissage non supervisé a alors été exploré afin de relever le défi des attaques inconnues.

5 Détection d'anomalies à l'aide de l'apprentissage non supervisé

Suite à l'inefficacité des modèles précédents pour détecter les nouvelles attaques, le choix s'est naturellement porté sur l'apprentissage non supervisé, à priori plus adapté à cette problématique. Parmi les algorithmes les plus connus, on peut citer le Séparateur à Vaste Marge à une classe (*One class SVM*) ou les forêts d'isolations (*Isolation Forests*). Ce dernier algorithme tolère des donnés d'entrainement potentiellement contaminés par les *outliers*. Ceci est intéressant dans la mesure où il est difficile d'avoir des données composées uniquement de flux IP normal. Malheureusement, la mémoire RAM du PC a été insuffisante pour entrainer le modèle forêts d'isolations et les tests effectués (avec un sous-échantillonnage assez fort) n'ont pas été concluants. Les efforts se sont concentrés alors sur la détection d'anomalies à l'aide de SVM à une classe.

5.1 Détection d'anomalies à l'aide SVM à une classe

SVM à une classe a été proposé par (Schölkopf, Platt, Shawe-Taylor, Smola, & Williamson, 2001). C'est un algorithme d'apprentissage non supervisé qui nécessite la connaissance d'une seule classe, le flux IP normal. Cet algorithme apprend alors les caractéristiques de cette classe afin de prédire si un échantillon appartient ou pas à la classe qu'il connait. Cet algorithme a besoin d'être paramétré selon le type de kernel sélectionné. Le kernel RBF a été retenu pour les tests. Ce kernel est généralement le choix par défaut pour SVM:

$$K_{RBF}(x, x') = e^{-\gamma|x-x'|^2}$$

γ est un paramètre du kernel RBF qui représente intuitivement la portée d'un échantillon dans la décision. Une valeur basse de γ implique une large portée et une valeur haute implique une influence sur les points proches.

SVM à une classe a un deuxième paramètre : **ν**, il correspond intuitivement à la fraction maximale d'erreurs (ou faux positifs) dans l'échantillon d'apprentissage. ν est compris entre 0 et 1.

5.2 Paramétrage du SVM à une classe

Le modèle SVM à une classe est défini par le couple de paramètres (ν, γ) comme vu précédemment. Il choix a été fait de choisir différentes valeurs de ν et de trouver le γ de telle sorte que le nombre de faux positifs obtenu soit exactement égal à ν. Cette phase d'entrainement est effectuée uniquement à l'aide d'échantillons appartement à la classe flux IP normal (*BENIGN*). Plusieurs modèles ont été construits en faisant varier ν. Choisir le bon modèle consiste à faire un compromis entre minimisation du taux des faux positifs et maximisation du taux des vrais positifs.

5.3 Résultats

Plusieurs modèles ont été entrainés avec des valeurs ν variant entre 0 et 0.2. Le taux de vrais positifs a été analysé pour l'ensemble des attaques et puis dans le détail pour 10 attaques. Les résultats montrent que la détection est très variable en fonction du type d'attaque.

Attaque \ (ν, γ)	(**0.01**,0.001)	(**0.02**,0.01)	(**0.04**,0.02)	(**0.06**,0.04)	(**0.1**,0.2)	(**0.2**,0.4)
Ensemble des attaques	2%	32%	35%	36%	58%	63%
DDoS	5%	25%	30%	32%	33%	41%
SSH patator	0%	0%	0%	0%	0.1%	22%
FTP patator	0%	0%	1%	2%	13%	23%
DoS GoldenEye	9%	33%	52%	68%	92%	96%
DoS slowloris	9%	54%	57%	57%	88%	97%
DoS Slowhttptest	23%	68%	90%	95%	98%	99%
Web Attack Brute Force	0%	5%	5%	5%	5%	5%
Bot	2%	2%	2%	3%	4%	9%
PortScan	0%	0%	1%	1%	51%	55%
DoSHulk	3%	59%	62%	63%	70%	75%

Tableau 5- Pourcentage des attaques détectées

L'ensemble des données calculées pour ces différents modèles permettent alors de tracer la courbe du taux des vrais positifs en fonction du taux des faux positifs. La courbe obtenue correspond à la courbe ROC (Receiving Operator Characteristic).

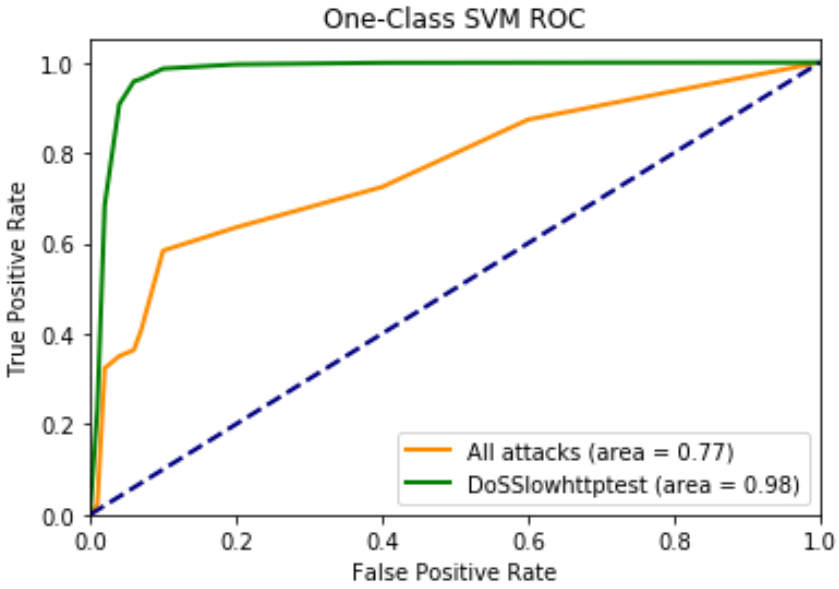

Figure 13-Courbe ROC

Selon les résultats du tableau 5, le modèle est capable de détecter 50% de l'ensemble des attaques à partir 10 % de faux positifs. Ce taux serait très élevé pour un système en production. Les résultats montrent également que le modèle est performant pour certaines attaques de type DoS. Pour *DoS Slowhttptest*, le système arrive à détecter 68% des attaques pour un taux de faux positif de 2%, 90% pour un taux de faux positifs de 4% et peut aller jusqu'à 98% pour un taux de faux positifs de 10%. Ce système, malgré ses résultats prometteurs, n'a pas la maturité nécessaire pour partir en production. Le modèle de classification arbre de décision a alors été retenu afin de créer une application de détection d'intrusions qui peut servir comme preuve de concept (POC).

6 Application IDS en temps réel

L'application développée est un système de détection d'intrusion (IDS) qui tourne sur un PC. Cette application est développée à l'aide de *flask* et se base sur le modèle arbre de décision persisté à l'aide de joblib.

6.1 Persistance du model avec joblib

Les modèles entrainés sont chargés en RAM dans le contexte d'un exécutable python. Ils sont alors déchargés quand on quitte le notebook. Il est alors indispensable de persister les

modèles construits afin de les réutiliser par la suite dans le cadre de prédictions par exemple. La documentation officielle de Scikit-learn préconise d'utiliser joblib.dump et joblib.load afin de, respectivement, persister et charger les modèles. Le tableau suivant liste les modèles persistés durant ce projet.

Modèle	Taille	Description
classification_DT_model.pkl	4.8 Mo	Modèle arbre de décision
ocsvm_model.pkl	10.5 Mo	Modèle One Class SVM
ocsvm_scaler_model.pkl	3 Ko	Fonction de standardization

Tableau 6- Modèles persistés

C'est l'arbre de décision : « classification_DT_model.pkl » qui a été retenu afin de construire l'application.

6.2 Application avec Flask

L'application IDS a été codée en python à l'aide de la librairie *flask*. Cette librairie est une sorte de *framework* qui permet de faire du développement web. L'application développée ouvre un fichier de log qui contient des échantillons de flux IP correspondants à différents classes de trafic réseau. L'application lit ce fichier ligne par ligne, transforme chaque ligne en un vecteur puis effectue la prédiction de la nature du flux en se basant sur le modèle déjà chargé.

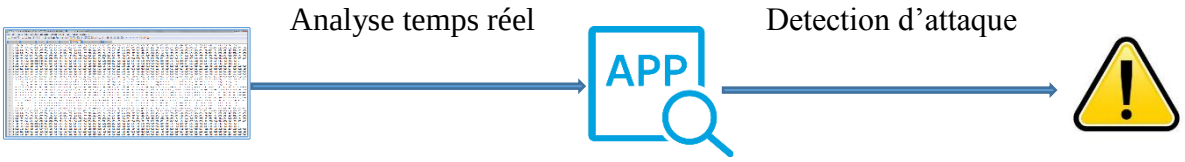

Figure 14- Détection d'intrusion en temps réel sur PC

L'application affiche une fenêtre d'alerte si le modèle prédit que le flux analysé correspond à une attaque.

7 Passage à l'échelle avec l'architecture Lambda

L'application conçue à l'aide de *flask* est une preuve de concept qui ne tient pas réellement compte de certains aspects tels que la scalabilité et la tolérance aux erreurs. En pratique, l'application doit être capable d'analyser le réseau d'une organisation avec une quantité de trafic réseau importante. Un tel trafic peut également être amené à évoluer dans le temps et le système doit être capable de s'adapter. Il convient alors de faire appel à un ensemble de technologies big data afin de permettre le passage à l'échelle du système.

Une fois en production, le système de détection d'intrusion comprend trois fonctionnalités principales. La première consiste à construire le modèle. La deuxième correspond à la prédiction. La troisième consiste à exposer les résultats. La construction du modèle doit être effectuée assez régulièrement afin de tenir compte de l'évolution du trafic et des attaques. Ce traitement doit être effectué hors ligne, c'est-à-dire en mode batch. Spark, grâce notamment à Spark SQL et MLlib, permet de remplir parfaitement cet objectif. Spark peut s'appuyer sur un stockage HDFS pour le traitement batch. La prédiction, contrairement à la construction du modèle, doit être assurée en continue. Spark Streaming s'avère alors être un bon choix. Ce choix permet en plus d'utiliser une seule technologie (Spark) pour la construction du modèle et la prédiction. On garde ainsi une certaine cohérence dans le système. Spark streaming peut être alimenté par Kafka. En ce qui concerne l'exposition des résultats, il est possible de faire appel à Cassandra afin de stocker les données. Cette architecture en 3 couches correspond à une **architecture Lambda** avec une couche batch (*batch layer*) pour les traitements réguliers, une couche temps-réel (*spead layer*) pour traiter un flux continu de données et une couche de service (*service layer*) pour la présentation des résultats.

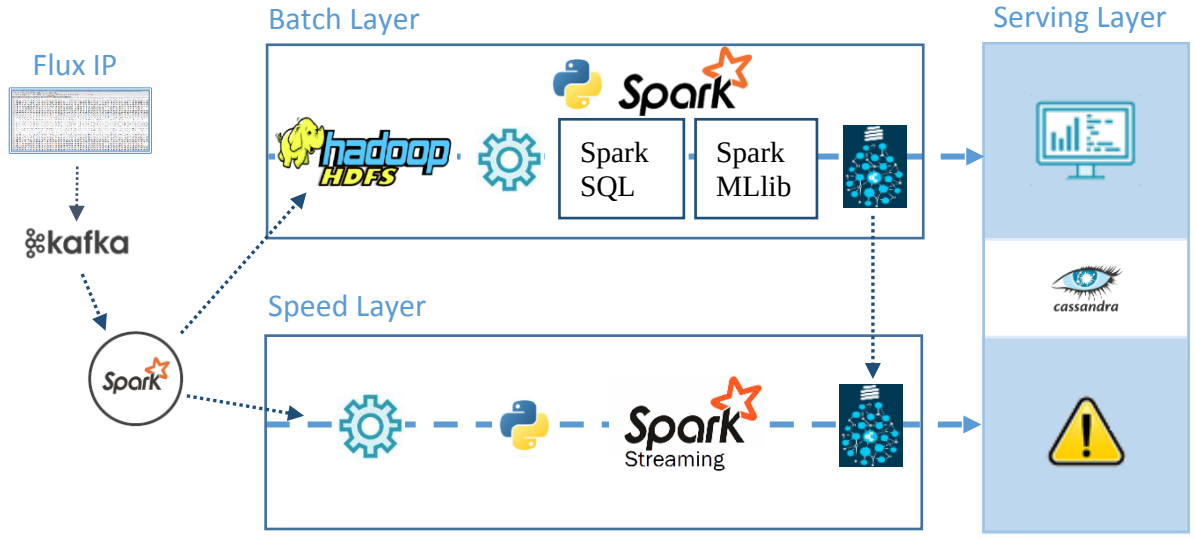

Figure 15- Système de détection d'intrusion – Architecture Lambda

7.1 Couche batch

L'objectif essentiel de cette couche est de traiter les données brutes et d'effectuer les transformations nécessaires afin d'entrainer par la suite un modèle prédictif. Le nettoyage des données est identique à ce qui a été fait avec *pandas*. Cependant, MLlib présente quelques

spécificités notamment en ce qui concerne le format des données qui servent à l'entrainement du modèle. MLlib exige, en fait, la présence d'un vecteur « features » qui contient toutes les variables indépendantes et d'une colonne « label » qui contient les valeurs dépendantes. Le label représente une chaine de caractère qui correspond au type du flux IP et il est nécessaire de l'indexer en valeur numérique.

Une fois le modèle entrainé, il est nécessaire de définir un pipeline qui contient toutes les transformations effectuées en plus du modèle entrainé. Ce pipeline doit être par la suite sauvegardé afin d'être réutilisé pour la prédiction.

L'implémentation effectuée dans le cadre du projet s'est concentrée sur les transformations spécifiques à MLlib (vecteur « features » et indexation de « label ») ainsi que l'entrainement et l'évaluation du modèle. Les performances du modèle MLlib sont légèrement moins bonnes que celles du modèle Scikit-learn vu que le déséquilibre de classes n'a pas été réalisé au niveau du code Spark. Les résultats sont tout de mêmes très bons :

Exactitude : 0.959411
Précision : 0.954285
Rappel : 0.959411
F-mesure : 0.952071

7.2 Couche temps-réel

L'objectif principal de cette couche et de pouvoir traiter le flux de données en continue afin d'effectuer les prédictions. Spark streaming ne fait pas, à proprement dit, de streaming temps réel mais traite les données en micro batch. Ceci dit, il est parfaitement adapté à ce projet, vu que qu'une latence de l'ordre de la seconde est tout à fait acceptable dans le cadre de la détection d'intrusion.

Spark streaming propose un connecteur Kafka afin de consommer les données en continue. Chaque microbatch est traité en appliquant le pipeline qui contient toutes transformations et le modèle.

7.3 Couche service

Cette couche a deux missions principales. La première consiste à remonter les alertes en cas de prédiction d'une attaque. La deuxième consiste à remonter différentes métriques et indicateurs suite à l'analyse des données stockées au niveau de la base Cassandra.

8 Conclusion

Ce projet a permis de concevoir et de développer une preuve de concept d'un système de détection d'intrusion basé sur le flux IP en utilisant l'apprentissage supervisé. Le système, conçu à l'aide de l'algorithme arbre de décision, a des résultats très satisfaisants même s'il n'est pas adapté contre certaines attaques de type web. L'architecture Lambda proposée permettrait à ce système d'être opérationnel dans des conditions industrielles. La suite naturelle de ce projet consiste alors à continuer l'implémentation relative à cette architecture et de faire le déploiement sur un environnement distribué afin de tirer parti des technologies big data. Le défi majeur de cette approche est la construction de données labélisées dans un réseau où les nouvelles attaques apparaissent régulièrement.

Ce projet a également permis d'explorer l'apprentissage non supervisé. L'algorithme SVM à une classe s'avère particulièrement efficace contre certaines attaques. Mais d'autres pistes doivent être explorées afin d'atteindre les performances d'un système industrialisable. Il faudrait notamment essayer d'autres algorithmes avec des ressources matérielles plus adaptées tels que des serveurs de calculs dans le cloud. Il serait également intéressant d'explorer les approches basées sur les séries temporelles qui seraient parfaitement adaptées pour la détection d'intrusion.

Afin de s'imposer plus largement, les IDS basés sur le flux doivent faire preuve de fiabilité sur la durée et gagner ainsi la confiance des experts. Ces systèmes souffrent pour le moment des faiblesses propres à tout système basé sur l'apprentissage automatique. D'une part, certains modèles prédictifs sont assez difficiles à interpréter. D'autre part, ces systèmes peuvent être les cibles d'attaques contradictoires (*adversarial machine learning attacks*) ce qui représente une nouvelle surface d'attaque pour les cybercriminels. D'énormes défis doivent être relevés par la recherche dans le domaine de l'apprentissage automatique afin de favoriser sa large adoption dans le monde de la cybersécurité. Malgré les limites actuelles de cette approche, elle sera incontournable, d'ici quelque années, afin de de sécuriser le réseau dans un monde dans lequel tout, ou presque, devient connecté !

Bibliographie

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence*, 321–357.

Freeman, D., & Chio, C. (2018). *Machine Learning and Security.*

Gerard Draper-Gil, A. H., Mamun, M. S., & Ghorbani, A. A. (2016). Characterization of Encrypted and VPN Traffic Using Time-Related Features. *In the proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP 2016)*, (pp. 407-414). Italy.

http://blog.datadive.net/interpreting-random-forests/

http://www.unb.ca/cic/datasets/ids-2017.html

https://github.com/andosa/treeinterpreter

https://transparencyreport.google.com/https/overview

https://www.internetworldstats.com/stats.htm

https://www.thekerneltrip.com/machine/learning/computational-complexity-learning-algorithms/

https://www.pwc.fr/fr/espace-presse/communiques-de-presse/2018/mars/76-pourcent-des-eti-ont-deja-subi-une-attaque-cyber.html

Lashkari, A. H., Draper-Gil, G., & Ghorbani, M. S. (2017). Characterization of Tor Traffic Using Time Based Features. *3rd International Conference on Information System Security and Privacy, SCITEPRESS.* Porto,Portugal.

Louppe, G. (2015). UNDERSTANDING RANDOM FORESTS. p. 96;98.

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J., & Williamson, R. C. (2001). Estimating the Support of a High-Dimensional Distribution. *Neural Computation 13, 1443–1471*.

Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. *In Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP).* Portugal.

Glossaire

CART : Classification And Regression Trees

CIA : Confidentiality Integrity Availability

CIC : Canadian Institute for Cybersecurity

CPU : Central Processing Unit

CSV : Comma Separated Value

DDoS : Distributed Denial of Services

DoS : Denial of Service

DT : Decision Tree

FN: False Negatives

FP : False Positives

HTTPS : HyperText Transfer Protocol Secure

IDS : Intrusion Detection System

IP : Internet Protocol

MVP : Minimum Viable Product

NaN : Not a Number

NAT : Network Address Translation

NB : Naive Bayes

OCSVM : One-Class SVM

PC : Personal Computer

POC : Proof Of Concept

RAM : Random Access Memory

RBF : Radius Basis Function

RF : Random Forest

RFC : Request For Comment

ROC : Receiving Operator Characteristic

SMOTE : Synthetic Minority Over-sampling TEchnique

SVM : Support Vector Machine – Séparateur à vastes marges

TCP : Transmission Control Protocol

TLS : Transport Layer Security

TN : True Negatives

TOR : The Onion Router

TP : True Positives

VPN : Virtual Private Network

Annexe 1 : Liste des variables

Feature name	Description
Duration	Duration of the flow in Microsecond
total_fpackets	Total packets in the forward direction
total_bpackets	Total packets in the backward direction
total_fpktl	Total size of packet in forward direction
total_bpktl	Total size of packet in backward direction
min_fpktl	Minimum size of packet in forward direction
min_bpktl	Minimum size of packet in backward direction
max_fpktl	Maximum size of packet in forward direction
max_bpktl	Maximum size of packet in backward direction
mean_fpktl	Mean size of packet in forward direction
mean_bpktl	Mean size of packet in backward direction
std_fpktl	Standard deviation size of packet in forward direction
std_bpktl	Standard deviation size of packet in backward direction
total_fiat	Total time between two packets sent in the forward direction
total_biat	Total time between two packets sent in the backward direction
min_fiat	Minimum time between two packets sent in the forward direction
min_biat	Minimum time between two packets sent in the backward direction
max_fiat	Maximum time between two packets sent in the forward direction
max_biat	Maximum time between two packets sent in the backward direction
mean_fiat	Mean time between two packets sent in the forward direction
mean_biat	Mean time between two packets sent in the backward direction
std_fiat	Standard deviation time between two packets sent in the forward direction
std_biat	Standard deviation time between two packets sent in the backward direction

fpsh_cnt	Number of times the PSH flag was set in packets travelling in the forward direction (0 for UDP)
bpsh_cnt	Number of times the PSH flag was set in packets travelling in the backward direction (0 for UDP)
furg_cnt	Number of times the URG flag was set in packets travelling in the forward direction (0 for UDP)
burg_cnt	Number of times the URG flag was set in packets travelling in the backward direction (0 for UDP)
total_fhlen	Total bytes used for headers in the forward direction
total_bhlen	Total bytes used for headers in the forward direction
fPktsPerSecond	Number of forward packets per second
bPktsPerSecond	Number of backward packets per second
flowPktsPerSecond	Number of flow packets per second
flowBytesPerSecond	Number of flow bytes per second
min_flowpktl	Minimum length of a flow
max_flowpktl	Maximum length of a flow
mean_flowpktl	Mean length of a flow
std_flowpktl	Standard deviation length of a flow
min_flowiat	Minimum inter-arrival time of packet
max_flowiat	Maximum inter-arrival time of packet
mean_flowiat	Mean inter-arrival time of packet
std_flowiat	Standard deviation inter-arrival time of packet
flow_fin	Number of packets with FIN
flow_syn	Number of packets with SYN
flow_rst	Number of packets with RST
flow_psh	Number of packets with PUSH
flow_ack	Number of packets with ACK
flow_urg	Number of packets with URG

flow_cwr	Number of packets with CWE
flow_ece	Number of packets with ECE
downUpRatio	Download and upload ratio
avgPacketSize	Average size of packet
fAvgSegmentSize	Average size observed in the forward direction
fAvgBytesPerBulk	Average number of bytes bulk rate in the forward direction
fAvgPacketsPerBulk	Average number of packets bulk rate in the forward direction
fAvgBulkRate	Average number of bulk rate in the forward direction
bAvgSegmentSize	Average size observed in the backward direction
bAvgBytesPerBulk	Average number of bytes bulk rate in the backward direction
bAvgPacketsPerBulk	Average number of packets bulk rate in the backward direction
bAvgBulkRate	Average number of bulk rate in the backward direction
sflow_fpacket	The average number of packets in a sub flow in the forward direction
sflow_fbytes	The average number of bytes in a sub flow in the forward direction
sflow_bpacket	The average number of packets in a sub flow in the backward direction
sflow_bbytes	The average number of bytes in a sub flow in the backward direction
min_active	Minimum time a flow was active before becoming idle
mean_active	Mean time a flow was active before becoming idle
max_active	Maximum time a flow was active before becoming idle
std_active	Standard deviation time a flow was active before becoming idle
min_idle	Minimum time a flow was idle before becoming active
mean_idle	Mean time a flow was idle before becoming active
max_idle	Maximum time a flow was idle before becoming active
std_idle	Standard deviation time a flow was idle before becoming active
Init_Win_bytes_forward	The total number of bytes sent in initial window in the forward direction

Init_Win_bytes_backward	The total number of bytes sent in initial window in the backward direction
Act_data_pkt_forward	Count of packets with at least 1 byte of TCP data payload in the forward direction
min_seg_size_forward	Minimum segment size observed in the forward direction

Annexe 2 : Les livrables des sprints

Code source

Sprint	Livrables	Description
Sprint 1 : Exploration et traitement des données	01_explore-process-data.ipynb	Nettoyage et analyse des données
Sprint 3 : Sélection des variables et classification	sp3-01_features-selection.ipynb	Première classification à l'aide de RF et sélection des variables
	sp3-02_classification-model.ipynb	Classification à l'aide de NB, DT et RF
	sp3-03_ids_app.py	Application basée sur le modèle DT
Sprint 4 : Détection d'anomalies	sp4_anomaly-detection-model.ipynb	Détection d'anomalies en utilisant OCSVM
Sprint 5 : Passage à l'échelle	sp5-Spark-ml-classification.ipynb	Modèle de classification à l'aide de Spark

Modèles

Sprint	Livrables	Description
Sprint 3 : Sélection des variables et classification	classification_DT_model.pkl	Modèle DT utilisé par l'application sp3-03_ids_app.py
Sprint 4 : Détection d'anomalies	ocsvm_model.pkl ocsvm_scaler_model.pkl	Modèle OCSVM de détection d'anomalies et le modèle de standardisation

Annexe 3 : Analyse des attaques

Attaques	IP Externe	IP Source	IP Destination	Port Source	Port Dest	Nombre
DoS Hulk	NaN	172.16.0.1	192.168.10.50	Multiple	80	215590
PortScan	NaN	172.16.0.1	192.168.10.50	Multiple	Multiple	158787
DDoS	205.174.165.71 24814 205.174.165.69 8935 205.174.165.70 7023 205.174.165.205. 174.165.70 1	172.16.0.1 40800 192.168.10.50 3	192.168.10.50 40800 172.16.0.1 3	Multiple	80	40803
DoS GoldenEye	NaN	172.16.0.1	192.168.10.50	Multiple	80	9623
FTP-Patator	NaN	172.16.0.1	192.168.10.50	Multiple	21 7910 80 1	7911
SSH-Patator	NaN	172.16.0.1	192.168.10.50	Multiple	22	5897
DoS slowloris	NaN	172.16.0.1	192.168.10.50	Multiple	80	5410
DoS Slowhttptest	NaN	172.16.0.1	192.168.10.50	Multiple	80	5090
Bot	NaN	205.174.165.73 701 192.168.10.15 360 192.168.10.8 268 192.168.10.9 226 192.168.10.14 209 192.168.10.5 178 192.168.10.12 2 192.168.10.17 2	205.174.165.73 1241 192.168.10.15 208 192.168.10.9 146 192.168.10.14 139 192.168.10.5 108 192.168.10.8 100 52.7.235.158 2 52.6.13.28 2	8080 – Multiple	8080 – Multiple	1946
Web Attack Brute Force	NaN	172.16.0.1	192.168.10.50	Multiple	80	1410
Web Attack XSS	NaN	172.16.0.1	192.168.10.50	Multiple	80	614
Infiltration	NaN	192.168.10.8	205.174.165.73	Multiple	444	36
Web Attack Sql Injection	NaN	172.16.0.1	192.168.10.50	Multiple	80	15
Heartbleed	NaN	172.16.0.1	192.168.10.51	45022	444	11

Annexe 4: Importance des variables

1) Init_Win_bytes_backward 0.078754
2) Init_Win_bytes_forward 0.035087
3) FwdPacketLengthMax 0.034939
4) min_seg_size_forward 0.031537
5) PacketLengthMean 0.029018
6) SubflowFwdBytes 0.025572
7) BwdPacketLengthMean 0.024790
8) BwdPacketLengthMax 0.024562
9) AvgBwdSegmentSize 0.024307
10) BwdPackets/s 0.023839
11) FlowIATMean 0.023653
12) BwdPacketLengthMin 0.023111
13) FlowIATMin 0.022906
14) BwdHeaderLength 0.022014
15) FwdIATMin 0.021740
16) TotalLengthofBwdPackets 0.021548
17) FlowPackets/s 0.021383
18) TotalLengthofFwdPackets 0.020728
19) SubflowBwdBytes 0.019838
20) FwdIATMean 0.019306
21) AveragePacketSize 0.018960
22) FlowDuration 0.018541
23) MaxPacketLength 0.018135
24) FlowIATStd 0.017782
25) FwdIATTotal 0.017776
26) FwdPacketLengthMean 0.017283
27) FlowBytesPs 0.017121
28) PacketLengthVariance 0.016894
29) FwdIATStd 0.016774
30) FlowIATMax 0.016687
31) FwdIATMax 0.016588
32) PacketLengthStd 0.016365
33) FwdPackets/s 0.016233
34) FwdHeaderLength 0.015763
35) SubflowFwdPackets 0.015477
36) AvgFwdSegmentSize 0.014766
37) SubflowBwdPackets 0.014744
38) act_data_pkt_fwd 0.012818
39) FwdPacketLengthStd 0.012324
40) TotalFwdPackets 0.012314
41) TotalBackwardPackets 0.011787
42) BwdIATMin 0.009737
43) BwdPacketLengthStd 0.009441
44) URGFlagCount 0.008752
45) MinPacketLength 0.008558
46) BwdIATMean 0.008311
47) PSHFlagCount 0.007801
48) BwdIATTotal 0.007629

49) IdleMean 0.007477
50) FwdPacketLengthMin 0.007063
51) BwdIATMax 0.006621
52) ACKFlagCount 0.005397
53) IdleMin 0.004964
54) IdleMax 0.004802
55) BwdIATStd 0.004211
56) FwdPSHFlags 0.003340
57) Down/UpRatio 0.003333
58) SYNFlagCount 0.002340
59) ActiveMin 0.002178
60) ActiveMax 0.001545
61) ActiveMean 0.001529
62) FINFlagCount 0.000637
63) IdleStd 0.000354
64) ActiveStd 0.000214
65) CWEFlagCount 0.000001
66) FwdURGFlags 0.000000
67) RSTFlagCount 0.000000
68) ECEFlagCount 0.000000