Algorithmes d'Intelligence Artificielle en Cybersécurité & Intégration en environnements contraints

Stéphane Morucci[1], Stéphane Davy[2], Nicolas Raux[2], Jérémy Scion[3], Guillaume Lerouge[4], Marc Le Nué[1], Nathan Rydin[1], Dorian Screm[1]

[1] Orange Labs, 4 rue du Clos Courtel, 35510 Cesson-Sévigné
[2] Orange Cyber Défense, 9 rue du Chêne Germain, 35510 Cesson-Sévigné
[3] Orange Cyber Défense, 54 place de l'Ellipse, 92000 Nanterre
[4] SoftAtHome, 9 rue du Débarcadère, 92700 Colombes

Résumé. L'Intelligence Artificielle (IA) dans le domaine de la cybersécurité offre un formidable champ d'exploration et promet pour la première fois la possibilité de détection d'attaques jusqu'alors inconnues. Les promesses de l'IA pour la cyber couvrent également une limitation des ressources opérationnelles nécessaires grâce à un filtrage en amont du trafic (moins de flux au niveau des équipements de surveillance) et des gains de temps pour les exploitants (diminution des faux-positifs, meilleure pertinence des alertes). Par ailleurs, le recours à des solutions non-supervisées permet aussi d'envisager une mise en exploitation très rapide comparée aujourd'hui à plusieurs semaines pour la mise en place d'un SIEM (System Information and Event Management, la pierre angulaire d'une solution de supervision de la sécurité).

Cet article propose de comparer les performances de quelques algorithmes de détection d'anomalies sur deux jeux de données (« datasets ») et d'étudier un déploiement dans un contexte opérationnel.

Travailler sur des datasets de production nécessite la mise en place de techniques de protection de données dont certaines seront exposées avec leurs impacts associés.

L'intégration d'algorithmes d'IA dans un environnement de production impose des contraintes à différents niveaux. Nous présentons dans cet article, trois types d'intégrations : l'intégration dans deux SIEMs du marché, l'intégration planifiée dans une solution Open-Source de type Big-Data et notre vision d'une intégration dans un écosystème d'opérateur Télécom, en particulier au niveau des box ADSL/Fibre Grand-Public et PME.

Mots-clés: Cybersécurité, Détection d'anomalie, SIEM, Intégration, environnements contraints.

1 Introduction

Nous vivons dans un monde où les objets connectés, comme les smartphones et les ordinateurs, sont omniprésents. Pour communiquer entre eux, ces objets utilisent des réseaux, principalement Internet. Aujourd'hui, la sécurité de ces réseaux est compromise par la multiplication de cyber attaquants qui exploitent les failles des équipe-

ments et des logiciels pour, par exemple, exfiltrer des données confidentielles, paralyser les systèmes informatiques, chiffrer des données à des fins d'extorsion, etc.

Sécuriser ces réseaux constitue une problématique majeure. Des offres de services se sont développées autour de centres d'opérations sécurité des réseaux, les SOCs (Security Operations Center), qui ont la charge de surveiller les réseaux pour en assurer leur sécurité. Pour cela, ils enregistrent et appliquent des algorithmes sur le trafic et alertent si son comportement devient anormal. Cette détection d'anormalité s'appuie sur des règles statiques correspondant à des comportements suspicieux préalablement connus et généralement contractualisés avec le client final.

La fluctuation du trafic réseau, l'automatisation des attaques (ou au moins des prises d'empreinte) et la difficulté de maintenir en continu les règles statiques d'un SIEM challengent le travail d'un SOC.

L'émergence des techniques d'IA, avec leur capacité à détecter automatiquement des comportements anormaux sans connaissance préalable, se présente aujourd'hui comme une formidable opportunité pour compléter efficacement les capacités de défense des SOCs.

Dans le cadre de cette démarche d'intégration d'algorithmes d'IA aux ressources IT des équipes opérationnelles, la donnée devient clé. Cette approche « Data-Driven » entraîne avec elle son lot de contraintes, notamment en lien avec le RGPD (Règlement Général pour la Protection des Données). Nous présentons dans cet article quelques techniques qui ont été appliquées pour permettre de « libérer » la donnée et son utilisation afin de la rendre accessible, directement ou indirectement, aux différentes parties prenantes.

L'intégration de l'IA à proprement parler, impose certaines contraintes liées aux contextes d'exécution de ces algorithmes. Dans cet article, nous présentons trois types d'intégrations possibles dans des environnements différemment contraints: contraintes mémoires et CPU, contraintes en termes de performances attendues et contraintes liées à des systèmes embarqués.

Cet article est organisé de la façon suivante : le paragraphe II détaille les deux jeux de données utilisés. La troisième partie présente quelques techniques utilisées pour protéger les données manipulées lors des phases d'apprentissage ainsi que les caractéristiques retenues et calculées servant d'entrée aux algorithmes d'IA. Nous discuterons également d'une solution pour compenser le manque de représentativité de données dans certaines classes d'attaques. Les différents algorithmes utilisés et les démarches d'optimisation sont présentés dans le paragraphe 4. Viennent ensuite les différentes intégrations réalisées : (i) au sein d'architectures contraintes comme les outils de production du SOC Orange et (ii) au sein d'une solution Open-Source dédiée à la cyber sécurité (Apache Metron). L'architecture cible pour un déploiement sur des environnements particulièrement contraints comme des box d'accès Internet termine le paragraphe 5. Le paragraphe 6 présente les prochains travaux envisagés.

2 Jeux de données utilisés (Datasets)

2.1 Dataset CIC IDS 2017

Le Canadian Institute of Cybersecurity (CIC) de l'Université du New Brunswick a publié plusieurs jeux de données particulièrement intéressants. Nous nous intéressons ici au dataset CIC-IDS 2017 [1]. Ce dataset reproduit le trafic réseau d'une semaine d'un LAN (pour Local Area Network) et contient plusieurs attaques. Fait important, les logs d'attaque sont clairement identifiés : ces données sont donc labélisées ce qui est un point clé pour pouvoir entrainer efficacement et comparer entre eux différents algorithmes supervisés et non supervisés, et valider les résultats obtenus. Le jeu de données propose environ 80 caractéristiques (features) qui peuvent être utilisées pour la détection d'attaques.

A noter également qu'un autre dataset a récemment été mis à disposition par ce même organisme [2].

2.2 Logs internes Cybertest

Ces logs sont issus de plateformes internes qui ont capturé le trafic généré lors du déploiement et de l'utilisation d'un réseau complet d'entreprise pendant une période donnée suivie d'une période d'attaque. Une partie seulement des attaques est labélisée par des experts sécurité Orange. Ce dataset est divisé en deux parties : une partie pour l'apprentissage et une partie pour la validation d'algorithmes, sur lesquels nous avons appliqué des algorithmes de Machine Learning supervisés et non supervisés. Ce dataset comprend essentiellement des logs proxy et firewall.

3 Confidentialité et standardisation des données

3.1 Confidentialité des données

Les données de production mises à disposition des équipes en charge de la réalisation d'algorithmes d'IA sont soit des données issues du trafic interne, soit des données issues du trafic de clients d'Orange. Dans les deux cas, il est essentiel de sécuriser ces données afin d'assurer la confidentialité des informations contenues dans ces données (vie privée, ressources critiques d'une entreprise, etc.).

Les techniques généralement utilisées consistent à sécuriser en amont ces données en remplaçant certaines informations sensibles (adresse IP source, adresse MAC source, identifiant de connexion …) par des hashs voire des tokens aléatoires (tokenisation). Il est important, pour les équipes qui fournissent ces données, de conserver la correspondance entre les données initiales et les données transformées dans une structure de type dictionnaire (avec accès contrôlé). Les futures anomalies détectées pourront ainsi être exploitées efficacement en inversant le processus de tokenisation à l'aide de ce type de dictionnaire. Cette méthode a l'avantage d'être simple et rapide à mettre en place, même si certaines informations peuvent être perdues. Par exemple,

remplacer des adresses IP d'un même sous-réseau par un hash ou un token aléatoire, fait perdre l'information que toutes ces adresses IP appartiennent à un même sous-réseau. Dans le cadre d'une analyse de mouvements latéraux (propagation de virus notamment), la perte de cette information peut être pénalisante.

Une autre technique employée avec succès entre les équipes de recherche et opérationnelles consiste à déplacer les algorithmes plutôt que les données. Le processus mis en place s'inspire des méthodes de type DevOps [6] : (i) les équipes opérationnelles envoient un échantillon représentatif des données à traitées aux équipes d'IA. (ii) Ces dernières mettent en place les connecteurs et les traitements nécessaires pour exploiter les données. (iii) Les algorithmes compilés (éventuellement offusqués) sont envoyés aux équipes opérationnelles pour être exécutés par les exploitants sur les plateformes hébergeant les données de production. (iv) Une boucle de retour permet aux équipes opérationnelles de confirmer ou infirmer des anomalies remontées par les algorithmes d'IA (après analyse). Un nouveau cycle peut alors démarrer.

A noter que cette boucle de retour (point (iv)) devrait être mise en place systématiquement dans une démarche collaborative d'amélioration continue des performances des algorithmes d'IA. L'analyse humaine est en effet très précieuse pour les calibrer régulièrement.

Une méthode complémentaire consiste à contrôler l'accès et à chiffrer les données stockées. Nous avons utilisé avec succès les fonctionnalités de chiffrement de données de l'écosystème Hadoop basées sur l'authentification Kerberos.

Dans le monde du grand-public, notamment pour les usages liés au trafic Internet, le problème de la confidentialité est moindre. Bien sûr, les informations collectées par les Livebox sont protégées afin de rendre impossible toute identification du trafic (à l'aide de techniques de diffential privacy [21] notamment). En revanche, comme le trafic est globalement similaire pour une grande partie des utilisateurs grand-public, il est possible de définir assez finement des profils de trafic. De ce fait, étudier une sous-partie des clients devient suffisant pour développer des algorithmes de détection d'anomalies. Ces algorithmes pourront également être déployés chez nos clients, et ce, sans aucun impact sur la confidentialité de leurs données.

3.2 Traitement des données

Au-delà du filtrage nécessaire des données d'entrée (nos « datasets »), voire du rééquilibrage de classes pour l'apprentissage supervisé, nous avons décidé de « standardiser » certaines caractéristiques afin que nos travaux puissent être portés plus facilement en environnement de production où les logs à disposition ne présentent pas toujours toutes les caractéristiques que nous pouvons trouver dans le dataset CIC IDS 2017 par exemple.

Dans cette étude, nous nous concentrons sur les champs suivants : le timestamp, le nombre d'octets envoyés (sent bytes), le nombre d'octets reçus (received bytes), l'adresse IP de l'émetteur (source IP), l'adresse IP du destinataire (dest. IP), le port TCP/UDP source (source port) et le port TCP/UDP de destination (dest. port).

L'intérêt principal est que ces champs se retrouvent quasiment systématiquement dans les logs des équipements réseaux gérés par Orange en interne ou par délégation de ses clients. Nous avons donc fait un choix restrictif sur le nombre de caractéristiques considérées, dans un souci de portabilité de nos algorithmes, quitte à perdre en précision. Les résultats obtenus sur le dataset CIC IDS 2017 ont été appliqués sur notre dataset interne Cybertest (avec le passage d'un trafic de type flux pour le dataset CIC IDS à un trafic de type proxy ou firewall) et ce, après réduction de features. Par extension, notre idée est de nous appuyer le plus possible sur une base de logs labélisés et d'appliquer les modèles ainsi obtenus sur une base de logs de production, idéalement lorsque les distributions statistiques des features retenues restent similaires. Par ailleurs, nous privilégions une approche itérative avec les analystes sécurité (i.e. les experts métiers) et, dans ce cadre, un nombre limité de features permet de simplifier les interprétations des résultats des modèles et d'augmenter le stock de labels après validation. Cette approche a également l'avantage de pouvoir capitaliser sur les résultats obtenus auprès de plusieurs clients finaux.

Nous rajoutons à nos données brutes réduites, certains champs statistiques qui doivent contribuer à l'explication de certaines attaques. Le traitement au final est le suivant :

1. Sur l'ensemble du dataset, identifier les 10 ports de destination les plus utilisés et les appeler D1, D2, …, D10 ; le choix de cette valeur de 10 est arbitraire même s'il facilite les échanges avec les opérationnels. Pour les autres ports, appeler D11 un port compris entre 0 et 1023, D12 un port compris entre 1024 et 49151 et D13 un port compris entre 49152 et 65535 (les ports allant de 0 à 1023 sont des ports généralement utilisés par des services standards. Les ports allant de 1024 à 49151 sont des ports enregistrés, alloués à la demande pour divers protocoles et logiciels. Les ports allant de 49152 à 65535 sont les ports dynamiques et sont utilisables librement. C'est une information métier importante qu'il est pertinent d'ajouter aux vecteurs).

2. Par période d'agrégation (qui varie de la minute à l'heure), collecter tous les logs associés à une paire Source IP et Destination IP. La période d'agrégation utilisée est un hyperparamètre à optimiser comme un autre, même si elle peut être contrainte par certaines demandes opérationnelles (détecter du beaconing impose par exemple une période d'agrégation intimement liée à la fréquence de communication avec les serveurs de Command & Control). Les différentes expérimentations menées, en lien avec les attaques devant être détectées, nous ont conduits à choisir, à ce jour, le quart d'heure (versus 1 minute, 5 minutes, 1 heure et 1 jour).

3. Pour chaque période et pour chaque paire, construire le vecteur d'agrégation suivant : [Nombre (nb) de logs de la paire (sourceIp, destinationIp), nb de logs de source ports différents, nb de logs avec le destinationPort D1, D2, … D13, Minimum Sent Bytes, Maximum Sent Bytes, Moyenne Sent Bytes, Somme Sent Bytes, Minimum Received Bytes, Maximum Received Bytes, Moyenne Received Bytes,

Somme Received Bytes, Bit d'attaque (0 lorsque l'ensemble des logs sur l'agrégation considérée ne contient aucune attaque, 1 lorsque ce même ensemble contient au moins un log d'attaque)].

4 Algorithmes étudiés

4.1 Random Forest (RF) [18]

RF est un algorithme performant pour la classification et qui entraine un ensemble d'arbres de décision sur un sous-ensemble d'échantillons et de caractéristiques choisies aléatoirement pour chaque arbre. La classification est réalisée après un vote sur les arbres de la forêt. RF a également été choisi pour ses performances significatives en classification, son implémentation optimisée dans plusieurs langages de programmation (notamment Spark MLlib et Python), le faible niveau de ressources nécessaires pour l'utiliser et une interprétabilité meilleure que pour les réseaux de neurones [20]. Ses autres avantages incluent une résistance au bruit et une certaine efficacité face à des jeux de données déséquilibrés.

RF est piloté par plusieurs paramètres importants comme le nombre d'arbres dans la forêt (estimators), la profondeur de chaque arbre (max_depth), le split ratio (pourcentage de données pour l'entrainement et le test), le type de split (stratify) et le sample weighting (afin de mieux considérer des classes sous représentées).

En considérant les paramètres suivants (estimators=10, max_depth=10, train_size=0.75, stratify=None, class_weight=Non) et sans calibration spécifique, RF fournit des résultats très moyens sur le dataset CIC IDS 2017, et se révèle incapable de détecter certaines attaques.

La figure ci-dessous présente la matrice de confusion associée. L'exactitude (au sens accuracy définie comme le total des prédictions correctes divisé par le nombre total de prédictions) moyennée par classe arrive à peine à 65%.

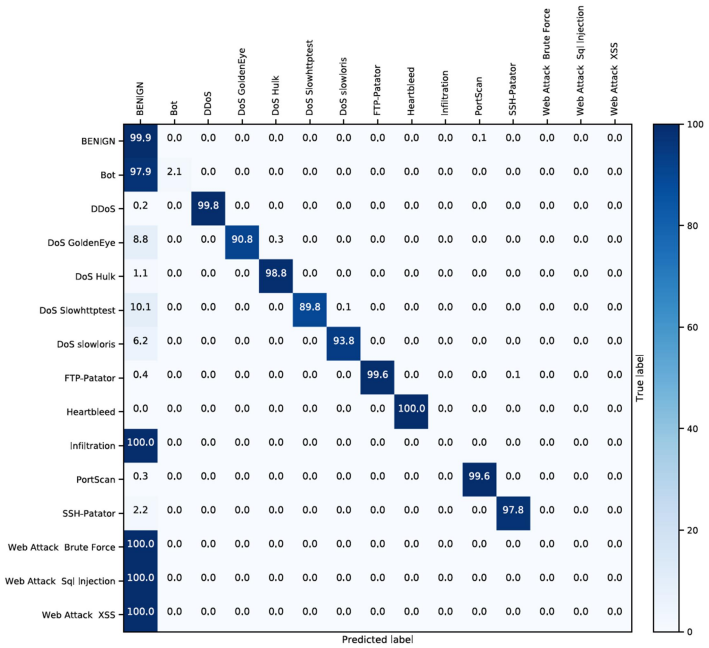

Fig. 1. Matrice de confusion / Random Forest sans optimisation / CIC IDS 2017

En positionnant class_weight à balanced (afin que mieux considérer les différentes attaques), la moyenne par classe augmente à 92%.

L'activation du stratify conduit à une diminution de la moyenne par classe (91%) mais équilibre bien mieux les échantillons ce qui permet d'éliminer le biais dans l'apprentissage : l'intérêt est que les classes d'attaques sont mieux prédites (autrement dit, on évite que le précédent modèle ne soit efficace que sur certaines classes d'attaques sur-représentées). Cependant, cela n'élimine pas le fait de la différence de représentativité des attaques au niveau du dataset: au moins une attaque reste sous-représentée (heartbleed).

Les logs normaux (benign, c'est-à-dire des logs ne représentant pas une attaque) sont sur-représentés dans le dataset initial. Un sous-échantillonnage de ces logs permet d'améliorer l'exactitude : nous avons choisi de garder autant de logs benign que de logs totaux d'attaques, les logs à supprimer ayant été choisis de façon aléatoire. Après entrainement, la moyenne par classe s'établit à 93%.

Nous avons ensuite fait varier le nombre total d'arbres et avons testé les valeurs suivantes : 5, 10, 20, 50 et 100. Le tableau suivant récapitule les différents résultats avec deux autres métriques (une sur l'exactitude globale et une pour chaque attaque).

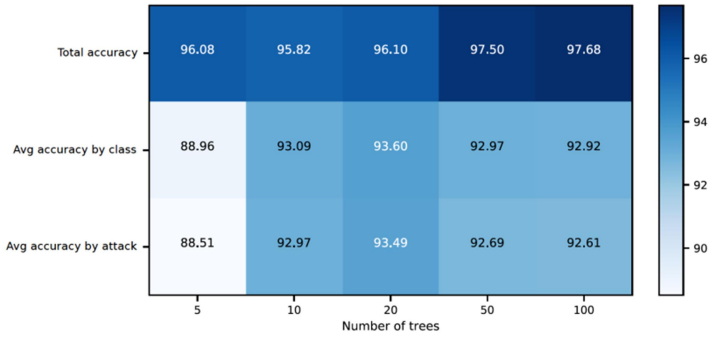

Fig. 2. Exactitude en fonction du nombre d'arbres pour RF

Le meilleur résultat global est obtenu pour 20 arbres. Si un objectif est d'obtenir le moins de faux-positifs possibles (une demande régulière de la part des équipes de production), le meilleur résultat est obtenu pour 100 arbres, même s'il est nécessaire de s'assurer qu'il n'y ait pas de sur-apprentissage à ce niveau.

La profondeur des arbres est également un paramètre important. Nous avons testé avec le paramètre None puis avec des valeurs comprises de 5 à 30 par pas de 5 (la valeur None correspond à une expansion de l'arbre jusqu'à que les feuilles ne contiennent plus qu'une seule valeur). Ceci peut conduire à un sur-apprentissage, donc non pertinent pour généraliser l'apprentissage en production mais ce choix constitue un bon test pour valider les autres profondeurs.

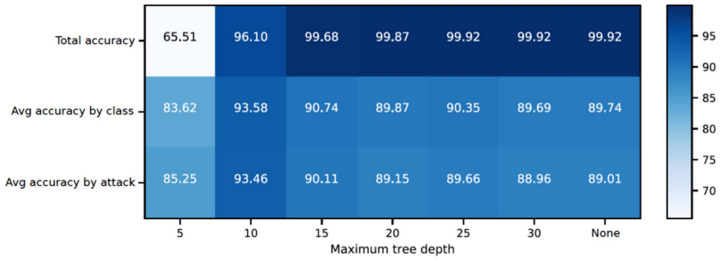

Fig. 3. Exactitude en faisant varier la profondeur des arbres RF

Une profondeur de 10 semble constituer le meilleur compromis pour ce dataset.

Avec ces différents paramètres, la matrice de confusion obtenue pour les différentes attaques est la suivante :

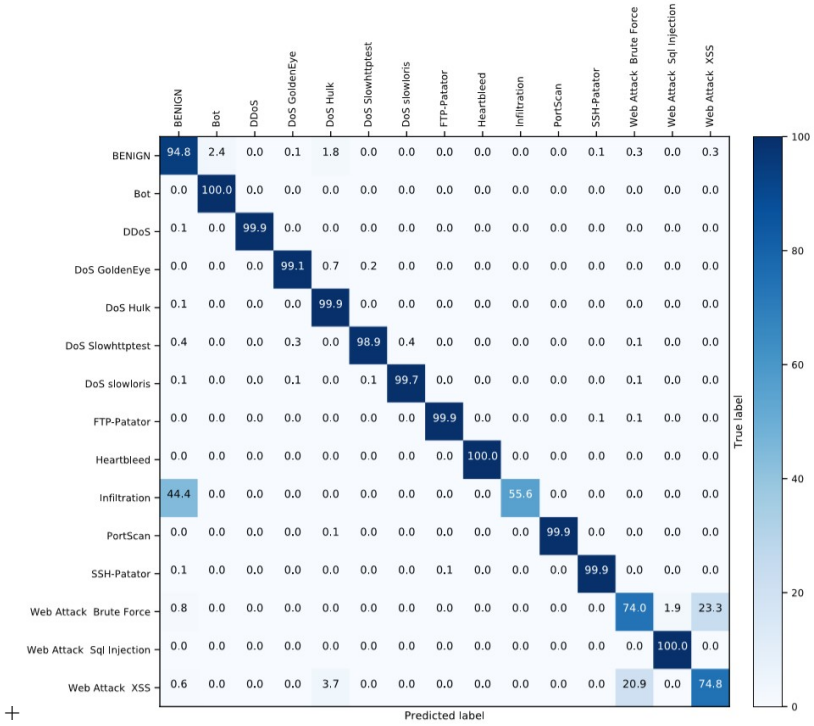

Fig. 4. Matrice de confusion après optimisation.

Ces premiers résultats montrent une détection satisfaisante sur une majorité des attaques. En revanche, certaines attaques sont mal détectées (attaques Web et Infiltration).

Les études qui ont suivi, ont consisté à grouper les attaques soit par classes, soit en deux catégories : attaques et benign. Les attaques Web (Brute Force, Sql Injection et XSS) ont été groupées dans la famille Web et les attaques DoS (Hulk, GoldenEye, Slowloris et SlowHttptest) ont été groupées dans la famille DoS (différente de DDoS). Les mesures d'exactitude sont données ci-dessous :

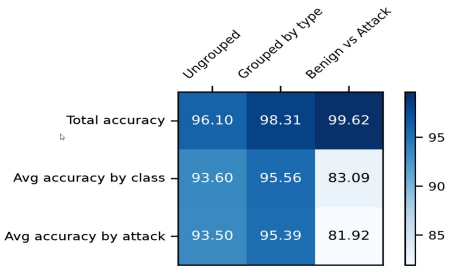

Fig. 5. Mesure d'exactitude avec les groupements des attaques

4.2 Isolation Forest

Isolation Forest [12] est un algorithme intéressant à plusieurs égards. Il s'appuie sur des arbres de décision ce qui rend l'interprétation de ses résultats plus simple qu'avec les réseaux de neurones. Il est orienté détection d'anomalie ce qui lui permet d'être plus rapide dans leur détection et moins consommateur en terme de ressources informatiques (les anomalies sont en effet proches de la racine des arbres, ce qui limite par construction leur profondeur). Il est également performant sur des espaces de grandes dimensions ce qui permet d'envisager des évolutions opérationnelles dans la prise en compte de nouvelles variables d'entrée.

Isolation Forest n'étant pas disponible sous Spark MLlib, nous avons codé notre propre implémentation à partir de fichiers sources disponibles sur Internet, notamment sur GitHub [17].

4.3 Comparaison de Random Forest et d'Isolation Forest

Ce paragraphe compare les résultats obtenus avec Random Forest et Isolation Forest sur notre dataset interne (Cybertest). Les données opérationnelles n'étant jamais labélisées, l'objectif ici est de déterminer si l'algorithme Isolation Forest est pertinent en production.

Compte-tenu de notre volonté de disposer de modèles portables entre plusieurs datasets, il est nécessaire de réduire le nombre de features utilisées lors de l'optimisation de Random Forest. Nous ne retenons donc que les features indiquées au paragraphe 3.2. Parmi les features brutes, notons que deux d'entre elles sont considérées comme étant les deux plus importantes par Random Forest : port de destination et port source. Cette réduction de dimension nous impose de ré-entrainer notre modèle Random Forest, en particulier au niveau de la profondeur des arbres. En proposant moins d'informations en entrée, les arbres doivent être plus profonds pour séparer correctement les différentes classes en entrée. Après optimisation, une profondeur de 20 (contre 10 auparavant) donne le meilleur compromis performance/sur-apprentissage.

Nous avons choisi de prendre, dans un premier temps, les mêmes hyperparamètres pour Random Forest et pour Isolation Forest. Nous justifions ce choix par le fait que les structures mathématiques de ces deux algorithmes sont proches (i.e. des arbres de décision). Une étude plus approfondie consisterait à faire varier ces hyperparamètres mais une application directe nous semble ici intéressante.

Les tableaux ci-dessous présentent les attaques labélisées par les experts métiers sur une base du quart d'heure ainsi que les comparaisons entre les 2 modèles. L'application du modèle Isolation Forest sur le dataset Cybertest a fait remonter les anomalies suivantes :

#	IP	Labélisé comme attaque	Attaque	Probabilité Random Forest	Commentaires
1	10.131.60.121	Oui	Scan port	0.99	IF et RF similaires
2	10.131.60.131	Non	Trafic normal	0.63	Faux positif (IF et RF)
3	10.131.60.142	Non	Mise à jour base antivirale	0.63	Faux positif (IF et RF)
4	10.131.60.148	Oui	Exploit kit	0.66	IF et RF similaires
5	10.131.60.154	Oui	Exploit kit	0.62	IF et RF similaires
6	10.131.60.160	Oui	Téléchargement virus	0.97	RF plus performant
7	10.131.70.127	Oui	Exploit kit	0.63	IF et RF similaires
8	10.131.70.128	Oui	Exploit kit	0.60	IF et RF similaires
9	10.131.70.130	?	Tunneling	0.62	IF et RF similaires
10	10.131.70.132	Non	Trafic normal	0.62	Faux-positif (IF et RF)
11	10.131.70.140	Oui	Exploit kit	0.62	IF et RF similaires
12	10.131.70.143	Oui	Exploit kit	0.71	IF et RF similaires
13	10.131.70.145	Oui	Exploit kit	0.8	IF et RF similaires
14	10.131.70.147	Oui	Exploit kit	0.56	IF et RF similaires
15	10.131.70.152	Oui	Exploit kit	0.77	IF et RF similaires
16	10.131.70.156	?	Tunneling	0.56	IF et RF similaires
17	5.42.176.145	Oui	Scan port	0.62	IF et RF similaires
18	101.10.60.09	Non	LAN proxy	-	Faux-positif IF uniquement
19	172.17.0.1	Non	Trafic normal	-	Faux-positif IF uniquement
20	172.22.255.255	Non	Trafic normal	-	Faux-positif IF uniquement

Table 1. Comparaison qualitative des anomalies obtenues entre Random Forest et Isolation Forest sur le dataset Cybertest.

Dans ce tableau, les comportements qui diffèrent entre Random Forest et Isolation Forest sont indiqués en gris. D'une manière globale, les deux modèles se comportent de façon similaire et détectent les mêmes attaques et les mêmes faux-positifs (lignes 2, 3 et 10). Notons quand même pour Isolation Forest, la non détection d'un téléchargement de virus (ligne 6) et trois (3) faux-positifs remontés (lignes 18, 19 et 20).

Nous avons la certitude que notre dataset interne ne présente pas d'attaques pendant une période donnée. Il est donc possible d'extrapoler le nombre total de faux-positifs calculés par Isolation Forest à partir du nombre de faux-positifs trouvés sur cette période saine. Ce taux extrapolé de faux-positifs atteint ainsi 20,8%.

4.4 Réseaux de neurones

Plusieurs expérimentations ont été menées sur des réseaux de neurones avec néanmoins une cible d'utilisation différente de celle de Random Forest (ou d'Isolation Forest). Notre idée principale, à ce jour, est que le manque d'interprétabilité des résultats des réseaux, les rendent peu utilisables seuls dans un contexte opérationnel pour de la détection d'anomalie. Il s'agit donc de combiner leurs résultats avec des algorithmes à base d'arbres de décision dans le but de confirmer une attaque et/ou d'éliminer des faux-positifs. Une évolution naturelle de cette approche consiste à utiliser des méthodes d'ensemble learning [9] pour améliorer les prédictions.

Nous avons également prévu d'explorer deux autres voies qui pourraient remettre en question cette posture. D'une part, les travaux dans le domaine de l'attention mechanism (cf. par exemple [10]) devraient pouvoir identifier les parties les plus représentatives des données d'entrée et donc fournir aux analystes des pistes d'exploration suite à une détection d'anomalies à l'aide de réseaux. D'autre part, la spécialisation des réseaux de neurones sur une attaque particulière ne devrait pas nécessiter la mise à disposition d'éléments contextuels supplémentaires pour les opérationnels : une fois l'attaque identifiée par le réseau, l'analyste métier peut ainsi recourir à son expertise pour confirmer ou non cette attaque.

Pour construire et entrainer nos réseaux, nous avons fait varier à la fois leur architecture et l'algorithme d'apprentissage. Nous avons utilisé la librairie Python Keras [19] sur les vecteurs d'agrégation générés à partir du dataset public labélisé CIC-IDS 2017 sur l'ensemble des 80 features et sur un nombre réduit de features (cf. 3.2).

Nous nous sommes aussi limités dans un premier temps aux réseaux de neurones simples à 3 couches cachées. La couche d'entrée du réseau possède autant de neurones que de features (soit 80, soit 24) et la couche de sortie en contient toujours 2 pour indiquer la probabilité d'attaque et la probabilité d'être bénin. Les neurones sont interconnectés de la manière dite « fully-connected ». Autrement dit, chaque neurone d'une couche donnée prend en entrée toutes les sorties des neurones de la couche précédente. La fonction d'activation est la fonction ReLU. Parmi les algorithmes de descente de gradient stochastique, nous avons expérimenté la méthode de gradient adaptatif et l'algorithme d'optimisation Adam.

Pendant toute la phase de tests, certains paramètres sont restés identiques. Ainsi, dans un premier temps, nous avons testé plusieurs architectures en fixant l'algorithme d'apprentissage et ses paramètres associés (Learning Rate par exemple). Nous avons alors fait varier le nombre de neurones sur chaque couche cachée. Un nombre trop réduit de neurones (inférieur à 10) ou trop élevé (supérieur à 500) ne conduisait pas à de bons résultats. Après plusieurs essais en faisant varier le learning rate, l'algorithme de gradient adaptatif avec un learning rate de $2x10^{-5}$ a donné les meilleurs résultats.

Sur les 80 features du dataset CIC IDS, nous obtenons une précision (au sens precision[1] en anglais) de 0,98 et un recall[2] de 0,97, en phase avec les résultats initiaux [1]. Dans le cadre d'une application opérationnelle, les réseaux ont été ré-entrainés avec les 24 features indiquées au paragraphe 3.2. La précision s'est établie à 0,95 et le recall à 0,47. Une précision élevée est un résultat intéressant car cela limite le nombre de faux-positifs, ce qui est recherché par les opérationnels. En revanche, le recall n'est pas satisfaisant. Nous pensons que le déséquilibre des données dans le dataset initial est en partie responsable de ces résultats mitigés. Une première piste explorée a consisté à sur-échantillonner le nombre d'attaques afin de rééquilibrer les différentes classes. L'algorithme Adasyn [13] a été utilisé pour générer de façon synthétique

---

[1] Precision = TP/(TP+FP), TP est le nombre de vrais positifs et FP le nombre de faux positifs

[2] Recall = TP/(TP+FN), TP est le nombre de vrais positifs et FN le nombre de faux négatifs

environ 8% de données supplémentaires. La précision est alors passée à 0,98 et le recall à 0,802. Ces résultats sont encourageants même si cette technique augmente le temps d'apprentissage et introduit le risque d'un sur-apprentissage.

5 Intégration en environnements contraints

5.1 Préambule

Le terme « diLAN » est le nom de code du projet visant à intégrer des solutions d'IA au niveau des équipes opérationnelles. Le résultat concret de cette intégration est la production d'un score donnant soit la probabilité d'une attaque (algorithme supervisé) soit un score d'anomalie (algorithme non supervisé). Hormis les réseaux de neurones qui, pour l'instant ont été développés en Python, nos autres algorithmes d'IA ont été construits dans un environnement Scala/Spark.

Dans la situation où des labels sont disponibles, les algorithmes supervisés sont en général plus performants que les non-supervisés. En production, les labels ne sont presque jamais disponibles. La labélisation peut néanmoins s'envisager a posteriori, en mettant en place une boucle de retour associant les résultats des algorithmes d'IA aux conclusions des analystes de sécurité sur un incident donné (comme proposé dans le paragraphe 3.1).

5.2 Contraintes d'intégration

La première contrainte « naturelle » est celle de l'empreinte mémoire et CPU des algorithmes déployés et ce, même au sein d'un SIEM : les SIEMs sont en effet très souvent sollicités par les opérationnels pendant leurs investigations avec des attentes fortes en terme de temps de réponse. Ajouter directement des processus qui viendraient requêter les SIEMs avec des calculs statistiques pour les algorithmes d'IA conduirait inévitablement à un allongement de ces temps de réponse avec comme conséquence probable un rejet de ces algorithmes. Nous proposons dans ce paragraphe une première approche pour réduire cet impact avec une déclinaison pour les SIEMs IBM QRadar et Splunk.

Un autre type de contrainte est lié à la consommation de ressources externes. Pour faciliter l'intégration d'algorithmes d'IA (qui requièrent beaucoup de données à consommer) le choix a été fait de se connecter aux APIs[3] des SIEMS. Les données nécessaires y sont en effet centralisées et utiliser ces APIs facilitent grandement le déploiement rapide de ces algorithmes. Il ne devient pas nécessaire de dupliquer les flux de données déjà existants pour ces algorithmes. Les données statistiques peuvent alors être calculées directement par les SIEMs, ce qui limite les ressources nécessaires au niveau des machines hébergeant les algorithmes. Ces derniers peuvent également s'appuyer sur des APIs de type Threat Intelligence par exemple dont l'utilisation est

[3] Application Programming Interface

généralement soumise à quotas. Dans ces deux cas, il est important de solliciter de façon intelligente ces APIs afin de ne pas saturer les systèmes interrogés. Ces derniers doivent en effet rester très réactifs pour ne pas pénaliser les investigations menées par les analystes. Nous proposons ici quelques pistes dans le cadre du SIEM IBM QRadar même si elles peuvent s'appliquer à d'autres contextes.

La troisième intégration présentée n'impose pas nécessairement de restrictions en termes d'empreinte CPU et mémoire, mais nécessite une architecture logicielle spécifique. La solution Open-Source Apache Metron fournit notamment un cadre pour intégrer des algorithmes d'enrichissement de données (Threat et scoring d'anomalie). L'objectif ici est de rendre compatible le mode streaming d'Apache Metron avec le calcul de scores issu de nos algorithmes d'IA. Après une présentation de son architecture, nous précisons la façon dont nous avons prévu d'intégrer nos algorithmes à cette infrastructure logicielle. Dans ce cadre, l'utilisation des APIs des SIEMs n'est plus indispensable (même si celle-ci peut être maintenue). Il reste nécessaire d'alimenter les algorithmes à l'aide de broker d'évènements spécialisés (Apache Kafka par exemple), mais sans nécessité de stockage de l'ensemble des données (car les traitements se font en mode streaming)

Enfin, la dernière intégration présentée est une déclinaison des architectures précédentes dans un contexte très contraint comme l'écosystème de la Livebox. Nous pensons que certains algorithmes d'IA pourraient avoir un impact bénéfique pour nos clients Grand-Public et PME et le sujet de l'intégration de ces algorithmes avec un minimum d'impact sur l'existant mérite d'être étudié.

5.3 Intégrations QRadar et Splunk

La première itération afin de rendre possible l'intégration de nos algorithmes dans nos environnements contraints a consisté à retravailler notre architecture logicielle en nous calquant simplement sur les processus mis en œuvre dans une démarche de Machine Learning : une séparation de la partie apprentissage et de la partie scoring (i.e. le fait d'appliquer un ou plusieurs modèles sur les données entrantes). Cette découpe a l'avantage de la simplicité et de faciliter la communication entre les data scientists et les experts IT.

L'autre avantage est que l'intégration dans un environnement contraint en termes de ressources IT s'en trouve grandement facilitée. La phase d'apprentissage requiert beaucoup de puissance de calcul alors que la phase de scoring n'en nécessite que très peu. Le processus de scoring pourra donc être positionné au plus près des opérationnels et éventuellement développé dans un autre langage compatible avec leur environnement IT. En revanche, le processus de calcul de modèles (l'apprentissage) devra être déporté sur des machines plus performantes et non forcément co-localisées avec les ressources opérationnelles.

La vision haut-niveau de cette intégration au sein d'un SIEM est donnée ci-dessous :

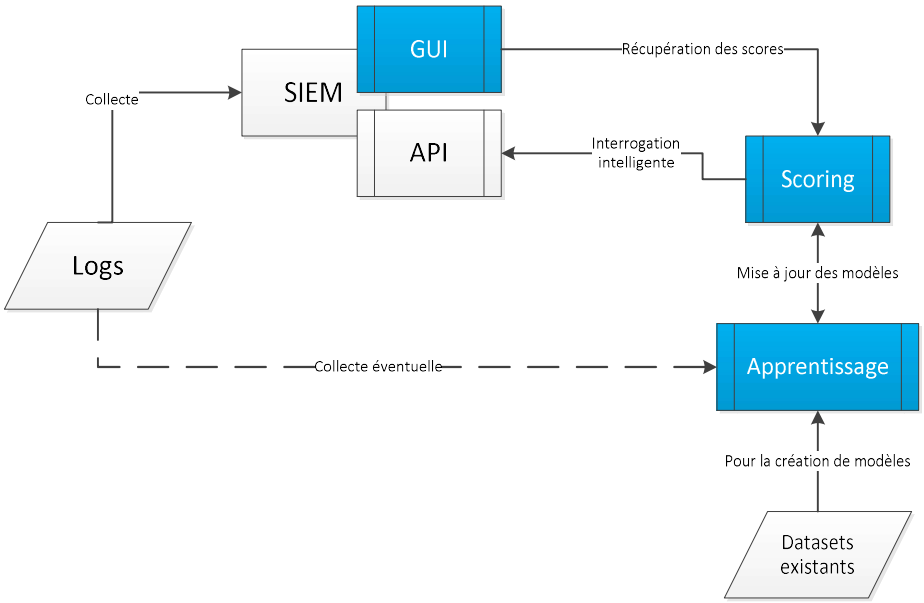

Fig. 6. Architecture logique générique

L'architecture est ainsi décomposée en 3 composants logiques : (i) l'interface utilisateur (GUI), (ii) la partie dont le rôle est de calculer un score d'anomalie ou une probabilité de classification (Scoring), et (iii) la partie en charge de calculer les modèles. La mise en place d'API (Application Programming Interface) au niveau des composants (ii) et (iii) permet d'envisager une distribution opportune des algorithmes et notamment une adaptation à des environnements contraints. L'idée principale consiste donc à positionner les composants (i) et (ii) sur des systèmes compacts et consommant peu.

Même si, par nature, ces composants requièrent peu de ressources, il est quand même nécessaire de gérer de façon fine les différents flux de données. En fait, la principale difficulté qui émerge suite à cette architecture logicielle est celle du calcul et de la maintenance des modèles, avec notamment l'interrogation intelligente des APIs.

QRadar

Nous disposons d'une instance de pré-production du SIEM IBM QRadar en version v7.3.1. Cette instance nous sert à qualifier les solutions de sécurité d'IBM et à expérimenter l'intégration de scores issus de nos algorithmes d'IA, afin de pouvoir très rapidement les intégrer aux instances de production.

L'intégration fine au sein de QRadar, nécessite le développement d'une application QRadar spécifique (« une QApp »). Deux types de déploiement sont possibles pour cette QApp : soit au même niveau que l'instance de QRadar (« la Console »), soit sur une machine dédiée.

Un déploiement sur la Console contraint assez fortement les ressources disponibles pour la QApp qui ne peut demander qu'au maximum 10% de la mémoire. Les ressources demandées sont déclaratives (dans le Manifest.json de la QApp) sans qu'il soit possible pour la QApp de dépasser les ressources déclarées.

La figure ci-dessous présente l'architecture logique d'une QApp qui est automatiquement déployée dans un Docker par QRadar. Un reverse-proxy assure le routage des URLs vers la bonne instance de la QApp.

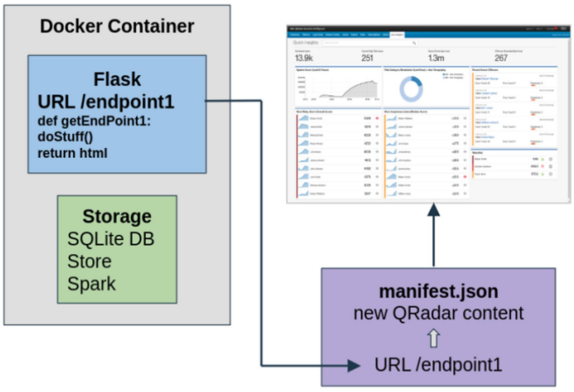

Fig. 7. Architecture logique d'une QApp

L'environnement d'exécution se limite à Python 2.6.6 sur CentOS 6.8 (même si nous avons utilisé Python 2.7 avec succès). Les requêtes entrantes au niveau de la QApp sont gérées par le framework Flask [15].

L'autre solution consiste à utiliser une machine dédiée : un « Appnode », même si notre version de QRadar n'autorise qu'un seul Appnode. Ainsi, même sur une machine dédiée, nos algorithmes d'IA devront partager les ressources disponibles avec d'autres QApp potentiellement gourmandes en ressources (sinon, elles auraient été déployées sur la Console).

La figure suivante présente l'interface principale de diLAN intégrée dans QRadar :

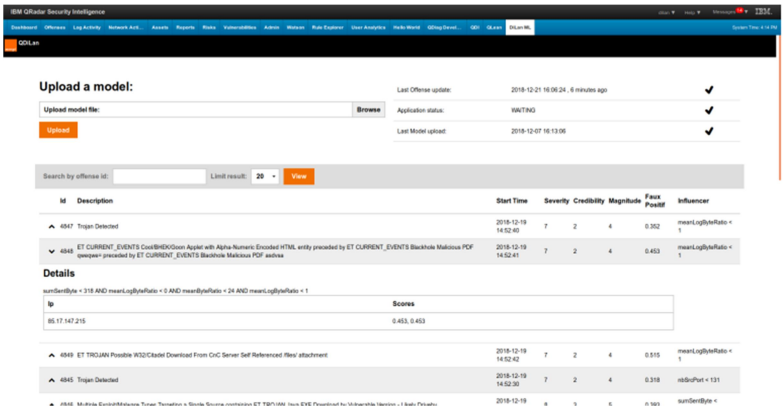

Fig. 8. Intégration diLAN au sein de QRadar

Splunk

Splunk est un logiciel de stockage, de traitement et d'analyse de données. Employé comme SIEM, il s'avère être un outil très efficace et apprécié, notamment par les analystes cybersécurité. Cette intégration a pris la forme d'un plugin Splunk.

La réalisation de l'application s'est révélée grandement facilitée par les outils de développement mis à disposition par Splunk. Aussi, nous avons pu réutiliser, à la fois l'architecture réalisée pour QRadar ainsi qu'une partie du code de l'application QRadar. Le développement de l'application diLAN pour Splunk s'est donc essentiellement portée sur les liens à faire entre l'API Splunk et les scripts déjà existants. L'architecture de l'application est représentée ci-dessous.

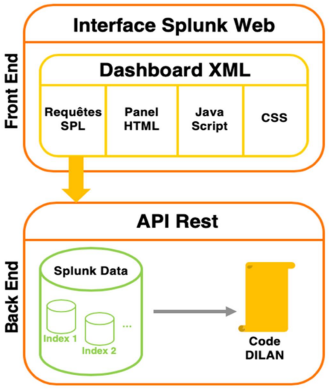

Fig. 9. Architecture de l'application diLAN intégrée à Splunk

L'application est aujourd'hui opérationnelle et permet, depuis l'interface Web, de lancer une analyse des logs contenus dans Splunk en utilisant les algorithmes de diLAN et de scorer les différentes « Sources Ip » en cours d'investigation par les experts métier. L'illustration ci-dessous présente l'interface principale de ce plugin.

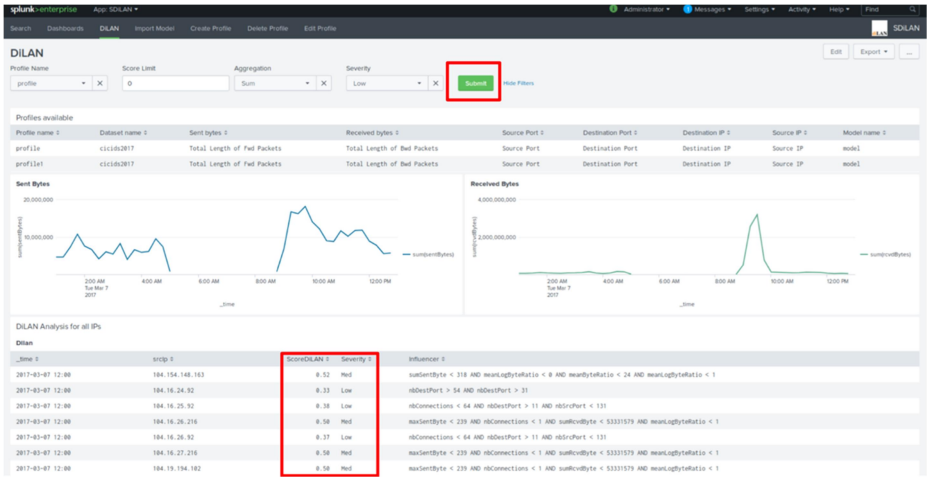

Fig. 10. Interface principale de diLAN pour Splunk

Apache Metron [4]

Apache Metron est un framework de sécurité informatique permettant aux entreprises d'intégrer, de traiter et de stocker divers flux de données afin de détecter les anomalies et de permettre aux organisations de réagir plus rapidement.

Il fournit une structure d'analyse de sécurité évolutive fondée sur les technologies Hadoop. Il permet de surveiller le trafic réseau et les journaux machines en consommant des flux continus de données [3], ce qui en fait une solution de choix pour des traitements en quasi temps-réel (dont l'alerting).

Nous avons installé Apache Metron sur un cluster de 4 nœuds, chaque nœud disposant de 64 Gb RAM et de 40 cœurs CPU. Le stockage total (extensible) est actuellement de 12 To.

Les contraintes pour intégrer nos algorithmes d'IA ne sont pas matérielles dans ce cas. Si nous voulons profiter du potentiel d'Apache Metron, et en particulier profiter de son architecture Storm (streaming de données), la principale contrainte imposée est de s'intégrer à son framework MaaS (Model As a Service). Cette intégration permet d'intégrer nos propres algorithmes d'IA potentiellement consommateurs de ressources IT : Apache Metron va ainsi gérer la montée en charge et le Load Balancing vers nos algorithmes déployés. Il va également permettre d'entrainer et d'ajuster les modèles

existants en parallèle des flux entrants. Une description de cette architecture peut être trouvée en [5].

Pour réussir cette intégration, l'objectif est que nos algorithmes de scoring répondent en quelques millisecondes. Même si l'architecture est scalable par construction, l'atteinte de cette performance n'est pas forcément triviale. Comme indiqué précédemment, tous les algorithmes ont été développés en Scala/Spark. Or l'initialisation d'un contexte Spark est un processus très consommateur en temps (de l'ordre de 200 ms), ce qui le rend incompatible avec les performances attendues. Dans l'architecture MaaS de Metron, chaque instance de nos modèles doit potentiellement initialiser et conserver un contexte Spark, ce qui n'est pas toujours possible.

Bien que non finalisée à l'heure actuelle, cette intégration se présente de la façon suivante :

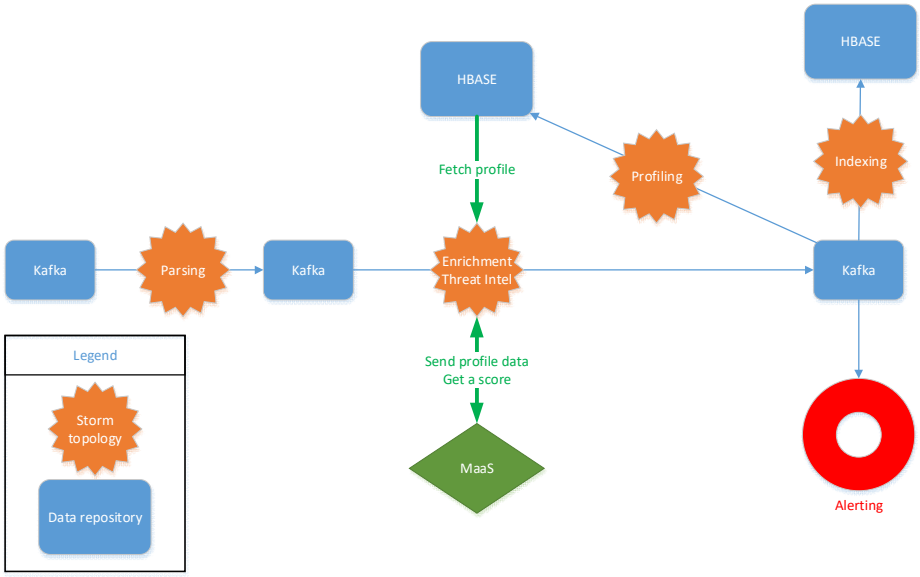

Fig. 11. Architecture globale avec Apache Metron

- Les données brutes sont injectées dans Apache Kafka via diverses solutions en fonction des contextes (Rsyslog, Apache Nifi [16], scripts,…). Apache Kafka est le point d'entrée de la solution Metron.
- Les données sont ensuite traitées via différentes topologies Storm afin de les analyser, les enrichir et les stocker.
- En parallèle, le « profiler » de Metron permet de calculer des agrégations (ou profils) qui seront utilisées lors de la phase de prédiction via le Machine learning
- En fonction des résultats de la Threat Intelligence, utilisant ou non le Machine learning, il est possible de taguer un évènement en tant qu'alerte et de le propager ensuite dans le système adéquat (par exemple via Apache Nifi)

Les 2 principales difficultés à résoudre sont les suivantes :

- Utilisation du profiler Metron: nos algorithmes utilisent des agrégats (ou profils) afin de réaliser un calcul de score sur une période de temps donnée. Il serait possible de réaliser ce genre d'agrégation en utilisant Spark Streaming, mais cela nécessite de redévelopper une fonctionnalité a priori présente dans Metron. La configuration du profiler se faisant via un fichier JSON, nous devons être en mesure de recoder via JSON les agrégations initialement développées en Scala. D'autre part, l'accès aux données du profiler, stockées dans HBASE, ne peut se faire que via le langage STELLAR dans le cadre du workflow de traitement d'un évènement. Là encore il faut s'assurer que les 2 formalismes sont compatibles.

- Machine Learning : autant il est tout à fait possible de réaliser la phase d'apprentissage via Spark, ce qui présente un réel intérêt vues les quantités de données à traiter, autant il parait peu judicieux d'utiliser Spark en prédiction, et ce même si le contexte Spark est préalablement chargé. Les temps de réponse sont de l'ordre de la centaine de ms, ce qui n'est pas acceptable. C'est pourquoi nous avons étudié la solution MLeap qui nous permet d'exécuter des pipelines préalablement définis et sauvegardés en Spark. En d'autres termes, les phases de Machine Learning se déroulent de la façon suivante : (i) définition du pipeline de traitement dans Spark : sélection des features, calcul des arbres Isolation Forest basés sur les données d'apprentissage, (ii) sauvegarde de la configuration du pipeline et du modèle (arbres) dans un format compatible MLeap (bundle MLeap), (iii) copie de ce bundle dans l'environnement du MaaS, (iv) initialisation d'un service REST utilisant MLeap pour calculer un score d'anomalie sur les données reçues en paramètre du service REST. La phase de prédiction s'effectue alors en quelques millisecondes.

Au final, nous allons intégrer diLAN dans la solution Apache Metron en bénéficiant au maximum des fonctionnalités proposées par cette solution (profiling, MaaS, Threat Intelligence,..) et focaliser nos développements au maximum sur la partie algorithmes de Machine Learning.

Livebox

L'environnement autour de la Livebox est un écosystème intéressant à développer dans le cadre de la détection d'anomalie pour plusieurs raisons.

- La première est liée à la volumétrie sous-jacente : plusieurs millions de routers[4] peuvent être utilisés afin de détecter des anomalies ce qui rendra le système consolidé particulièrement robuste,
- la seconde est liée à la localisation des routeurs : elles se trouvent très proches des sources de données et des cibles initiales des attaques ce qui fiabilisera les données d'analyse. Par ailleurs, les données brutes des utilisateurs peuvent ne pas sortir du

[4] Dans cette partie, les termes Routeur et Livebox sont interchangeables

réseau local, ce qui présente un avantage intéressant en termes de respect de la vie privée. A noter également que pendant les phases de mises au point des algorithmes de Machine Learning, les données consolidées et remontées au niveau du Edge sont bruitées (via diffential privacy) sans que cela n'impacte la détection d'anomalies au niveau de l'ensemble des routeurs (le bruit est lissé grâce à la quantité des données).

- La dernière est économique. S'appuyer sur les techniques actuelles de collecte de données pour la détection d'anomalies (c'est-à-dire en stockant de façon centralisée des données au niveau d'un SIEM) sur le réseau Orange France ferait exploser les coûts des serveurs notamment avec le déploiement massif de la fibre aujourd'hui au Gb/s et demain avec des débits de 10Gb/s. Le fait d'utiliser les ressources hardwares des Liveboxes va permettre d'envisager une collecte importante à des coûts contrôlés, les calculs d'agrégats pouvant se faire directement au niveau de ces routeurs.

Cet écosystème apporte néanmoins son lot de complexité : les routers ont des performances inférieures aux serveurs dans le Cloud, leur fréquence de mises à jour est moins soutenue que sur des serveurs et la priorité pour ces routeurs est la qualité des services de type Internet, VoIP et TV et non la collecte et le traitement de la donnée pour de la détection d'anomalie.

Afin de répondre à ces contraintes, une architecture inédite a été développée ; le principal avantage de cette architecture est de proposer une architecture hybride composée d'éléments intégrés dans les routeurs, d'éléments en bordure de réseau (Edge Computing) et d'éléments dans le Cloud. Le Edge va gérer un ensemble de Liveboxes (entre 100 000 et 1 million) avec une notion de « plaque » afin de structurer à la fois l'apprentissage et le déploiement de modèles de façon souple. Le Cloud peut venir ensuite agréger plusieurs plaques. Le schéma ci-dessous illustre l'architecture envisagée :

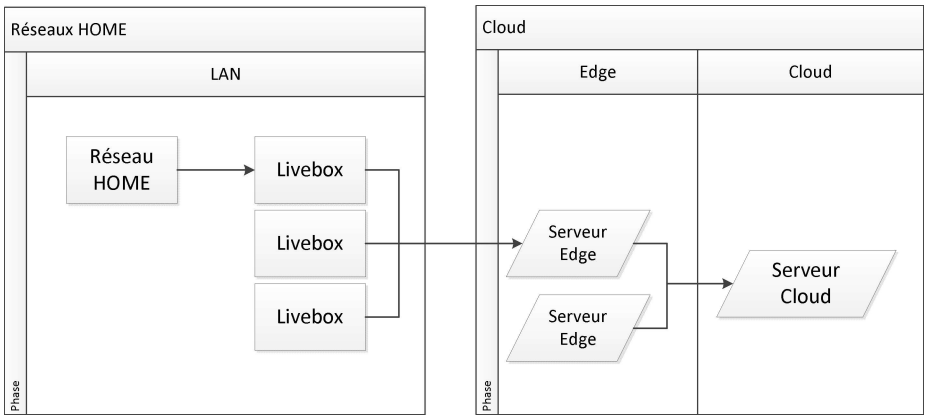

Fig. 12. Architecture schématique envisagée pour la collecte des données des Liveboxes

- La première étape consiste à collecter de la donnée de qualité en temps réel. Le temps réel est important car il va nous permettre dans un second temps d'exécuter des algorithmes au niveau du Edge afin de réaliser des actions rapides de remédiation (mise en quarantaine de certains flux malveillants par exemple)
- La seconde étape est l'entrainement et l'exécution des modèles. Une fois l'entrainement fait au niveau du Edge, il est possible d'exécuter ces modèles toujours depuis le Edge et de lancer des actions sur la Livebox pour affiner la collecte de certaines données ou bloquer des flux. L'avantage de cette phase est qu'il est possible de mettre à jour les algorithmes de manière agile et régulière car ces mises à jour se font sur le Edge.
- Une fois les algorithmes considérés comme stables, la phase d'optimisation peut commencer. Cette étape consiste à l'écriture d'un programme qui pourra être exécuté dans un environnement contraint de la Livebox. A la fin de cette phase, les algorithmes peuvent être déployés sur des millions de Liveboxes pour agir au plus près des sources de données. Si nécessaire, à ce stade, aucune donnée utilisateur gérée au niveau de routeurs sélectionnés ne remontera au niveau du Edge.
- Dans la phase de production, il est néanmoins important qu'une proportion significative de Liveboxes soit toujours pilotée depuis le Edge. Cela permet de développer de nouveaux algorithmes, de ré-entrainer les anciens afin de détecter de nouvelles anomalies tout en conservant les précédentes capacités de détection

Cette architecture hybride permet aussi d'envisager la remontée des modèles et des informations consolidées de chaque plaque vers le Cloud pour des actions à froid (threat hunting, validation inter-plaques voir optimisation si nécessaire).

6 Travaux futurs

Ces travaux ont présenté une première étape de déploiement de bout en bout d'algorithmes de Machine Learning avec une intégration au sein d'infrastructures de production. Nous avons pu identifier un certain nombre de difficultés mais nous disposons, à ce jour, d'une première chaine complète et opérationnelle. L'objectif consiste désormais à travailler de concert avec les équipes techniques et d'améliorer les éléments clés de cette chaine technique en fonction des retours terrains.

En plus d'une expérimentation sur des données de production qui a débuté en septembre 2019, nous avons identifié plusieurs axes de travail :

- L'intégration dans Apache Metron afin de valider la détection en mode streaming (avec intégration également de bases de Threat Intelligence)
- Une expérimentation grandeur réelle sur un parc contrôlé et significatif de Liveboxes en production (donc fournissant des données non labélisées). Un candidat idéal pour les prédictions (voire l'entrainement, à confirmer) est l'algorithme de Machine Learning non supervisé Isolation Forest, pour ses performances mais surtout ces caractéristiques intrinsèques : cet algorithme, par construction, n'a besoin que de peu de données d'entrainement (256 échantillons est la valeur empi-

rique présentant le meilleur compromis performance/consommation de ressources IT [12]) et il ne requiert pas une profondeur importante au niveau des arbres (puisque les anomalies sont localisées proches des racines des arbres).

- Afin d'affiner nos algorithmes, des expérimentations supplémentaires sont prévues sur d'autres data sets comme le nouveau data set fourni par le CIC [2]. Il est également envisagé d'étudier le data set KDD [14] qui sert souvent de référence pour l'évaluation de la performance des algorithmes de Machine Learning, même si la qualité des données semble perfectible [1]. Le sur-échantillonnage de classes sous-représentées (classes d'attaques) à l'aide de l'algorithme Adasyn [13] et le sous-échantillonnage de logs benign plus avancé que la simple suppression aléatoire telle que réalisée aujourd'hui devraient aussi pouvoir contribuer à l'amélioration globale des différents algorithmes.
- Enfin, l'intégration d'informations métiers modélisées par des graphes de connaissances est une piste de travail intéressante pour contextualiser les anomalies trouvées et aider les analystes à qualifier plus rapidement un début de comportement anormal sur le périmètre surveillé.

7 Remerciements

Nous tenons à remercier chaleureusement nos deux principaux sponsors Philippe Cubizol et Adam Ouorou ainsi que les personnes suivantes pour leur soutien (ordre alphabétique) : Jean-Marc Blanco, Eric Bouvet, Philippe Calvet, Jean-François Calvez, Baptiste Chevreau, Olga Dergachyova, Eric Dupuis, Matthieu Jouzel, Patrick Lomenech, Sok-Yen Loui, Nicolas Noël, Baptiste Olivier, Tony Quertier, Olivier Rodier et Sébastien Roncin.

8 Références

1. Iman Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization", 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, January 2018 - https://www.unb.ca/cic/datasets/ids-2017.html, dernière consultation 4 oct. 2019
2. https://www.unb.ca/cic/datasets/ids-2018.html, dernière consultation 7 oct. 2019
3. http://www.adaltas.com/fr/2018/05/29/apache-metron-dans-le-monde-reel/, dernière consultation 4 oct. 2019
4. https://metron.apache.org/, dernière consultation 4 oct. 2019
5. https://metron.apache.org/current-book/metron-analytics/metron-maas-service/index.html, dernière consultation 4 oct. 2019
6. https://fr.wikipedia.org/wiki/Devops, dernière consultation 4 oct. 2019
7. http://dev.splunk.com/view/dev-guide/SP-CAAAE29 - Architecture Application Splunk, dernière consultation 4 oct. 2019
8. https://mleap-docs.combust.ml/ - Solution de machine-learning standalone, dernière consultation, dernière consultation 4 oct. 2019
9. https://en.wikipedia.org/wiki/Ensemble_learning, dernière consultation 3 oct. 2019

10. Attention Branch Network: Learning of Attention Mechanism for Visual Explanation - https://arxiv.org/pdf/1812.10025.pdf, dernière consultation 3 oct. 2019
11. Breiman, L. Machine Learning (2001) 45: 5. https://doi.org/10.1023/A:1010933404324, dernière consultation 4 oct. 2019
12. Isolation Forest, Fei Tony Liu, Kai Ming Ting, Zhi-Hua Zhou, https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/icdm08b.pdf, dernière consultation 4 oct. 2019,
13. ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning, Haibo He, Yang Bai, Edwardo A. Garcia, and Shutao Li, https://sci2s.ugr.es/keel/pdf/algorithm/congreso/2008-He-ieee.pdf, dernière consultation 4 oct. 2019
14. University of California, I. U. (2007). Kdd cup 1999.
15. https://www.fullstackpython.com/flask.html, dernière consultation 7 octobre 2019
16. https://nifi.apache.org/, dernière consultation 7 octobre 2019
17. https://github.com/titicaca/spark-iforest, dernière consultation, 7 octobre 2019
18. Ho, Tin Kam (1995). Random Decision Forests (PDF). Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282.
19. https://keras.io, dernière consultation 8 octobre 2019
20. David Gunning, DARPA, Explainable Artificial Intelligence (XAI) Program Update, novembre 2017, slide 9, https://www.darpa.mil/attachments/XAIProgramUpdate.pdf, dernière consultation 9 octobre 2019.
21. https://en.wikipedia.org/wiki/Differential_privacy, dernière consultation le 11 octobre 2019