Détection d'intrusion par approche statistique sur un réseau SDN

Contexte

Un système de détection d'intrusion permet aux administrateurs systèmes de détecter les brèches d'un réseau. Il répond aux problématiques de sécurité auxquelles les entreprises et les administrations font face, notamment dans le cas de réseaux étendus et sensibles.
A ce jour, beaucoup de recherches ont été réalisés concernant des NIDS (Network Intrusion Detection System) en particulier dans le cas de réseaux SDN (Software-Defined Networking). La centralisation logique des contrôleurs d'un SDN est atout certain pour la mise en place d'algorithmes de gestion et d'analyse de trafic. C'est pourquoi les recherches liées à la détection d'intrusion s'effectuent majoritairement sur ce type de réseau. Les recherches ont été conduites dans des conditions spécifiques (systèmes, algorithmes, datasets) en utilisant les outils d'apprentissages statistiques (deep learning) permettant de produire des algorithmes théoriques d'implémentation d'un tel système de détection d'intrusion.
Cependant, les réseaux SDN sont aussi la cible de plusieurs attaques, notamment les attaques DDos (Distributed Denial of Services) qui utilisent les vulnérabilités des plateformes SDN.

Description du projet

Le projet à pour objectif d'implémenter un système de détection complet basé sur des méthodes de Deep Learning et à partir des résultats de recherches issues de différents articles scientifiques parus en 2016 et 2017.
Le projet se focalise sur la mise en application des résultats de recherches dans un système capable de fonctionner en temps réel sur un réseau SDN. La partie Deep Learning du système sera mis en place en suivant les indications des différents articles de recherches et en utilisant des Framework permettant d'implémenter rapidement ces technologies (TensorFlow ou Caffe2 par exemple).
Le projet se focalise sur la gestion des différentes entrées du système de détection et sur la mise en place d'algorithmes de traitement de flux de données permettant d'adapter le modèle de détection en continue. Ce système sera codé en Python.
La deuxième partie du projet consiste à la mise en place d'une interface web permettant d'interagir avec le système de détection, d'afficher les intrusions et de signaler les faux positif.

Pour finir, il est nécessaire d'obtenir un ensemble de données tests montrant l'activité d'un réseau lors de différentes attaques afin de pouvoir réaliser la phase d'apprentissage initiale du système. Il existe actuellement diverses bases de données en ligne permettant de récupérer ces informations, mais elles sont pour la plupart ancienne. Avec l'aide de M.Blanc,

nous espérons obtenir une base de données plus récente de la part de la communauté scientifique.

Livrables attendus

- Prototype d'IDS (Intrusion Detection System) avec une interface graphique permettant d'afficher les intrusions en temps réel
- Compte-rendu
- Poster

Contact

Tuteur principal
- Gregory Blanc (gregory.blanc@telecom-sudparis.eu)

Groupe Cassiopée
- Charles Mure (charles.mure@telecom-sudparis.eu)
- Félix Molina (felix.molina@telecom-sudparis.eu)

Analyse Benchmark

Notre projet consiste à l'intégration d'un IDS basé sur un algorithme de Deep Learning sur un réseau SDN. La mise en place de cet IDS peut se découper en 3 points:

1. Le choix d'un contrôleur SDN
2. L'extraction en temps réels des données réseaux
3. L'apprentissage et l'utilisation en temps réel du modèle de Deep Learning pour classifier les paquets réseau.

Nous avons donc réalisé dans un premier temps un benchmark chaque point.

Le choix d'un contrôleur SDN

Il existe actuellement de nombreux contrôleurs SDN OpenSource basés sur le protocole OpenFlow. A l'heure actuelle, les projets les plus prometteurs sont ONOS, OpenDayLight, Ryu.
ONOS et OpenDayLight sont développés en Java, Ryu est développé en Python.
Les contrôleurs ONOS et OpenDayLight sont très complets mais aussi beaucoup plus lourd et plus complexes à mettre en place dans un environnement de test que le contrôleur Ryu.
Le choix du contrôleur a un impact si nous souhaitons par la suite implémenter dans notre système des méthodes de réponses aux attaques.
Nous choisirons donc le contrôleur Ryu car il est plus simple à déployer et il existe déjà plusieurs pour des IDS existant, ce qui pourra faciliter la mise en place de notre projet.

L'extraction en temps réels des données réseaux

Il existe 2 possibilités pour pouvoir récupérer les informations réseaux nécessaires à notre analyse de trames:

- Récupérer directement à partir du Contrôleur SDN les informations réseaux en se connectant à son API.
- Récupérer les informations en utilisant un IDS existant (Bro ou Snort) installé sur le même hôte que le contrôleur.

	Avantages	Inconvénients
API contrôleur	- Flexible - Utilisation de features propre au protocole OpenFlow - Plus "léger" qu'un IDS complet	- Assez peu de features - Impossible d'effectuer une analyse approfondie des paquets - Nécessiter d'adapter à chaque contrôleur

		- Il n'existe pas de jeux de données pour notre phase d'apprentissage.
"Flow Meter" + "Flow Collector" et utilisation des protocoles **NetFlow** et **IP Flow Information eXport** (explications)	- Très flexible par rapport à nos besoins - Solution performance (actuellement déployée en entreprise) - Nombreuses features accessibles	- Complexe à mettre en place - Nécessité de développer notre propre architecture - Nécessité d'adapter les jeux de données existant.
Bro IDS	- Complet et approuvé par la communauté OpenSource. - features calculées accessibles directement dans des fichiers .logs - Il existe des jeux de données d'attaques générées avec cet IDS - Facile à mettre en place	- Lourd (l'IDS comprend de nombreux outils) - demande des ressources de calculs supplémentaires - Peu flexible dans le choix des features
Snort IDS	- Complet et approuvé par la communauté OpenSource. - features calculés accessibles. - il existe des PoC de connecteurs pour les contrôleurs OpenFlow. - Facile à mettre en place	- Lourd (l'IDS comprend de nombreux outils) - demande des ressources de calculs supplémentaires - Pas de jeux de données existants - Peu flexible dans le choix des features

Actuellement, les informations de contrôle de flow et de paquets dans le protocole OpenFlow sont encore limitées. L'intégration d'un IDS directement à partir des données fournies par le contrôleur SDN ne semble donc pas être une solution viable à l'heure actuelle. L'utilisation d'un flow meter et d'un flow collector ainsi que des protocole prévus pour surveiller les flux d'un réseau semble être la solution la mieux adapté à nos besoins. Cependant, cette solution nécessite des développement supplémentaires pour correctement choisir les features que nous souhaitons utiliser. Nous devons donc approfondir nos recherche sur ce sujet pour connaître la faisabilité ou non de cette solution dans le cadre de notre projet (temps de développement nécessaire).

L'utilisation d'IDS existant comme Snort ou Bro nous permettrait de réduire le temps de développement nécessaire à la mise en place d'une solution d'extraction de features. Cette solution est moins flexible et n'est pas envisageable dans un outil complet permettant de surveiller le réseau. Cependant, dans le cadre de notre projet, nous nous intéressons uniquement à la faisabilité d'un IDS basé sur le DeepLearning et non la mise en place d'un outil capable d'être déployé dans une entreprise.

Il serait donc préférable d'utiliser la solution basé sur NetFlow et IP Flow Information eXport mais si celle-ci s'avère trop complexe à mettre en place, nous pourrons utiliser des IDS existant sans que cela perturbe la réalisation de notre projet.

4. **L'apprentissage et l'utilisation en temps réel du modèle de Deep Learning pour classifier les paquets réseau**.

Les papiers de recherches concernant l'utilisation des modèles de Deep Learning ou de réseaux de neurones pour catégoriser les attaques sur un réseau sont apparus récemment (2016-2017). En effet, les articles de recherches précédent mentionnaient l'utilisation de classifier plus simple (SVM par exemple) ou bien de système de détection d'anomalies qui permettent de découvrir des attaques encore non répertoriées.

Les systèmes de détection d'anomalies sont actuellement déployés sur certains IDS mais l'inconvénient est qu'ils n'apportent aucune information sur l'attaque potentielle (trafic normal ou anormal uniquement). C'est ce gap que cherche à combler les modèles de Deep Learning permettant de catégoriser le trafic.

Pour notre projet, nous avons sélectionnée 3 modèles à partir de divers papiers de recherches.

	Avantages	Inconvénients
Multi Layer Perceptron (MLP)	- Modèle simple - Apprentissage rapide	- Ne permet pas d'extraire des corrélations complexes entre les variables - Modèle ancien et avec une faible précision
Convolutional Neural Network + SVM	- Capacité à extraire de nouvelles features - Surpasse les modèles MLP dans de nombreux domaines	- Apprentissage long - Modèle complexe à mettre en place
CapsNet	- L'état de l'art dans la reconnaissance d'image - modèle récent (Novembre 2017) - Précision extrême	- Apprentissage très long - Modèle très complexe à mettre en place

Dans le domaine de la sécurité réseau, les 2 premiers modèles ont déjà fait l'objet d'articles de recherche (publication en janvier 2018 pour le 2ème modèle). Dans notre projet cassiopée, nous allons essayer de reproduire les résultats annoncés par dans les différents article.

Cependant, le benchmark des modèles est réalisé sur un jeu de données appelé NSL-KDD qui est efficace pour évaluer les performances du modèles mais qui est inutile si nous

souhaitons mettre en place un IDS complet. En effet, le NSL-KDD contient des attaques datant des années 90 et 2000. Par conséquent, si nous réalisons un apprentissage de notre réseau avec ce jeu de données, les performances de classification dans un véritable réseau seront médiocre.

Il existe néanmoins quelques datasets qui cherche à combler ce gap. Nous en avons identifiés 2 qui seront utile pour notre projet:

- UNSW-NB15 (2015)
- CICIDS2017

Concernant la mise ne place d'un système d'apprentissage en temps réel à partir de données réseau, il existe peu de travaux disponibles. Nous avons néanmoins trouvé un groupe de chercheur avant mis en place un système d'analyse en continue des données fournies par l'IDS Bro en utilisant Spark et Apache Kafka (leur code est disponible ici). Nous nous inspirerons de leur solution pour la réalisation de notre projet.

Benchmark des IDS utilisant le Deep Learning

Il s'agit ici du véritable gap que cherche à combler notre projet. Il n'existe à l'heure actuelle aucun IDS non commercial faisant appel au Deep Learning. En effet, les seuls sur le marché à proposer actuellement ce genre de service sont **IBM** et **Cisco**. Gemalto, leader mondial de la cybersécurité, est encore en phase de recherche sur ce domaine. Alphabet arrive lui aussi dans ce domaine et cherche à combler le gap avec une nouvelle société appelée Chronicle.

Néanmoins, nous avons pu voir à travers notre benchmark qu'il existe actuellement tous les outils permettant de mettre en place un tel IDS mais aucun article de recherche mentionnant sa mise en place n'existe. Tous les articles que nous avons trouvé indiquent que ce genre de système est faisable mais aucun ne l'implente réellement: cette tâche est généralement définie dans la partie "future work" de l'article.

Macro Planning

Cassiopée - NIDS 14 mars 2018

Tâches 2

Nom	Date de début	Date de fin
Analyse des papiers de recherche	29/01/18	06/02/18
Acquisition des jeux de données (datasets)	07/02/18	12/02/18
Mise en correlation des attributs provenant du SDN et ceux provenant du datasets	13/02/18	21/02/18
Choix des attributs utiles pour une implémentation SDN	13/02/18	21/02/18
Benchmark des différents modèles de réseaux de neurones	07/02/18	26/02/18
Implémentation de modèles	19/02/18	13/03/18
Implémentation Multi-Layer Perceptron	19/02/18	08/03/18
Implémentation Convolutional Model	19/02/18	13/03/18
Implémentation du Random Forest (Classifier)	05/03/18	13/03/18
Implémentation du SVM (Classifier)	05/03/18	13/03/18
Choix du modèle le plus performant	14/03/18	14/03/18
Test du modèle avec de nouveaux datasets	15/03/18	26/03/18
Correction de l'implémentation pour une "mise en production" sans apprentissage en continu	27/03/18	02/04/18
Développement des outils de collecte de données	03/04/18	09/04/18
Tests sans apprentissage en continu	03/04/18	15/05/18
Mise en place des outils de collecte des données	10/04/18	01/05/18
Liaison du programme et du controlleur SDN	03/04/18	01/05/18
Tests en condition réelle du programme	02/05/18	15/05/18
Amélioration du traitement des données acquises	16/05/18	22/05/18
Interface Web	16/05/18	29/05/18
Correction et finalisation pour une mise en production complète	16/05/18	04/06/18
Réalisation livrables	05/06/18	15/06/18

GANTT project
2018
février
mars
avril
mai
juin
juillet
août
Nom
Date de d...
Date d...
Analyse des papiers de recherche 29/01/18 06/02/18
Acquisition des jeux de données (datasets) 07/02/18 12/02/18
Mise en corrélation des attributs provenant du SDN et ceux pro... 13/02/18 21/02/18
Choix des attributs utiles pour une implémentation SDN 13/02/18 21/02/18
Benchmark des différents modèles de réseaux de neurones 07/02/18 26/02/18
Implémentation de modèles 19/02/18 13/03/18
Implémentation Multi-Layer Perceptron 19/02/18 08/03/18
Implémentation Convolutional Model 19/02/18 13/03/18
Implémentation du Random Forest (Classifier) 05/03/18 13/03/18
Implémentation du SVM (Classifier) 05/03/18 13/03/18
Choix du modèle le plus performant 14/03/18 14/03/18
Test du modèle avec de nouveaux datasets 15/03/18 26/03/18
Correction de l'implémentation pour une "mise en production" s... 27/03/18 02/04/18
Développement des outils de collecte de données 03/04/18 09/04/18
Tests sans apprentissage en continu 03/04/18 15/05/18
Mise en place des outils de collecte des données 10/04/18 01/05/18
Liaison du programme et du controlleur SDN 03/04/18 01/05/18
Tests en condition réelle du programme 02/05/18 15/05/18
Amélioration du traitement des données acquises 16/05/18 22/05/18
Interface Web 16/05/18 29/05/18
Correction et finalisation pour une mise en production complète 16/05/18 04/06/18
Réalisation livrables 05/06/18 15/06/18

Compte-Rendu de l'avancement du projet

1 - Analyse et extraction de données d'articles scientifiques concernant la détection d'intrusion

Analyse des articles :
(A1) **A Deep Learning Approch for NIDS**
By Quamar Niyaz, Weiqing Sun, Ahmad Y Javaid, and Mansoor Alam, College Of Engineering, The University of Toledo
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$15.00

(A2) **Deep learning approach for Network Intrusion Detection in Software Defined Networking**
By Tuan A Tang ∗ , Lotfi Mhamdi ∗ , Des McLernon ∗ , Syed Ali Raza Zaidi ∗ and Mounir Ghogho∗†, ∗ School of Electronic and Electrical Engineering, The University of Leeds, Leeds, UK., †International University of Rabat, Morocco
978-1-5090-3837-4/16/$31.00 © 2016 IEEE

Le but est d'obtenir un NIDS à apprentissage supervisé (A2) ou non supervisé (A1), flexible et efficace (taux de détection élevé et faible taux de faux positif).

Attaques testées : DOS, Probe, U2R, R2L.

DataSet utilisés : NSL-KDD dataset – KDD Cup' 99. Les datasets sont non labellisés dans le cas de A1.

Nombres de paramètres (features) :
Pour A1 : 121 avant classification ;
Pour A2 : 6 spécifiques (duration, protocole_type, src_bytes, dst_bytes, count, srv_count)

Différents types d'apprentissage : ANN (le plus répandu), SVM, NB, RF (Random Forest), SOM, OPF, CAPSNET.

Aglos de détection : STL (Self-Taught Learning utilisant le SMR) et SMR (Soft Max Regression).

Algos de classification : J48 Decision Tree Classifier, DMNB, SMR.

Implémentation :
Pour A1 (dans le cas d'apprentissage STL):
1 – Discrétisation des valeurs nominales des attributs (1-to-n encoding)

2 – Elimination de l'attribut ayant pour valeur 0
3 – Normalisation des valeurs entre 0 et 1 (max-min normalization)
4 – Sparse Auto-Encoder
5 – Soft Max Regression

Résultats :

Pour A1 : taux de détection → 98,84 % (avec STL) et 96,79 % (avec SMR)
Pour A2 : environ 70 % de taux de détection

2 – Sélection des premiers attributs à utiliser et premières recherches sur les équipements utilisant OF (OpenFlow)

→ IP source, IP destination
→ Port source, port destination
→ Protocole (type protocole)
→ Date, heure
→ Durée du flux
→ Débit

Ces attributs sont présents dans les datasets NSL-KDD. Mise en corrélation avec les attributs que l'on pourra récupérer sur un switch SDN.

Monsieur Blanc nous a fourni les documentations des switchs SDN utilisant le protocole OpenFlow : Alcatel-Lucent, Dell et IBM.
Après l'étude rapide de ces documentations, il va nous être difficile de récupérer certaines valeurs comme les moyennes. Nous avons donc décidé de ne pas les prendre en compte dans les prochains datasets.

Installation de logiciel de simulation de réseau SDN :
→ mininet (http://mininet.org/)
→ OFnet (http://www.sdninsights.org/)

Pour les contrôleurs SDN :
→ Ryu (https://osrg.github.io/ryu/)
→ OpenDayLight (https://www.opendaylight.org/)
→ ONOS (https://onosproject.org/)

3 – Création des premiers modèles de réseau de neurones

Dans un premier temps, nous avons créé quelques fonctions de preprocessing nous permettant, entres-autres, de binariser et normaliser les valeurs des attributs des datasets. La binarisation consiste, dans notre cas, à transformer un texte en valeur décimale. La normalisation consiste à encoder la valeur initiale entre 0 et 1.

Le dataset que nous utilisons (KDD) contient 41 attributs et 5 classes (4 classes représentant les types d'attaques et une classe dite « normal »).

Puis, **nous avons tenté trois types de réseaux de neurones différents :**

1. Un **réseau de « base » (Multi Layer Perceptron)** que nous avons conçu avec les exemples donnés dans la dcumentation de Keras (module Python utilisé). Il contient 7 layers suivant ce modèle :

Layer	Input	L1	L2	L3	L4	L5	L6	Output
Nb entrée	41	82	164	164	82	41	20	6

2. Un **modèle évolué basé sur la convolution mathématique**. Il provient d'un article de recherche: A Few-shot Deep Learning Approach for Improved Intrusion Detection
By Md Moin Uddin Chowdhury*, Frederick Hammond†, Glenn Konowicz‡, Jiang Li§, Chunsheng Xin and Hongyi Wuk, Department of Electrical and Computer Engineering, Old Dominion University – Conference Paper October 2017 - DOI: 10.1109/UEMCON.2017.8249084)

TABLE I
CNN ARCHITECTURE FOR FEATURE EXTRACTION

Layer Index	Layer	Output Shape	Padding & Stride
1	Conv2D (64 filters, size: 1x3)	1, 38, 64	0,1
2	Conv2D (64 filters, size: 1x3)	1, 36, 64	0,1
3	Maxpooling2D	1, 18, 64	
4	Conv2D (128 filters, size: 1x3)	1, 16, 128	0,1
5	Conv2D (128 filters, size: 1x3)	1, 16, 128	1,1
6	Conv2D (128 filters, size: 1x3)	1, 16, 128	1,1
7	Maxpooling2D	1, 8, 128	
8	Conv2D (256 filters, size: 1x3)	1, 8, 256	1,1
9	Conv2D (256 filters, size: 1x3)	1, 8, 256	1,1
10	Conv2D (256 filters, size: 1x3)	1, 8, 256	1,1
11	Maxpooling2D	1, 4, 256	
12	Flatten	1024	
13	Fully Connected	100	
14	Dropout	100	
15	Fully Connected	20	
16	Softmax	5	

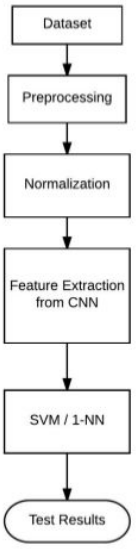

Fig. 1. Flowchart of our considered method.

3. **Un modèle CapsNet**, récupéré sur Github (https://github.com/XifengGuo/CapsNet-Keras).
A Keras implementation of CapsNet in the paper:
Sara Sabour, Nicholas Frosst, Geoffrey E Hinton. Dynamic Routing Between Capsules. NIPS 2017 (arXiv:1710.09829v2)
(Non fonctionnel actuellement)

4 – Résultats obtenus et résultats attendus

1. Pour le premier modèle, le modèle de base, aucun résultat n'était attendu. Ce modèle nous a permis de nous familiariser avec le module Python Keras.

Notre résultat : Model Accurency : ~60 %

2. Pour le second modèle, nous attendions un résultat un peu inférieur à celui obtenu dans le papier de recherche (~95 % avec la fonction de classification) du fait que nous avions pas codé la fonction de classification.

Notre résultat : Model Accurency : ~75 %

3. Ce modèle n'est pas encore fonctionnel sur des données réseaux mais il s'agit de l'état de l'art dans le domaine de la reconnaissance d'image.

Puissance de calcul :
Nos Pcs sont un peu limités pour une quantité de calcul aussi importante (environ un million multiplié par le nombre de d'entraînement (epoch)). Nous avons donc demandé un accès à un serveur de la plateforme de TSP (octa-core cadencé server, 32Go de RAM).

5 – Implémentation des réseaux de classification et modification des réseaux

Implémentation des réseaux de classification. Le but de ce réseau est l'analyse des données afin de les classer (par exemple : « normal » ou « anomaly » pour un réseau à deux classes)

→ SVM (Support Vector Machine) largement utilisé dans les recherches.
→ Random Forest

Ces méthodes de classifications peuvent être utilisé dans des réseaux de neurones bien spécifique et permettent d'améliorer les performances. Ils sont utilisés dans les réseaux dit de "feature extraction" ou "auto-encoder" qui sont généralement des réseaux à convolution.

Le réseau CapsNet était pour finir trop complexe à implémenter. Nous avons donc décidé de le supprimer.

Nous avons aussi compris **pourquoi nous n'obtenions pas les mêmes résultats que ceux des papiers de recherches.**
Dans les papiers de recherches le taux de détection est calculé de manière binaire. Par exemple, une donnée d'apprentissage (pour un réseau de 5 classes) donne le vecteur suivant (après son passage dans le réseau) [0 0 0 0 1].
Lors du test (avec une donnée similaire à celle de l'apprentissage) si le réseau donne un vecteur de sortie [0 0 0 1 0] alors le taux de détection sera de 60 %.

[0 0 0 0 1]
[0 0 0 1 0]
20 % 20 % 20 % 0 % 0 % = 60 %

Dans nos tests réalisés les semaines précédentes, nous calculons l'erreur de manière vectorielle et absolue => **si les deux vecteurs sont différents, le taux de détection = 0 %.** Nous trouvons en effet cette mesure de calcul plus pertinente dans le cadre d'un système de classification et permet d'obtenir des résultats plus "parlant".

Pour améliorer les performances, nous avons aussi suivi les indications du papier de recherche A Few-shot Deep Learning Approach for Improved Intrusion Detection qui propose de générer des nouveaux exemples à partir de ceux existant dans les classes qui sont sous-représentées (méthode dite de sur-échantionnellement). Cela n'a pas un impact important sur la précision globale, mais la précision dans les classes minoritaires est fortement améliorée.

Dans notre implémentation finale, nous devrons donc faire attention à avoir une représentation homogène de chaque classe dans nos jeux de données d'apprentissage.

6 – Tests des réseaux optimisés

Nous avons testé 4 types de réseaux avec calcul vectoriel du taux de détection :
→ Le réseau Multi Layer Perceptron **seul (75%)**
→ Le réseau convolutionnel **seul (77%)**
→ Le réseau Multi Layer Perceptron **avec la classification SVM (76%)**
→ Le réseau convolutionnel seul **avec la classification SVM (77%)**

Puis, nous avons testé avec calcul binaire du taux de détection :
→ Le réseau convolutionnel **seul (91%)**

La valeur obtenue n'est pas très juste car notre implémentation n'est pas encore complète. Mais on observe une nette amélioration.