Analyse statistique de filtrages réseaux et apprentissage automatique

Table des matières

[Introduction 1](#_Toc163807767)

[1. DES REGLES DE FILTRAGE AUX DONNÉES D'APPRENTISSAGE 2](#_Toc163807768)

[1.1 Extraction des règles de filtrage 2](#_Toc163807769)

[1.2 Représentation des plans d'adressage 3](#_Toc163807770)

[1.3 Représentation des ports/services 3](#_Toc163807771)

[1.4 Quelques tours de tournevis en plus 4](#_Toc163807772)

[2. APPRENTISSAGE NON SUPERVISÉ 5](#_Toc163807773)

[2.1 CAH et analyse des regroupements 6](#_Toc163807774)

[2.2 Exploitation des résultats 7](#_Toc163807775)

[3. CONSTRUCTION D'UN MODÈLE DE CLASSIFICATION SUPERVISÉE 8](#_Toc163807776)

[3.1 Principe 8](#_Toc163807777)

[3.2 Pycaret 9](#_Toc163807778)

[3.3 Industrialisation 9](#_Toc163807779)

[4. INTROSPECTION 10](#_Toc163807780)

[CONCLUSION 11](#_Toc163807781)

[RÉFÉRENCES 11](#_Toc163807782)

# Introduction

La tendance DevOps est arrivé chez vous et la maîtrise de certains de vos filtrages réseaux passe à la main des projets ? Vous voulez évaluer le niveau de sécurité qui en résulte ? Ou plus globalement vous faire une idée de l'état de conformité de vos filtrages ? Récit d'une approche statistique et par apprentissage.

Quand un audit de sécurité informatique aborde le périmètre du réseau, inévitablement un audit des filtrages en place est effectué. L'objectif est d'identifier les non-conformités à la politique de sécurité du périmètre concerné et d'évaluer les éventuelles configurations à risques de sécurité.

Concrètement, il existe deux approches, souvent complémentaires, pour effectuer une revue des filtrages de vos pare-feux. La première est l'analyse manuelle par un expert: trouvez un collaborateur motivé pour analyser quelques milliers de lignes de règles puis laissez-le travailler quelques jours ou mois suivant le périmètre.

La deuxième approche est une analyse à base de règles strictes, dérivées d'une politique de sécurité: par exemple, « les flux en provenance d'Internet sur le port 22 sont interdits ». Si ce genre d'approche est facilement automatisable, celle-ci n'est malheureusement sement pas parfaite, car elle se confronte au traitement des nombreuses exceptions qui peuplent le monde réel des systèmes d'information (SI).

Cet article vous propose une troisième approche, basée sur des outils actuellement en vogue d'apprentissage non supervisé et supervisé. Elle permet de réaliser ces revues de conformité et plus précisément d'identifier des configurations exposées à un risque de sécurité. L'approche a été expérimentée dans un contexte réel sur un périmètre d'environ deux mille machines.

Les sections suivantes décrivent les étapes de mise en œuvre de cette expérimentation: la collecte et la préparation des données pour une exploration pertinente en mode « non supervisé », suivies de quelques pistes d'industrialisation des algorithmes de détection basés sur nos découvertes et d'une approche critique de cette démarche.

Les illustrations de code sont données à titre d'exemple, en langage Python, à l'aide essentiellement des bibliothèques de reférence pandas [pandasl et

scikit-learn [scikit-learn]. Ces travaux étant fortement liés au contexte de chaque entreprise, un mode opératoire complet ne peut pas être fourni, seuls les points clés de la réflexion doivent être pris en compte.

À sa décharge, l'auteur de cet article ne possède qu'un bagage sommaire du domaine des statistiques et de l'apprentissage automatique et s'excuse par avance auprès des experts pour les approximations.

# 1. DES REGLES DE FILTRAGE AUX DONNÉES D'APPRENTISSAGE

La préparation de la donnée, « feature engineering» en anglais, est une étape clé pour toute approche d'apprentissage ou même de simple analyse statistique. Elle consiste à transformer nos données brutes, ici les règles de filtrage, en une représentation pertinente pour un algorithme d'apprentissage afin que celui-ci puisse en extraire de la connaissance.

## 1.1 Extraction des règles de filtrage

L'extraction des règles de filtrage est la première étape de notre ex-périmentation. Il est important de noter qu'il n'existe pas de méthode universelle d'extraction des règles de filtrage : chaque méthode est propre aux technologies de filtrage sous-jacentes.

Notre exemple porte sur une infrastructure OpenStack [openstack].

Cette technologie facilite l'extraction des règles, car elle peut s'effectuer de manière centralisée via l'utilisation de l'API, ou du client OpenStack [oscl, à travers des objets security group et firewall (groups,policies, rules).

Voici, à titre d'exemple, quelques commandes qui nous ont permis de sortir les configurations des règles

Img 1

Notre étude s'intéresse aux filtrages paramétrés par les équipes DevOps disposant d'un accès à l'infrastructure OpenStack. Cela correspond aux règles des flux entre cette infrastructure et le reste du SI, avec une granularité au niveau des machines virtuelles. Notre infrastructure OpenStack offre deux mécanismes de filtrage : les « security group » positionnées sur chaque machine, et un pare-feu au niveau de chaque « tenant ». Les règles du pare-feu s'appliquent sur toutes les machines du « tenant ». Il a fallu interpréter la configuration de ces deux couches pour en déduire les règles effectives entre chaque machine virtuelle et le reste du SI.

Le résultat a été mis sous forme de tableau avec, par flux autorisé, les colonnes suivantes : [serveur, réseau distant, port destination, direction].

La figure 1 en page suivante illustre un exemple de format de tableau, avec des données fictives, ainsi que les étapes du traitement explicite dans les chapitres suivants.

À première vue, rien de bien sorcier... Et pourtant, ces actions peuvent devenir chronophages, voire complexes, si votre architecture empile les couches de pare-feux.

Bon courage, révisez bien vos commandes pandas pour vous faciliter la vie, et jetez un petit coup d'œil à la bibliothèque python ipaddress pour gérer les appartenances des IP aux sous-réseaux.

Fig1

## 1.2 Représentation des plans d'adressage

Les données initiales étant maintenant disponibles, l'étape suivante consiste à les transformer en un vecteur de nombres (le « feature vector »). Ce format de vecteur est le seul gérable par nos algorithmes d'apprentissage. Il est aussi primordial que ce vecteur ait du sens et puisse retranscrire au mieux nos données de filtrage en mettant en avant des informations significatives pour la sécurité.

Une attention particulière est à porter sur la transformation du réseau distant. Certains articles disponibles sur Internet traduisent directement les adresses IP en entier à partir des nombres qui les constituent (par exemple, 192.168.1.1 = 192\*256^3 + 168\*256^2 + 1\*256 + 1).

Ensuite, nous ajoutons une colonne nb\_ ip afin de garder l'information du nombre d'adresses autorisées dans le filtrage sur la zone concernée. Nous reviendrons par la suite sur son traitement.

## 1.3 Représentation des ports/services

Il s'agit maintenant de s'atteler à la représentation des ports et des services dans notre feature space. Le point d'attention exprimé précédemment persiste dans cette nouvelle étape.

Si le numéro de port est directement converti en nombre, nos outils vont interpréter que deux numéros de ports proches ont la même signification.

Par exemple, les ports 22 (service SSH) et 25 (ser-vice SMTP) sont plus proches numériquement que le port 3389 (service RDP) alors que d'un point de vue sécurité, on aura tendance à analyser de manière similaire le 22 et 3389 en les considérant comme des flux d'administration, bien distincts des services de messagerie SMTP sur le port 25.

Là aussi, la parade consiste à catégoriser la donnée en regroupant les ports par type de service ayant un sens d'un point de vue de la sécurité. Dans notre étude, nous avons retenu les catégories suivantes : administration (SSH, RDP), VPN (SOCKS, OpenVPN, etc.), services systèmes (DNS, NTP, RPCBBIND, MICROSOFT-DS, etc.), bases de données (SQL, NFS, etc.) et WEB (HTTP, HTTPS, sur ports standaras ou alternatifs comme 8080, 8443, etc.). Ce regroupement est un parti pris propre à notre contexte, il doit être adapté au vôtre et à la finalité de l'analyse.

Bien entendu, la correspondance port/service réelle est purement spéculative, car il est techniquement possible de faire de l'administration sur le port 443 via des services web. Mais laissons de côté cette problématique qui est générale et sort du cadre de cet article.

## 1.4 Quelques tours de tournevis en plus

Le tableau [serveur, zone distante, service, direction, nb\_ip] est désormais construit. Il se compose de trois colonnes catégorielles et d'une colonne numérique; le serveur étant gardé pour le regroupement final des données.

Nous allons maintenant transformer les catégories en nombre : les différentes valeurs possibles des catégories vont devenir des colonnes et nous allons inscrire la valeur nb\_ipdans la colonne correspondante à notre ligne, le reste restant à 0. Plusieurs fonctions Python existent pour réaliser cette tâche, celle de la librairie pandas fait bien le travail : pandas .get\_dummies.

Une petite remarque sur les valeurs de notre tableau : celles-ci correspondent au logarithme du nombre d'adresses concernées par l'ouverture du filtrage en question sur la ligne.

En effet, si le tableau est laissé tel quel avec les valeurs de nb\_ip, l'écart entre 1 et 50 aura la même « signification » pour nos algorithmes que l'écart entre 20001 et 20050. Ouvrir un flux depuis 20001 ou 20050 adresses n'aggrave pas de manière significative notre risque de sécurité. Par contre, ouvrir un flux depuis une adresse ou depuis 50 adresses augmente de manière significative notre risque de sécurité au niveau de la surface d'attaque.

Pour faire passer cette information au reste du traitement, nous appliquons une fonction logarithmique (par exemple avec numpy. loglp) sur notre jeu de données qui affaiblit ces écarts sur les valeurs hautes. Cette représentation est discutable, mais simple à implémenter et plus pertinente que de laisser la valeur telle quelle.

À noter que ces actions de mise à l'échelle sont d'autant plus pertinentes dans le cas d'un apprentissage non supervisé, car faute de disposer de cible de classification, c'est principalement la distance entre les éléments qui est prise en compte. L'échelle utilisée a donc un impact important.

Pour finir, nous allons regrouper nos valeurs par serveur via une simple agrégation de valeurs.

La figure 1 résume les différentes étapes du traitement des données.

Suivant le nombre de services ou de zones réseaux que vous voulez différencier, le nombre de dimensions (c.-à-d. les colonnes du tableau) peut augmenter rapidement. Il est calculé ainsi : nb\_zone \* nb\_ services \* 2 dimensions. Dans notre cas, nous avons un nombre important de zones sur notre réseau interne, et nous avons obtenu plus de 500 dimensions.

Un nombre de dimensions important peut être pénalisant pour le temps de traitement des algorithmes d'apprentissage, voire perturbant pour certaines familles d'algorithmes, comme par exemple la régression linéaire. Pour réduire ce nombre dedimensions, nous avons supprimé les dimensions qui ont une corrélation parfaite, et donc n'apportent pas d'information supplémentaire dans la discrimination des éléments à classer. Cela arrive notamment parce que certains filtrages larges (typiquement 0.0.0.0/0), activent plusieurs zones réseaux de manière identique.

Internet est votre ami pour trouver des exemples de code pour effectuer cette réduction de dimensions ! Nous avons utilisé celui qui est présenté dans la référence : [corr].

# 2. APPRENTISSAGE NON SUPERVISÉ

Une fois ce travail ingrat, mais essentiel, de préparation de la donnée effectuée, passons à la partie exploration des données.

En l'absence de caractérisation de nos données (filtrage conforme ou non), procédons à un apprentissage non supervisé pour dégager des regroupements et obtenir une meilleure visibilité sur nos données.

Plusieurs techniques existent et nous privilégions la Classification Ascendante Hiérarchique (CAH).

## 2.1 CAH et analyse des regroupements

La Classification Ascendante Hiérarchique (CAH) adopte un principe itératif: à chaque itération, un regroupement de deux individus (ici nos serveurs), ou groupe d'individus, est effectué de proche en proche jusqu'à ne constituer qu'un seul groupe.

Dans le cadre de notre expérimentation, alors que nous n'avons aucune idée de la répartition de nos données et donc du nombre de regroupements que nous pouvons faire, la CAH est tout à fait perti-nente. En effet, son principe itératif nous permettra de visualiser pour chaque itération les groupes qui se constituent, surtout vers les dernières itérations.

Concrètement, cela se passe en deux lignes (ci-dessous, X est le tableau de nos données préparées à la section précédente) :

Img 2

La visualisation du résultat s'effectue généralement au travers d'un dendrogramme (voir [dendrol pour le code permettant de tracer ce genre de graphique).

La figure 2 illustre le dendrogramme de notre CAH.

Ce graphique montre les derniers regroupements qui sont effectués par itération. La hauteur des étages de regroupement représente la distance entre les deux individus, ou groupe d'individus, qui ont été regroupés: plus les groupes d'individus sont distants, plus la hauteur avant regroupement est importante.

À la lecture du dendrogramme (lecture de bas en haut), on note un saut dans la hauteur (de 20 à 50) pour le dernier regroupement et dans une moindre mesure dans le 4e en partant de la fin (du haut).

Nous avons décidé de nous arrêter à ce dernier regroupement (ligne rouge), ce qui représente six groupes (nombre de barres verticales bleues sous la ligne rouge).

Pourquoi avoir choisi six groupes et non pas seulement deux groupes ? Nous avons exploré plus précisément ces regroupements en analysant manuellement les filtrages d'un échantillon de machines pour chaque groupe. À cette fin, on relance une CAH, mais cette fois avec un nombre n\_clusters fixe et nous analysons manuellement les regroupements de machines pour voir s'ils sont pertinents.

Fig2

Fig3

Dans notre cas optimal à six groupes, voici la commande à utiliser. On sort le résultat de nos groupes (c.-a-d. un label de 0 à 5) dans une variable labels :

IMG3

Dans notre contexte, il se trouve que le groupe 1 comportait des serveurs avec un niveau de filtrage exemplaire (flux d'administration limité à quelques adresses uniques, restriction des accès web au strict nécessaire du besoin, pas d'ouvertures sur les ports systèmes ou bases de données) alors que les groupes 4 et 5 comportaient des serveurs avec une ouverture réseau peu raisonnable (quelques 0.0.0.0/0, ports non-standards ouverts très largement). Les groupes 2 et 3 ne présentaient pas de risques particuliers.

Enfin, le groupe 0 comportait la majorité des autres serveurs, avec des configurations plus disparates et dont le volume nous empêchait une analyse rapide.

Pour visualiser sur un graphique ces regroupements malgré les nombreuses dimensions des don-nées, une ACP (analyse en composantes principales) est pratique pour effectuer une projection humainement visible en deux dimensions. Elle consiste à transformer des variables liées entre elles en nouvelles variables décorrélées les unes des autres.

On sélectionne ensuite les nouvelles variables qui déterminent le mieux le jeu de données, ici deux pour effectuer une représentation approximative de nos données sur un plan.

IMG4

La figure 3 montre une telle projection avec nos groupes. On note l'isolation des groupes 4, 5 et 1, par rapport au reste : c'est bon signe, cela signifie que l'on pourra les isoler plus facilement par nos outils d'apprentissage.

Attention tout de même à ce genre de représen-tation, forcément trompeuse quand on passe d'une centaine de dimensions à deux dimensions : le rapprochement de deux groupes ne signifie pas forcément qu'ils sont aussi proches que la représentation graphique le laisse paraître.

À noter que d'autres méthodes de projection en deux dimensions existent pour visualiser des données à forte dimension, notamment t-SNE [tsne].

## 2.2 Exploitation des résultats

Parmi les 143 serveurs analysés, nous avons identifié 33 serveurs appartenant aux groupes 4 et 5 précédemment identifiés comme possédant des ouvertures de filtrage à risque. L'analyse manuelle des filtrages de ces 33 serveurs a permis de confirmer notre regroupement, car nous étions systématiquement en présence d'ouvertures de flux en entrée trop larges sur des ports d'administration ou sur du protocole applicatif spécifique, et notamment anormalement large pour le groupe 5. Les équipes DevOps en charge de ces projets ont été contactées pour corriger les regles de filtrage.

Comme piste d'amélioration pour le futur, nous notons que les groupes avec des filtrages « à risques» ne sont pas aussi isolés que le groupe 1 (bons élèves).

Cela laisse une opportunité pour poursuivre notre analyse avec un découpage plus fin à plus de 6 groupes pour identifier d'autres groupes à risques. Libre à vous d'arrêter ces analyses suivant vos exigences ou vos ressources.

En retour d'expérience, la conception de cette étape d'apprentissage a ete assez rapide: deux jours de travail, y compris les phases de vérification de nos résultats. L'application récurrente de la méthode ne devrait pas prendre plus que deux ou trois heures, car, dans notre cas d'usage, la classification était suffisamment fiable.

# 3. CONSTRUCTION D'UN MODÈLE DE CLASSIFICATION SUPERVISÉE

Récapitulons : nous avons modélisé nos filtrages, nous les avons visualisés, nous avons constitué des groupes de serveurs, et nous avons identifié des machines à risque.

Peut-on aller plus loin ? Automatiser un peu notre approche?

## 3.1 Principe

Ayons à l'esprit que chaque fois que nous recommençons nos regroupements avec des données dif-férentes, les numéros de cluster changent et il faut réanalyser chaque groupe pour évaluer son niveau de conformité et de risque.

Ce n'est pas impactant quand on effectue un audit ponctuel des configurations réseau, mais cela n'est pas adapté, par exemple, à la mise en œuvre d'un processus automatisé qui va réagir dans la journée, voire immédiatement sur alerte, à l'ouverture d'un filtrage inopportun.

Dans ce cadre, il faut compléter notre démarche par une approche d'apprentissage supervisée et constituer notre jeu de données à partir de l'étape précé-dente. Concrètement, on ajoute une colonne label dans notre tableau sur la base des groupes identifiés.

iMG5 :

Comme notre objectif est de prédire une classe à partir des données (filtrage serveur à risque ou non), nous sélectionnerons des algorithmes de classification supervisée.

## 3.2 Pycaret

Il existe de nombreux algorithmes pour faire de la classification supervisée.

Nous allons utiliser ici la bibliothèque pycaret [pycaretl qui est facilement abordable pour les débutants. D'autres bibliothèques existent, bien entendu. Utilisez le mot clé AutoML dans votre moteur de recherche préféré pour avoir un aperçu de l'offre.

Nous sélectionnons la catégorie des algorithmes de classification et lançons notre analyse sur les données labellisées préalablement :

IMG6

Dans notre cas, nous trouvons plusieurs algorithmes avec des scores « parfaits », c'est-à-dire qui arrivent à classer nos données sans erreurs : régression logistique, méthode des k plus proches voisins, forêt aléatoire, AdaBoost, etc. Ce n'est pas très surprenant, car nous avons assigné nos labels à des regroupements issus d'un apprentissage non supervisé et donc par définition bien distincts entre eux. Le résultat aurait été plus délicat si nous avions identifié des filtrages non conformes situés dans le cluster 0 de notre exemple précédent.

Pour sauvegarder votre modèle pour un usage ultérieur, n'oubliez pas d'effectuer une étape de fina-lisation. En effet, la comparaison de modèles s'effectue par validation croisée, donc avec une partie des données laissées de côté pour l'évaluation. La finalisation appliquera l'apprentissage sur l'intégralité des données.

IMG7

## 3.3 Industrialisation

Quelques mots tout de même concernant l'utilisation d'un tel modèle dans la « vraie vie », et les points à avoir en tête et à résoudre :

• Conserver un jeu de données d'apprentissage dans le temps; • Mettre en œuvre une boucle d'enrichissement des données : labellisation par des analystes (i.e faux positifs et faux négatifs), résultats de futures revues de configurations ;

• Mettre à jour régulièrement le modèle en relançant l'apprentissage ;

• Vérifier une éventuelle dérive du modèle et éventuellement changer d'algorithme ou les hyper-paramètres (paramètres utilisés pour contrôler le processus d'apprentissage) ;

• Mettre à disposition le modèle de classification (par exemple via un serveur MLflow [mlflow])

pour les applications clientes ;

• Intégrer les appels au modèle depuis les applications clientes (outils de ticketing, SIEM, etc.).

# 4. INTROSPECTION

Au-delà de l'usage d'outils d'apprentissage automa-tique, pratique à la mode, nous avons quand même pu analyser quelques centaines de machines, plusieurs milliers de règles de filtrage, en quelques heures, si l'on exclut les quelques jours de conception du traitement des données. L'automatisation du processus d'analyse nécessiterait un investissement bien plus important, se chiffrant plus en mois. Il faut aussi disposer de compétences en analyse de données, ce qui n'est pas forcément courant actuellement dans les équipes d'infrastructure.

Avec une approche plus classique, nous n'aurions pas pu humainement analyser chaque machine aussi rapidement, mais nous aurions pu faire quelques recherches sur les ouvertures les plus à risque, par exemple les ouvertures des ports d'administration.

Et franchement, nous n'aurions pas été très éloignés de reconstituer nos deux groupes non conformes, en quelques minutes ...

L'inconvénient de cette approche est qu'il faut se constituer des règles de contrôle « expertes » qui peuvent vite se multiplier et devenir ingérables si on entre dans les détails et exceptions : pas d'ouverture de SSH depuis les adresses Internet, sauf pour telle ou telle plage réseau de partenaires ou sous-traitants, mais que sur tel ou tel projet, sauf l'adresse du scanner de vulnérabilités, etc.

Pour simplifier grossièrement, l'approche par apprentissage supervisé permet, par l'ex-périence, de déduire les règles de contrôle par l'algorithme. Il ne reste plus qu'à l'alimenter par les analyses des auditeurs sur quelques règles ou groupes. La maintenance et l'écriture de règles de contrôle complexes sont simplifiées.

Par contre, l'écosystème devient plus « flou » : pas de visibilité sur les règles de décision de l'algorithme, et risque de mauvaise classification en cas de groupes de classes trop proches ou de choix de mauvais algorithme. Et le flou n'est pas forcément l'ami de la conformité, un bon choc de culture en perspective.

Pour en revenir à notre retour d'expérience, l'apprentissage non supervisé nous a permis d'avoir une vision globale de notre exposition au risque en identifiant les typologies de groupes de machines. La vision est approxima-tive, mais toujours plus lisible qu'un tableau de filtrage de plusieurs milliers de lignes, ou plus pertinente qu'un simple filtre sur les ports SSH ouverts sur Internet. Cette étude nous a permis d'identifier les dérives de configuration et d'y remédier à travers la rédaction d'une politique de filtrages plus adaptée et d'une sensibilisation ciblée auprès des équipes projet concernées. La mise en œuvre, de manière récurrente, d'une catégorisation automatique des filtrages a été écartée à cause de l'investissement trop coûteux, au profit d'alertes quotidiennes sur quelques règles simples de bon sens en mode garde-fou.

# CONCLUSION

Nous avons voulu que ce retour d'expérience puisse apporter des idées pour des applications plus proches de vos contextes respectifs, mais aussi démystifier les gains réels par rapport à l'investissement nécessaire. Comme vous avez pu le remarquer, la démarche n'est pas de promouvoir à tout prix des technologies qui ont le vent en poupe, mais bien de les replacer dans leur contexte d'usage global et d'apprécier leurs spécificités d'usage, leurs qualités, et leurs défauts.

# RÉFÉRENCES

[pandas] pandas, https://pandas.pydata.org/

[scikit-learn] scikit-learn, « Machine learning in python »,

https://scikit-learn.org/

[openstack] OpenStack, « Cloud Infrastructure for Virtual Machines, Bare Metal, and

Containers »,

https://www.openstack.org/

[osc] OpenStackClient,

https://docs.openstack.org/python-

openstackclient/latest/

[dendrol scikit-learn, « Plot Hierarchical Clustering Dendrogram »,

https://scikit-learn.org/stable/auto\_examples/

cluster/plot\_agglomerative\_dendrogram.html

corr Life with data, « How to remove Highly Correlated Features from a dataset »,

https://lifewithdata.com/2022/03/13/how-to-

remove-highly-correlated-features-from-a-

dataset/

[pycaret] Pycaret, « Low-code machine

learning », https://pycaret.org/ [mlflow] MLflow, « An open source platform for the machine learning lifecycle »,

https://mlflow.org/

lipaddress] Python, « IPv4/IPv6 manipulation library »,

https://docs.python.org/3/library/

ipaddress.html

[tsnel scikit-learn, « T-distributed Stochastic

Neighbor Embedding »,

https://scikit-learn.org/stable/modules/

generated/sklearn.manifold.TSNE.html