SWaT Security Report

Machine Learning pour la Cybersécurité

Anthony Bernard, David Frécon, Junyi Li, Louis Pagnier, Sron Léo

SCIA 2024

Résumé

Ce rapport d'analyse et de sécurité montre les étapes que nous avons suivis pour essayer de détecter les attaques du jeu de données SWaT.A3. Ces attaques ont pour but de nuire au système de traitement de l'eau en falsifiant les données des capteurs ou en actionnant certains switchs. Pour tenter de détecter les attaques grâce au machine learning, nous avons commencé par appliquer des méthodes de détection d'anomalies non-supervisés tels que Isolation Forest et LoF. Étant donné que le dataset est labélisé, nous avons essayé d'appliquer des méthodes d'apprentissage supervisé mais nous avons finis par rencontrer plusieurs problèmes importants liés aux données, nous contraignant à rester sur de l'apprentissage non-supervisé.

Table des matières

1 Introduction

Le banc d'essai de recherche Secure Water Treatment (SWaT) a été créé en 2015 par le Centre de recherche en cybersécurité de l'Université de technologie et de design de Singapour (iTrust) dans le but de fournir une plateforme de recherche et de développement pour évaluer la cybersécurité des infrastructures critiques liées au traitement de l'eau. Il se présente comme une reproduction à échelle réduite d'une véritable installation de traitement de l'eau (voir Figure 1).

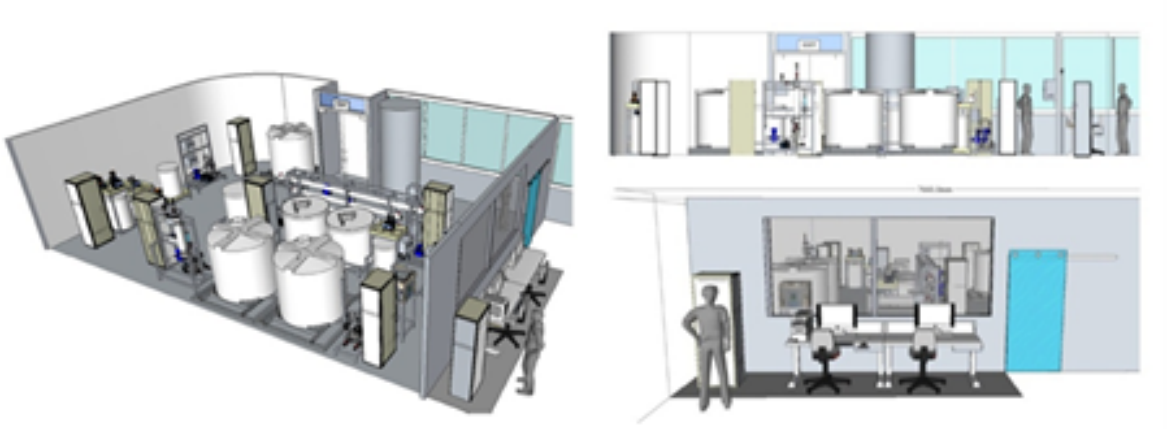


FIGURE 1 – Layout d'installation de SWaT

2 Présentation du jeu de données

Ce banc d'essai a été minutieusement conçu pour simuler une station d'épuration moderne, incluant un ensemble de procédures essentielles au traitement de l'eau (voir Figure 2). Ces procédures sont les suivantes :

— P1 : Prise d'eau brute, représentant le point de départ du processus de traitement, où l'eau non traitée est captée.
— P2 : Désinfection chimique, qui consiste en l'application de produits chimiques spécifiques pour éliminer les micro-organismes nocifs présents dans l'eau.
— P3 : Ultrafiltration, une étape cruciale impliquant l'utilisation de membranes pour filtrer les particules et les contaminants de l'eau.
— P4 : Déchloration à l'aide de lampes à ultraviolets, destinée à éliminer les résidus de produits chimiques utilisés dans l'étape précédente.
— P5 : Purification par osmose inverse, qui permet d'éliminer les impuretés restantes par un processus de filtration avancé.
— P6 : Lavage à contre-courant et nettoyage de la membrane d'ultrafiltration, visant à garantir l'efficacité continue du processus.

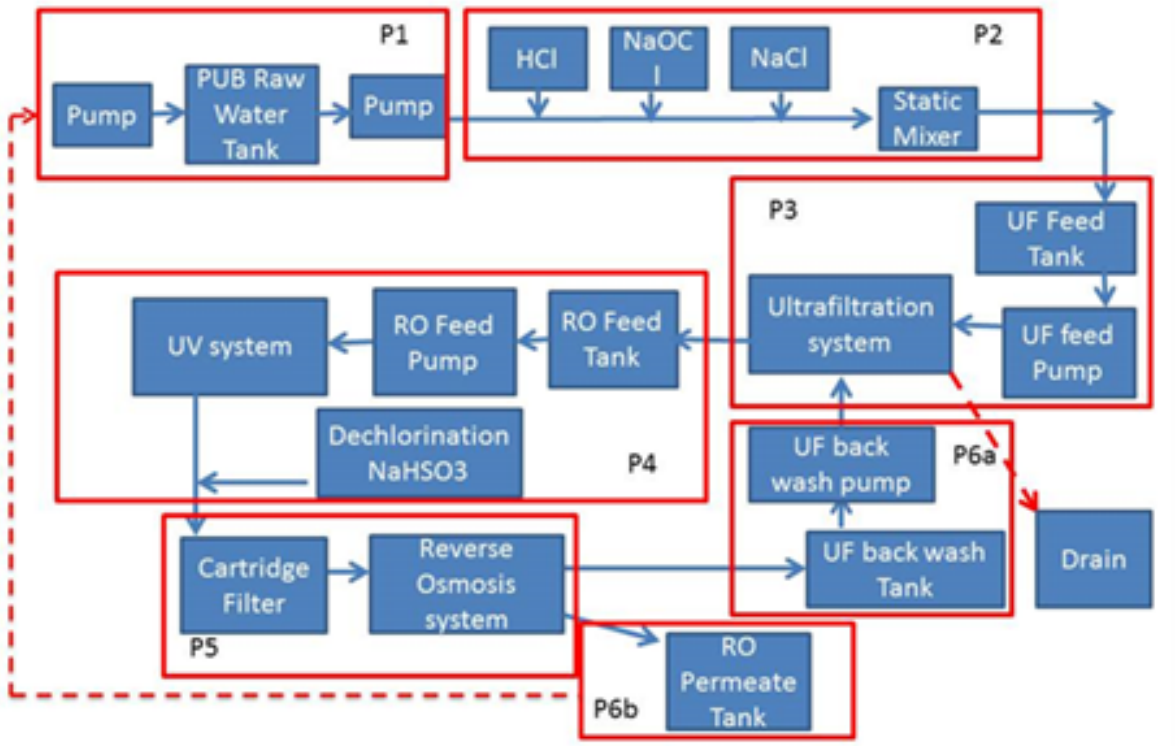


FIGURE 2 – Architecture SWaT

Le banc d'essai de recherche intègre un ensemble sophistiqué d'équipements de purification de l'eau, plusieurs niveaux de réseaux de communication, des automates programmables, un poste de travail SCADA (Supervisory Control and Data Acquisition), des postes de travail IHM et une infrastructure de stockage des données d'exploitation.

La Figure 3 offre une vue détaillée du réseau du banc d'essai. Celui-ci se compose de deux couches distinctes : un réseau PLC basé sur Ethernet (Zone-B) et un environnement SCADA (Zone-C/D). Chaque étape du processus est équipée de deux automates pour assurer une redondance architecturale et garantir un fonctionnement ininterrompu.

Ces automates interagissent en utilisant un protocole d'application nommé Ethernet/IP, qui utilise le protocole de transport TCP pour la communication. Le réseau de la Zone B partage un commutateur avec le réseau de la Zone C/D, qui est spécifiquement dédié au logiciel SCADA et à l'interface homme-machine, formant ainsi un système intégré et cohérent pour la supervision et le contrôle du processus de traitement de l'eau. Pendant son fonctionnement, le laboratoire de recherche iTrust a publié de nombreux jeux de données, qui ont été collectés à partir du banc d'essai. Certains de ces ensembles de données ne contiennent que des relevés bruts des valeurs des capteurs, tandis que d'autres comprennent également des captures de paquets de réseau.
Aujourd'hui, les ensembles de données SWaT figurent parmi les ensembles de données les plus étudiés pour la détection d'anomalies dans le domaine de la sécurité des réseaux.

En août 2019, les chercheurs ont publié l'ensemble de données SWaT A4, un jeu de données SWaT collecté en juillet 2019. Cet ensemble comprend 3 heures de fonc-

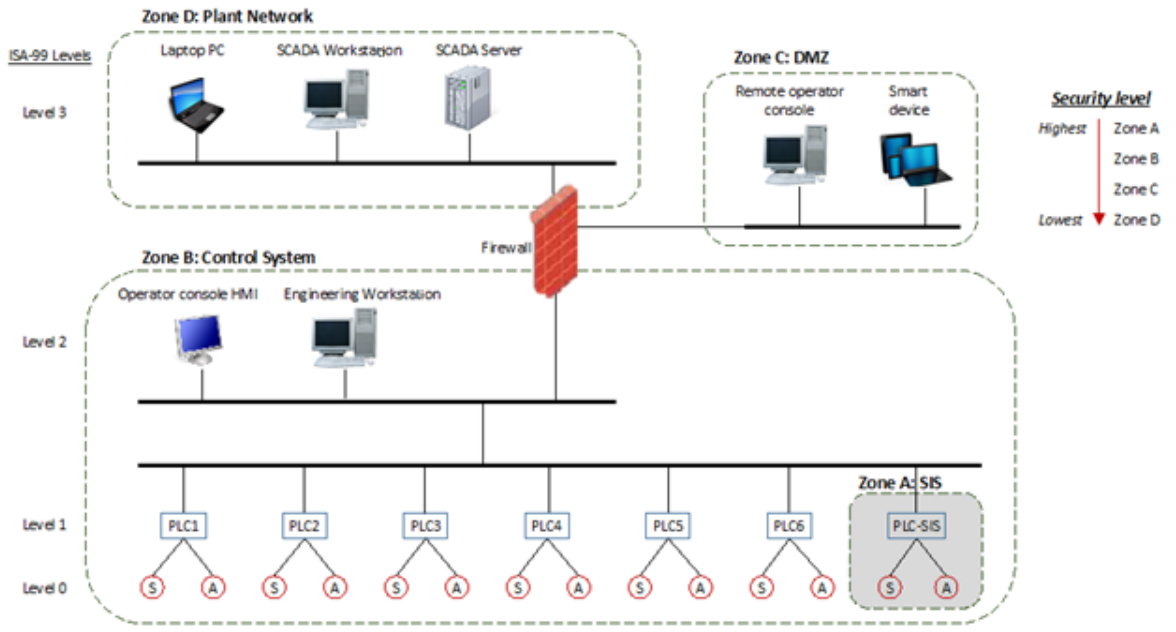


FIGURE 3 – Architecture réseaux de SWaT

tionnement de SWaT dans des conditions normales d'exploitation et 1 heure pendant laquelle 6 attaques ont été menées. En octobre 2019, une version plus compréhensive de ce même dataset à été publiée (SWaT A5), mais nous allons travailler sur le dataset SWaT A4. Nous avons accès à la description des 6 attaques menées :

1. Attaque sur FIT401 : Usurpation de la valeur de 0,8 à 0,5
   Objectif : Arrêter la déchloration en éteignant l'UV401
   Heure de début : 3 :08 :46 PM
   Heure de fin : 3 :10 :31 PM
2. Attaque sur LIT301 : Usurpation de la valeur de 835 à 1024
   Intention : Conduire éventuellement à un sous-débit dans T301
   Heure de début : 3 :15 PM
   Heure de fin : 3 :19 :32 PM
3. Attaque sur P601 : Passage de OFF à ON
   Intention : Augmenter la quantité d'eau dans le réservoir d'eau brute
   Heure de début : 3 :26 :57 PM
   Heure de fin : 3 :30 :48 PM
4. Attaque multi-cible : Passage de FERMÉ à OUVERT (MV201) et de OFF à ON (P101)
   Intention : Faire déborder le réservoir T301
   Heure de début : 3 :38 :50 PM
   Heure de fin : 3 :46 :20 PM
5. Attaque sur MV501 : Passage de OUVERT à FERMÉ
   Intention : Vider l'eau de l'OI
   Heure de début : 3 :54 PM
   Heure de fin : 3 :56 PM

6. Attaque sur P301 : Passage de ON à OFF
   Intention : Arrêter l'étape 3 (processus UF)
   Heure de début : 4 :02 :56 PM
   Heure de fin : 4 :16 :18 PM

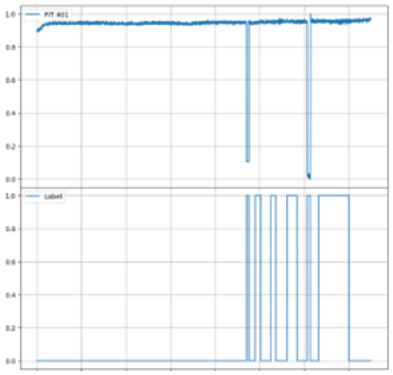
3 Étude des données brutes

Pour approfondir notre compréhension des données, nous examinons de près les caractéristiques de notre jeu de données, qui reflètent les sorties des divers capteurs et actionneurs du système SWaT. Cette étude préliminaire constitue une première étape cruciale pour cerner l'évolution des valeurs fournies par les cibles des six attaques que nous avons entreprises au cours de notre recherche.

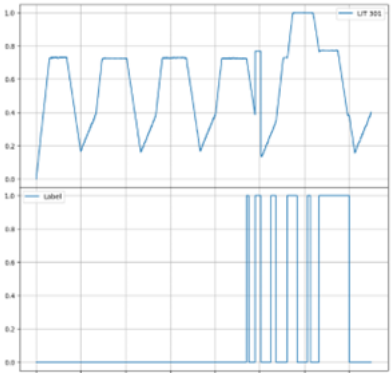
Il convient de souligner que chaque attaque est étroitement liée à un comportement anormal du capteur correspondant. Par exemple, la Figure 4a, qui représente le capteur FIT 401, cible de la première attaque, met en évidence de manière explicite les résultats aberrants générés au moment de ladite attaque.

Intéressant à noter, ce même capteur présente un comportement tout aussi anormal lors de la cinquième attaque, bien qu'il ne soit pas la cible directe de celle-ci. En effet, la cible réelle de cette attaque est l'actionneur MV 501. Cette observation nous amène à conclure que les effets d'une attaque peuvent se manifester sur divers éléments du système. Pour étayer cette hypothèse, nous avons entrepris une analyse de la racine carrée de la moyenne des carrés (RMS) sur les résultats de nos capteurs et actionneurs avant et pendant les attaques, puis nous avons procédé à une comparaison des résultats obtenus. Cela nous a permis d'observer des déviations inhabituelles sur divers capteurs et actionneurs, même lors d'attaques ciblant un élément unique. Prenons l'exemple de la sixième attaque, qui vise l'actionneur P 301 : on note une déviation marquée chez la cible de l'attaque, mais aussi chez l'actionneur LIT 401.

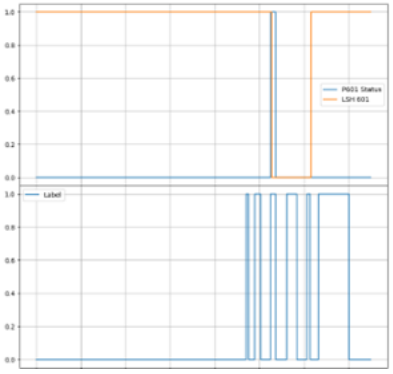
Ces constatations nous incitent fortement à envisager la détection des données aberrantes (outliers) comme une solution potentielle à notre problème de détection d'attaques. Il apparaît ainsi que cette approche pourrait apporter des avancées significatives dans notre démarche d'amélioration de la sécurité du système SWaT face à de telles menaces.



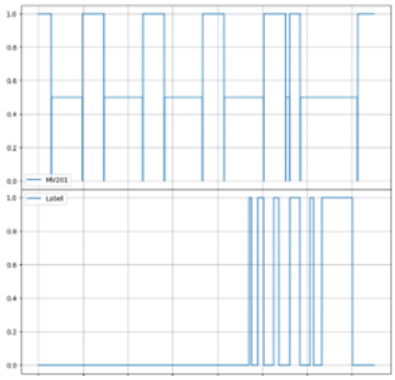
(a) FIT 401



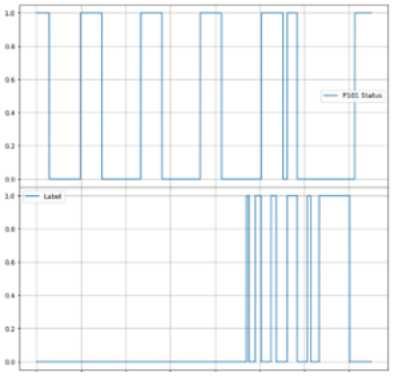
(b) LIT 301



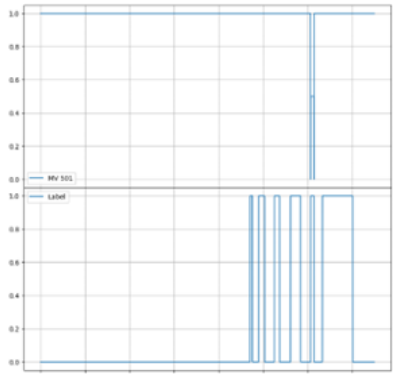
(c) P/LSH 601



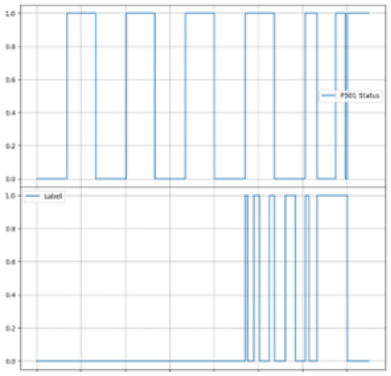
(d) MV 201



(e) P 101



(f) MV 501



(g) P 301

FIGURE 4 – Valeurs de sortie normalisées des cibles des attaques en fonction du temps, comparées au statut de la station (Attaquée ou Non)

4 Pipeline de traitement et d'analyse des données

Tout d'abord, les données tabulaires sous format CSV sont importées et rassemblées dans un DataFrame Pandas. Un traitement est fait pour remplacer les valeurs non définies par des valeurs bégnines pour l'apprentissage des modèles.
Ensuite, les données sont normalisées afin de pouvoir mieux visualiser et comparer les informations comprises dans le jeu de données.

Pour comparer et visualiser les prédictions des modèles avec les vraies attaques, nous effectuons une PCA (Principal Component Analysis) à 2 composants comme présenté sur la figure 5.

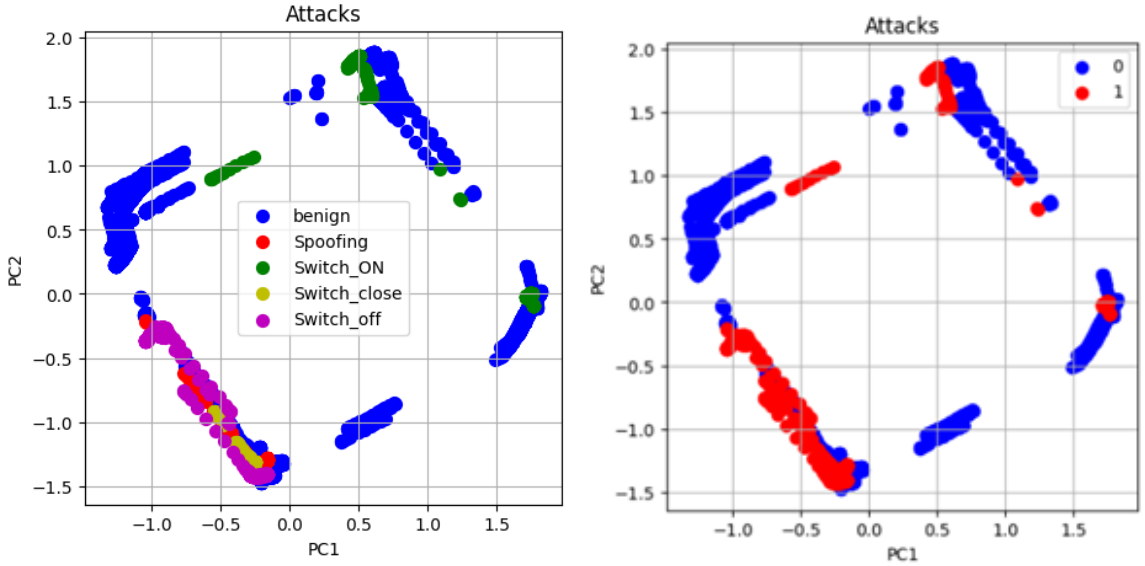


FIGURE 5 – PCA sur l'ensemble du jeu de données après pré-traitement

En ce qui concerne l'analyse des données et la détection des attaques, on a utilisé plusieurs méthodes d'apprentissage que l'on peut regrouper en 3 catégories :

- La détection d'anomalies non-supervisée avec les algorithmes Isolation Forest (IF) et Local Outlier Factor (LOF).
- Une classification de données séquentielles avec un LSTM (Long Short-Term Memory).
- La classification supervisée avec les méthodes Decision Tree, Random Forest, XGBoost et MLP (Multi Layer Perceptron).

Dans un premier temps, la séparation des données pour créer les jeux de données d'entraînement et de test a été fait de manière aléatoire en suivant une répartition arbitraire (par exemple : 80% pour l'entraînement et 20% pour le test).

4.1 Détection d'outliers non-supervisée

L'utilisation de méthodes non-supervisées est une bonne première étude pour ce jeu de données. En effet, à la vue du manque flagrant d'attaques labellisées, les algorithmes comme IF et LOF sont une réponse à ce genre de problème puisqu'ils n'utilisent pas de labels et se reposent sur d'autres particularités.

L'Isolation Forest est basé sur le concept que les anomalies sont plus rares que les exemples normaux. Il tire parti de cette caractéristique pour isoler rapidement les anomalies. L'idée fondamentale est que si vous prenez un échantillon aléatoire d'observations, les anomalies auront tendance à être plus isolées, c'est-à-dire qu'elles nécessiteront moins de divisions dans un arbre de décision pour être isolées des autres points de données.

Contrairement à IF, le Local Outlier Factor ne se base pas sur la construction d'arbres de décision aléatoires, l'idée centrale du LOF est d'évaluer la similarité locale de chaque point de données par rapport à son voisinage. Pour chaque point, on examine le degré auquel il est similaire à ses voisins les plus proches. Si un point est significativement moins similaire à ses voisins que les points voisins le sont entre eux, cela suggère qu'il pourrait être une anomalie.

Maintenant pour parler plus précisément du processus que l'on a appliqué, nous avons continuer de traiter la donner avant d'appliquer les algorithmes de détection. On enlève les colonnes contenant des informations que l'on ne veut pas telles que les labels et les timestamps.

Voici les résultats que l'on obtient pour les deux méthodes :

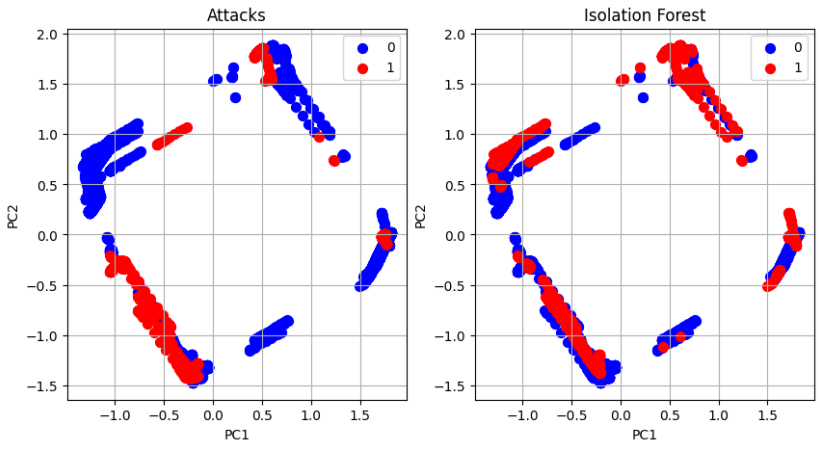


FIGURE 6 – Prédiction Isolation Forest

Accuracy	0.79
Precision	0.37
Recall	0.33
AUPRC	0.11
Matthews	0.23

TABLE 1 – Résultats des métriques d'évaluation

On peut voir que la méthode Isolation Forest permet de détecter les anomalies avec une accuracy de 0.79, une precision de 0.37 et un recall de 0.33. Ces résultats ne sont pas très bons. Dans notre cas, la métrique la plus importante est le recall car elle indique le nombre d'attaques détectées par rapport au nombre d'attaques réelles.

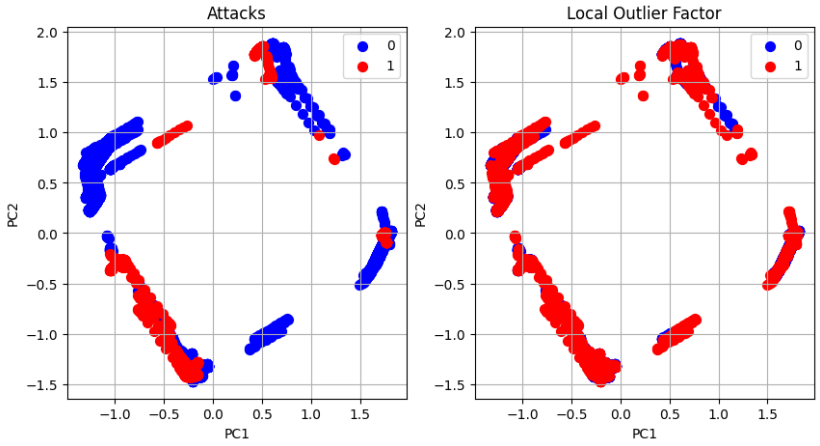


FIGURE 7 – Prédiction Local Outlier Factor

Accuracy	0.75
Precision	0.23
Recall	0.20
AUPRC	0.16
Matthews	0.07

TABLE 2 – Résultats des métriques d'évaluation

LOF fournit des résultats inférieurs à ceux de IF avec un recall de 0.20, cependant l'AUPRC (Area Under Precision Recall Curve) est supérieure à celle de IF, ce qui est très important dans notre cas car les classes sont très déséquilibrées.

4.2 LSTM

Un LSTM est un type de RNN (Recurrent Neural Network) utilisé en apprentissage automatique sur des données séquentielles. La raison pour laquelle nous avons choisi

ce modèle est que les données sont séquentielles et que les attaques sont des séquences d'actions qui durent un certain temps. On peut donc espérer que le LSTM puisse apprendre à détecter les attaques.

Les détails du modèle LSTM sont les suivants :

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 50)	19000
dense (Dense)	(None, 1)	51

TABLE 3 – Détail du modèle LSTM

Voici les résultats pour le modèle LSTM :

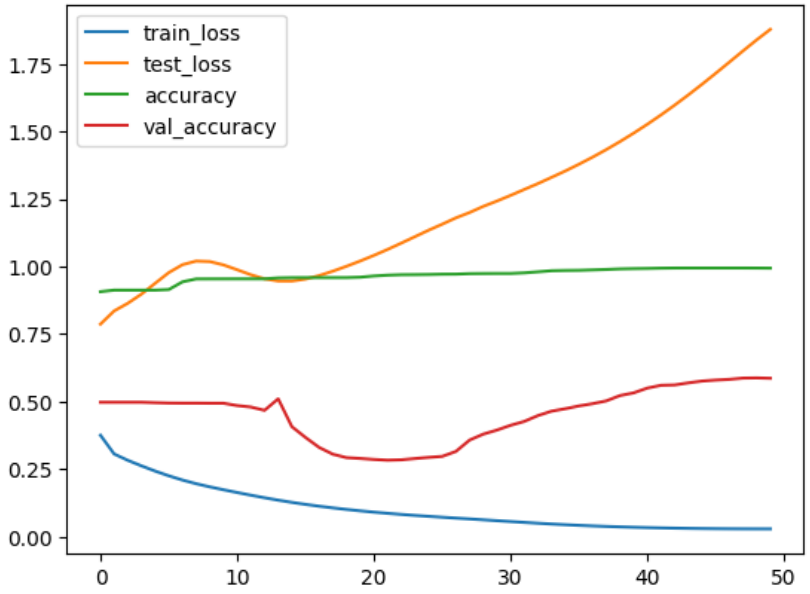


FIGURE 8 – Courbe d'apprentissage du LSTM

On peut voir que sur les données d'entraînement, le LSTM possède une accuracy presque parfaite, cependant sa validation accuracy est bien plus faible et proche de l'aléatoire.

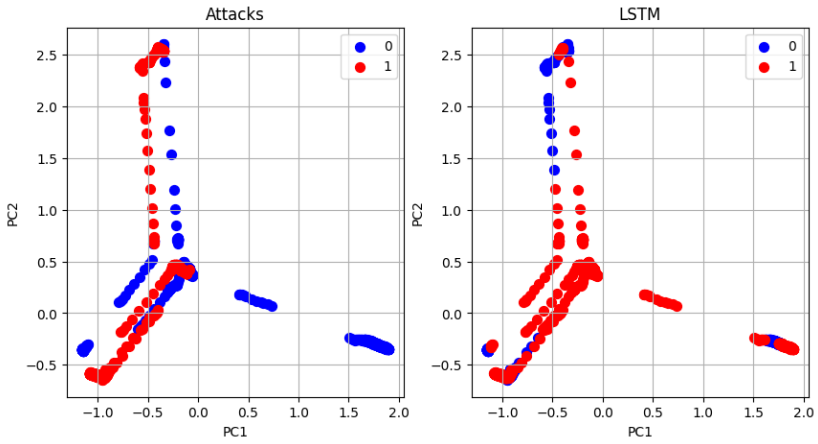


FIGURE 9 – Prédiction LSTM

Accuracy	0.59
Precision	0.56
Recall	0.86
Matthews	0.20

TABLE 4 – Résultats des métriques d'évaluation

Malgré une précision et une accuracy moyenne, le recall est de 0.86 ce qui est bon. On le voit bien sur la PCA, beaucoup de point sont prédits comme des attaques même s'il n'en sont pas mais par conséquent, beaucoup des attaques ont bien été prédites.

4.3 Classification supervisée

Le dataset présenté est déséquilibré, présentant 17% d'attaques et 83% d'attaques bénignes. Plusieurs techniques peuvent aider à contrebalancer nos données.

1. Undersampling

L'undersampling est une technique qui consiste à éliminer la classe majoritaire avant d'appliquer un algorithme de Machine Learning. Cependant, cette méthode présente un risque : elle peut ne pas bien fonctionner sur des données de test réelles biaisées car presque toutes les informations sont éliminées.

2. Oversampling

L'oversampling est une autre technique qui consiste à augmenter la classe minoritaire. Un exemple de cette technique est la Synthetic Minority Oversampling Technique (SMOTE). Cependant, l'oversampling peut parfois conduire à un surapprentissage car il répète les mêmes échantillons de la classe minoritaire.

Ces deux méthodes n'influent que très peu sur le problème majeur du manque de données et du surapprentissage.

Algorithmes de classification

1) MLP

Un MLP (Multi Layers Perceptron) est un réseau de neurones à propagation avant. Il utilise une fonction d'activation non linéaire pour transformer les entrées. L'erreur à la sortie du réseau est rétropropagée à travers le réseau pour ajuster les poids. C'est une méthode qui classifie simplement nos données en utilisant les labels.

Voici les résultats pour le Multi Layers Perceptron :

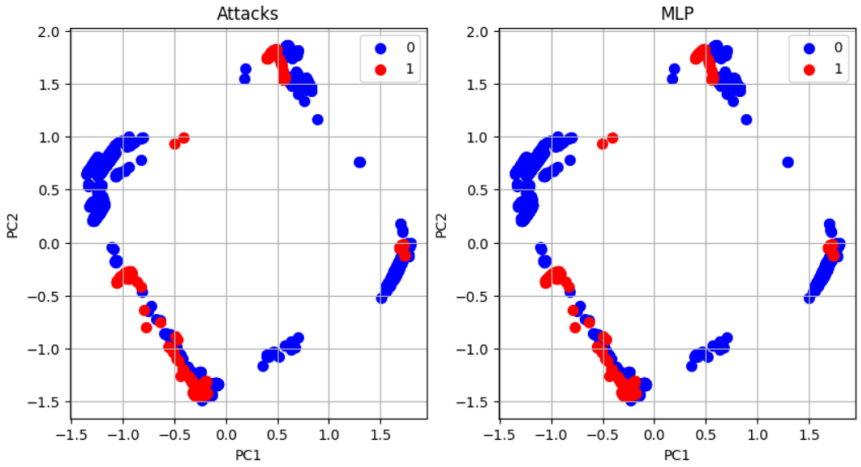


FIGURE 10 – Prédiction Multi Layer Perceptron

```
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
Matthews: 1.00
```

FIGURE 11 – Métriques d'évaluation du MLP

2) Arbres de décision

Les arbres de décision est un algorithme qui se base sur la subdivision de l'espace des données sur la base d'un arbre afin de faire de la classification. On peut donc voir l'arbre comme un ensemble de if où les feuilles seraient la classe prédite. L'objectif étant que si l'on connaît les attaques, elle vont produire des valeurs très différente et donc le découpage sera bien plus efficace. Deux exemples d'algorithmes basés sur les arbres de décision sont les forêts aléatoires et XGBoost.

a. Random Forest

Les forêts aléatoires sont un ensemble d'arbres de décision par classe de prédiction. Elles offrent de bons résultats pour les données déséquilibrées. Elle permette en générale de moins overfit et de mieux généralise sur l'apport de nouvelle donner externe au jeu

de donnée. Dans le cadre de cette entraînement le jeu de donnée est mélangé puis subdivisé.

Voici les résultats de la random forest :

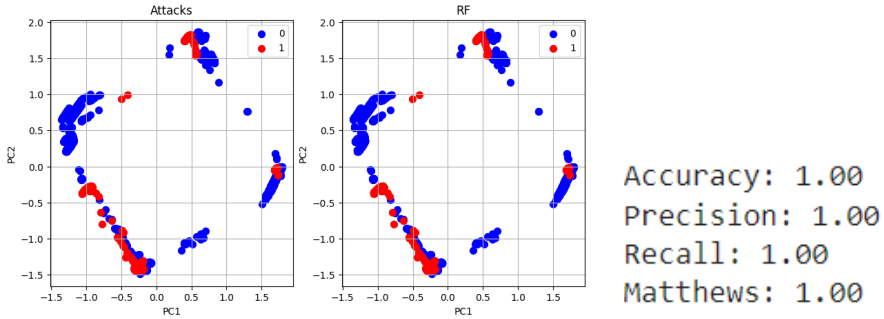


FIGURE 12 – Métriques d'évaluation de Random Forest

On remarque que le modèle à une une prédiction parfaite. Cela peut nous faire posé des doutes sur les données, le nettoyage des données ou sur la façon de subdivisé nos données lors de l'entraînement.

b. XGBoost

XGBoost, ou Extreme Gradient Boosting, est un algorithme qui utilise un ensemble d'arbres de décision pour faire des prédictions. Il est souvent le choix des gagnants des compétitions Kaggle. XGBoost gère bien les données déséquilibrées, fait face aux valeurs manquantes dans les données, permet une accélération via le traitement parallèle et pèse plus la classe positive par rapport à la classe négative. La prédiction XGBoost est basée sur l'ensemble d'arbres de décision. Chaque arbre donne une prédiction, et la prédiction finale est la moyenne des prédictions de tous les arbres.

Pour notre étude, nous avons choisit XGBoost.

Voici les résultats pour XGBoost :

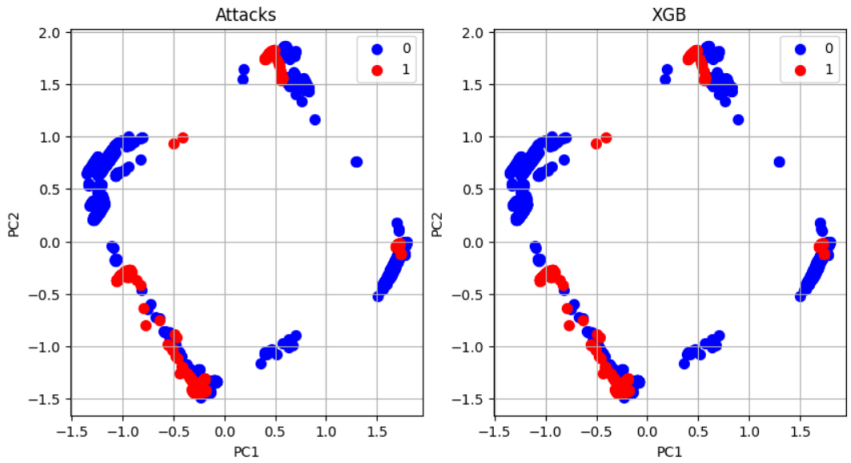


FIGURE 13 – Prédiction XGBoost

```
Accuracy: 1.00
Precision: 1.00
Recall: 1.00
Matthews: 1.00
```

FIGURE 14 – Métriques d'évaluation de XGBoost

5 Problèmes rencontrés

5.1 Séparation des données

Comme nous avons pu le voir dans notre benchmark, les résultats des modèles sont très bons. Avant d'aller plus loin, il serait intéressant de se demander ce qui pourrait expliquer d'aussi bons résultats.

En réfléchissant sur la manière dont étaient séparés les données à l'aide de la fonction train_test_split de sklearn, nous avons compris d'où venait ces bons résultats.

En effet, séparer le jeu de données de manière aléatoire avec train_test_split est une grosse erreur sur des données temporelles. Dans des données temporelles, les points sont très proches les uns des autres, de plus les labels sont tous regroupés ensembles de par la nature des attaques qui ont une certaine durée.

Si l'on schématise la séparation sur des données 1 dimension voici ce que l'on obtient :

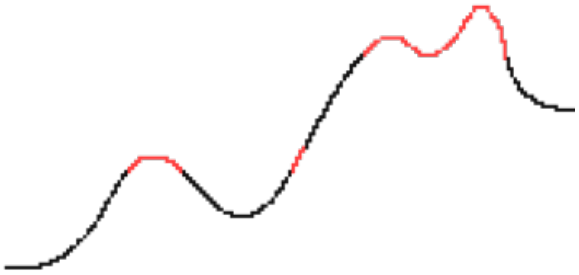
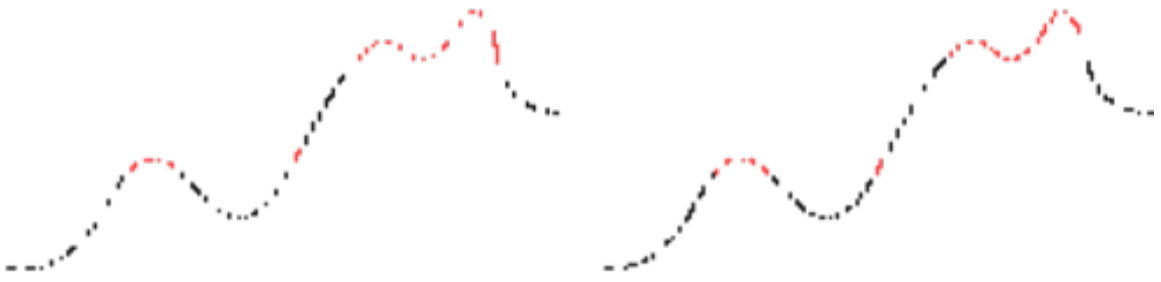


FIGURE 15 – Jeu de données



(a) Ensemble d'entraînement (b) Ensemble de test

FIGURE 16 – Séparation du jeu de données de manière aléatoire

Comme on peut le voir sur la Figure 16, les 2 courbes sont presque identiques. Ainsi apprendre sur la première courbe et tester ses performances sur la seconde implique

une performance presque parfaite pour un modèle ayant "mémorisé" tous les points du jeu de données d'entraînement.

Le fait que le paramètre random soit activé par défaut dans la fonction train_test_split de scikit-learn est donc très dangereux si l'on ne fait pas attention au type de données sur lequel on l'utilise.

Une solution naive pour résoudre ce problème serait de mettre simplement l'option random à False. De cette manière les 2 courbes de d'entraînements et de test auraient des point complètement différents. Cette solution pourrait effectivement fonctionner mais elle amène 2 problèmes. Le premier est le fait que cela introduise un biais entre nos deux jeux de données. Le dataset de train sera composé des premiers $x\%$ alors que le dataset de test contiendra les $(100 - x)\%$ suivants, ainsi si certains évènements ne se sont produis que vers la fin du dataset, ils ne seront pas pris en compte par le modèle.

Nous avons donc opter pour une méthode différente pour séparer les données. La méthode se déroule en 2 étapes. La première consiste à retirer entièrement des attaques, puis de les distribuer une à une aléatoirement à chaque modèle tout en respectant une distribution cohérente en fonction de la taille attendue des jeux de données. Ensuite pour le reste des données du dataset, nous les avons découpés en blocs de 10 minutes puis de manière similaire aux attaque, nous les avons distribués équitablement entre les deux jeux de données.

(a) Ensemble d'entraînement (b) Ensemble de test

FIGURE 17 – Séparation du jeu de données par blocs

On voit qu'avec cette séparation des données, le modèle n'a aucune possibilité de faire du sur-apprentissage.

5.2 Curse of dimensionality

Une fois que nous avions traité le problème de la séparation des données, nous nous sommes demandés pourquoi les résultats en apprentissage supervisé avaient drastiquement chutés. Certes, on pouvait s'attendre à une baisse des performances, mais pas autant que ce que nous pas pu observé. En regardant plus en détails les données ainsi

que la document du dataset nous avons finis par comprendre que nos faibles résultats étaient en réalité normal.

Le jeu de données comporte seulement 6 attaques distinctes et 4 natures différentes. Ainsi, certains types d'attaque ne se produisent qu'une seule et unique fois dans le jeu de données entier. Pour ce qui est des autres types d'attaques qui apparaissent seulement deux fois, il se produisent sur des capteurs différents et donc des colonnes complètement différentes. Le problème ici est donc que le modèle n'a aucune représentation des attaques sur les autres colonnes (7 colonnes sont concernées par les 6 attaques, sur un total de 44 colonnes). Il est donc impossible de faire prédire à un quelconque modèle des labels concernant une feature si l'on a aucun label positif impliquant celle-ci.

C'est ici que se trouve le problème de la malédiction de la dimensionnalité *(curse of dimensionality)*. La définition de ce théorème dit :

As the dimensionality of the features space increases, the number configurations can grow exponentially and thus the number of configurations covered by an observation decreases.

À mesure que la dimensionnalité de l'espace des caractéristiques augmente, le nombre de configurations peut croître de façon exponentielle et ainsi le nombre de configurations couvertes par une observation diminue.

Ainsi, plus la dimensionnalité du problème est élevé (ici 44 dimensions), plus le nombre de d'observations nécessaires pour couvrir l'ensemble des configuration est élevé (et croît exponentiellement). Dans notre cas, les observations présentes couvrent un champs très réduits de l'ensemble des configurations de cet espace.

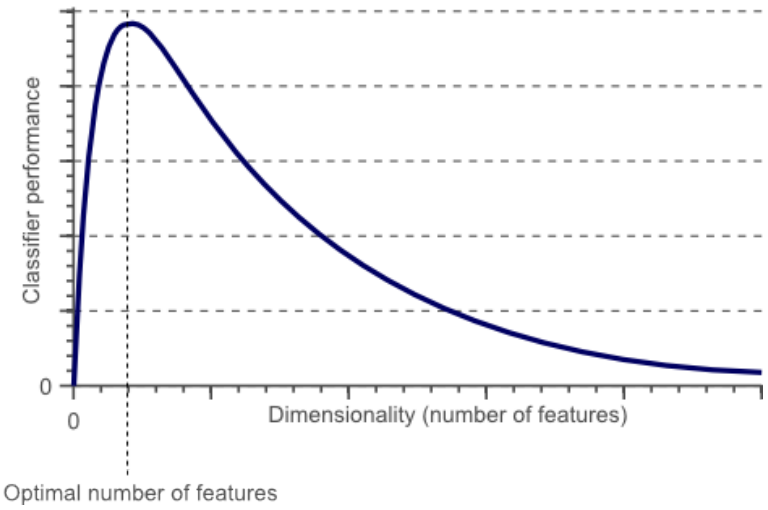


FIGURE 18 – Courbe de l'effet : Curse of Dimensionality

Pour appuyer cette affirmation, nous avons observé que chacune des 6 attaques du dataset, en plus de ne concerner qu'un seul capteur par attaque, ces attaques étaient fixes durant toute la durée de celle-ci.

0,796209753
0,796209753
0,796209753
0,796209753
0,5
0,5
0,5
0,5
0,5
0,5
0,5
0,5
0,5
0,5
0,5
0,5

FIGURE 19 – Effets de l'attaque 1 (Spoofing)

Par conséquent, ces points partagent énormément d'informations et peuvent presque être simplifiés en un seul point. En effet, la description de cette attaque est de passer la valeur de ce capteur de 0.8 à 0.5. On se rend tout de suite compte qu'aucun modèle ne peut généraliser sur des nouvelles attaques en ayant seulement 6 attaques contenant aussi peu d'informations.

6 Résultats finaux

Nous avons donc relancé tous nos algorithmes supervisés en prenant en compte les problèmes soulevés précédemment pour voir l'impact sur les performances de nos modèles.

6.1 LSTM

Tout d'abord, les résultats pour le LSTM :

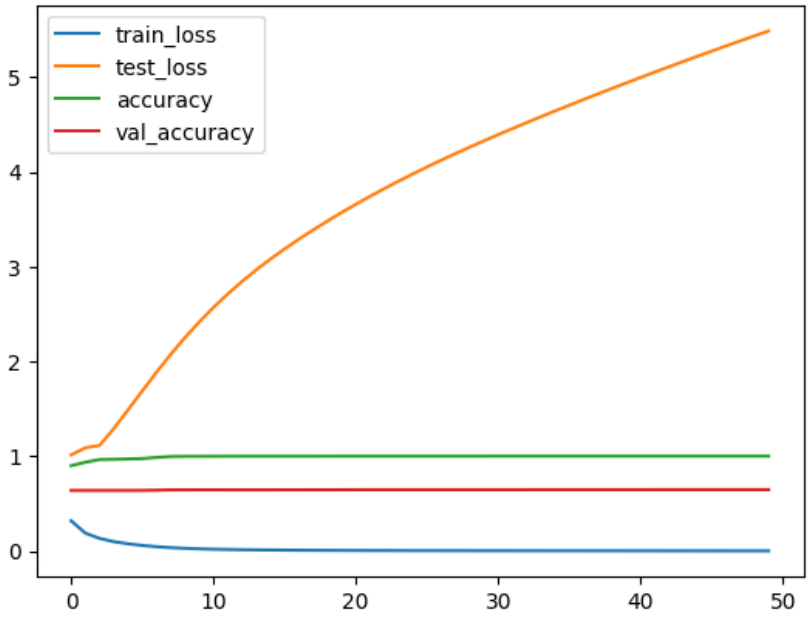


FIGURE 20 – Courbe d'apprentissage du LSTM

On peut voir que la test_loss augmente dramatiquement et que les accuracy ne subissent aucune variation.

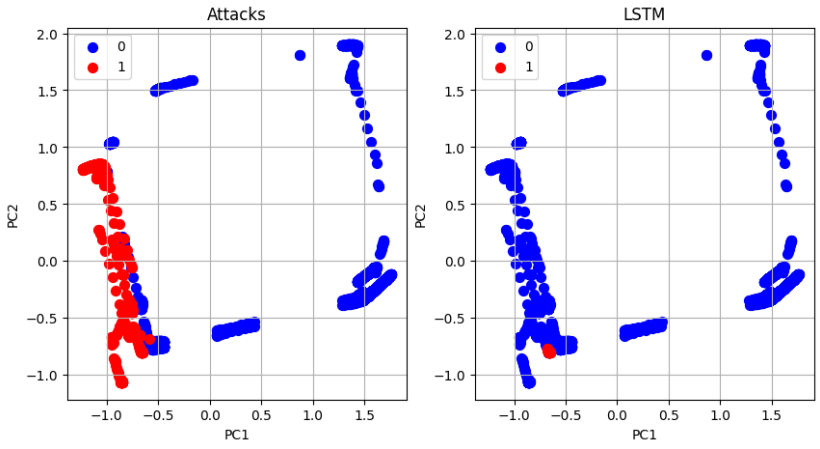


FIGURE 21 – Prédiction LSTM

Accuracy	0.65
Precision	1.00
Recall	0.02
Matthews	0.12

TABLE 5 – Résultats des métriques d'évaluation

Le modèle ne prédit plus qu'une seule attaque. Même si elle est effectivement une vraie attaque, le modèle n'est vraiment pas efficace avec un recall de 0.02. Ici, la précision n'est pas une métrique sur laquelle on peut se reposer.

6.2 Classification supervisée

Résultats pour MLP

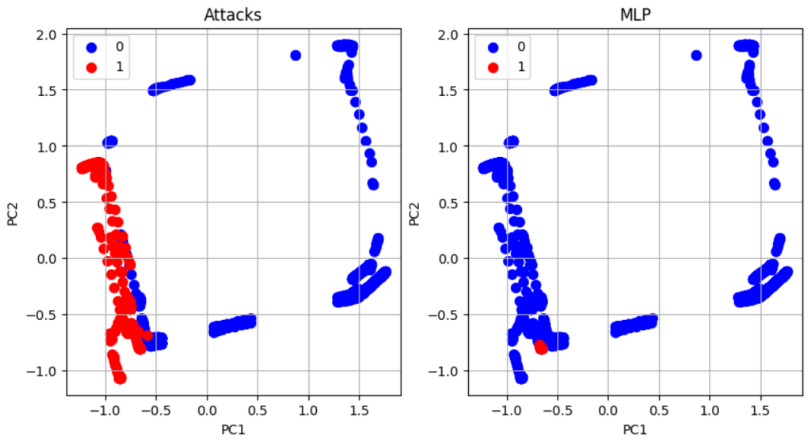


FIGURE 22 – Prédiction MLP sur un dataset non randomisé

```
Accuracy: 0.65
Precision: 1.00
Recall: 0.02
Matthews: 0.11
```

FIGURE 23 – Métriques d'évaluation MLP sur dataset non randomisé

On remarque qu'il a prédit une attaque, la même que le LSTM.

Résultats pour la Random Forest :

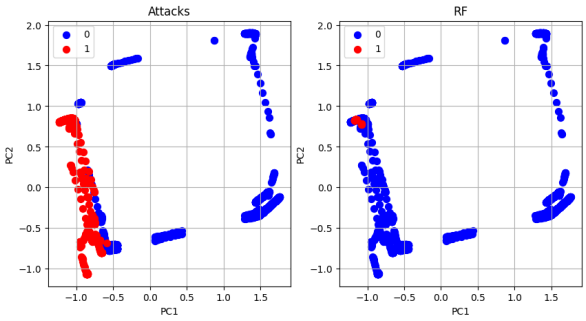


FIGURE 24 – Prédiction Random Forest sur un dataset non-randomisé

Accuracy	0.66
Precision	1
Recall	0.06
Matthews	0.20

TABLE 6 – Métrique Random Forest sur un dataset non-randomisé

Lors de ces résultats le modèle a appris sur un sous partie des valeurs normale et avec seulement 3 attaques sur les 6. On remarque qu'avec les résultats du modèles, la random forest a permis de découvrir un attaque qui n'avais jamais été vu sur les trois. Ce qui provoque un très faible recall, ce qui implique que cette méthode n'est pas fiable pour détecter des attaques jamais rencontré.

Résultats pour XGBoost :

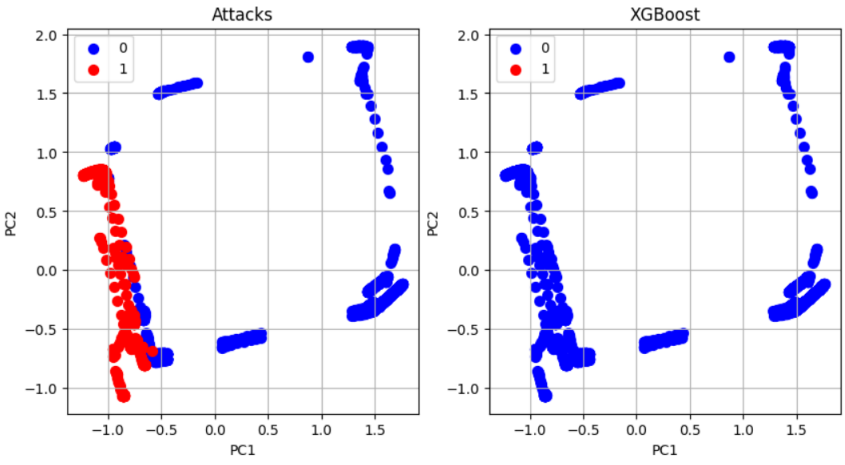


FIGURE 25 – Prédiction XGBoost sur un dataset non randomisé

```
Accuracy: 0.64
Precision: 0.00
Recall: 0.00
Matthews: 0.00
```

FIGURE 26 – Métriques d'évaluation XGBoost sur dataset non randomisé

7 Conclusion

Comme nous avons pu le voir dans ce rapport, l'apprentissage supervisé comporte certains problème dans le cadre du jeu de données utilisé. Que ce soit par la nature des données temporelles, au niveau de la séparation des données, mais aussi par le trop faible nombre d'observations différentes présentent dans les données. L'apprentissage non-supervisé semble donc être la seule issue à la détection d'outliers dans ce dataset. Ainsi, le modèle nous ayant permis d'obtenir les meilleurs performances est Isolation Forest dont le principe est de détecter les valeurs facilement séparables par des arbres dans le jeu de données.