Attaques adversariales sur les systèmes de détection d'intrusion basés sur l'apprentissage

Projet d'Application - FISE 2 - Sujet 4

Objectif : Étudier la vulnérabilité des IDS basés sur le Machine Learning face aux attaques adversariales et proposer des pistes de défense.

Public ciblé : Analystes en cybersécurité et pentesteurs souhaitant tester les techniques adversariales.

Equipe 41

MALQUY Valentin (Responsable Scrum)
CHOISNET Alexin (Lead Dev)
DEVERNAY Loup (équipe de dev)
GASQUET Paulin (équipe de dev)
AZZOUG Mohand (équipe de dev)

28 novembre 2025

Table des matières

1 Introduction

L'intelligence artificielle (IA) et l'apprentissage automatique (Machine Learning) sont devenus des piliers incontournables de la cybersécurité moderne. Ils permettent d'analyser des volumes massifs de trafic réseau pour identifier des menaces que des règles statiques ne pourraient détecter, améliorant la sécurité des trafics réseaux en premier lieu.

Cependant, cette dépendance à l'IA introduit une nouvelle surface d'attaque : la vulnérabilité des modèles aux attaques adversariales. Si un attaquant comprend comment le modèle réfléchit, il peut manipuler ses entrées, les modifiant légèrement pour passer inaperçu et ne pas être détecté. Ces données manipulées, bien que indétectables à l'œil humain, entraînent des erreurs de classifications très importantes qui amènent le système de détection à accepter des entrées malveillantes. C'est dans ce contexte du jeu du chat et de la souris que s'inscrit notre projet sur les attaques adversariales contre les systèmes de détection d'intrusion (NIDS).

Le code correspondant à cette étude : `https://github.com/Alexin-CH/AdversarialNIDS`

2 Contexte et données

2.1 Mise en place des datasets

Dans un premier temps, nous avons dû mettre en place des algorithmes de détection d'attaques réseaux afin de simuler des attaques adversariales. Les algorithmes que nous avons considérés font partie de la catégorie des algorithmes de classification. Il nous fallait donc utiliser une quantité importante de données, que ce soit pour l'entraînement des modèles (réseaux de neurones, Random Forest, ...) ou pour la validation des résultats. Pour cela, il a été considéré deux datasets : CICID2017 et UNSW-NB15 qui sont à la fois de grande taille et très représentatifs de trames réseaux classiques durant l'absence ou la présence d'attaque sur les réseaux.

2.2 Prétraitement des données

La première étape fut le traitement de ces données. En effet, les datasets contiennent de nombreux échantillons que nous ne voulons pas considérer. Pour effectuer ce traitement préliminaire, nous filtrons les données et supprimons et/ou modifions certains champs. Des filtres appliqués sont la suppression des doublons, le traitement des valeurs NaN, ainsi que des filtres qui vérifient le respect des types de données (ex : une durée est strictement positive). Certaines colonnes telles que l'adresse IP contenaient une unique valeur, ce qui n'apporte donc pas d'information pour la classification.

Suivant le dataset considéré, les types d'attaques ainsi que la distribution des attaques n'est pas la même. Le dataset UNSW-NB15 en particulier possède une répartition des types d'attaques disproportionnée par rapport à celle de l'absence d'attaque. Cette disproportion est toujours présente après un regroupement des classes d'attaques minoritaires en classes plus grandes.

Une distinction est donc faite sur les deux datasets : UNSW-NB15 sera seulement utilisé pour les entraînements dits bi-classes (détection de la présence d'attaque). CICID2017 sera utilisé pour les deux types d'entraînements, bi-classes et multi-classes (détection du type d'attaque, l'absence d'attaque étant considéré dans la catégorie attaque "bénigne"). Voir Annexe 1.

3 Modèles de Prédictions

Trois types de classifier ont été implémentés et testés :

3.1 Decision Tree et Random Forest

Il s'agit d'approches de classification reposant sur la mesure de l'entropie associée à chaque caractéristique des données. Le calcul de l'entropie est utilisé pour la création des arbres de décision : les données sont séparées en sous-groupes au fur et à mesure du parcours d'un arbre suivant si elles valident un certain seuil par rapport aux paramètres considérés par l'arbre.

Un Decision Tree est un modèle de base, rapide à entraîner et interprétable facilement.

Un Random Forest est un ensemble d'arbres de décision combinés, offrant robustesse et meilleure généralisation. Chaque arbre est entraîné sur un échantillon et n'examine qu'un sous-ensemble aléatoire de features, ce qui réduit la corrélation entre arbres et stabilise les prédictions. La prédiction se fait alors "à mains levées", la classe prédite par le Random Forest est la classe majoritairement prédite par l'ensemble des arbres de décisions présents dans la forêt.

3.2 K-Nearest Neighbors (KNN)

Un modèle K-Nearest Neighbors est un modèle de classification basé sur la similarité entre les observations. Chaque point est assigné à la classe majoritaire parmi ses k plus proches voisins, déterminé par une mesure de distance. Sa performance dépend fortement du scaling des données, ce qui nécessite une normalisation au préalable. KNN a principalement été évalué pour comparaison avec les modèles d'arbres. Il présentait une capacité à détecter les attaques particulièrement importante (Accuracy proche de 100%). Voir Annexe 4.

3.3 Réseaux de neurones PyTorch

3.3.1 Multi-Layer Perceptron (MLP) : Architecture classique de réseau de neurone en utilisant la fonction d'activation Mish avec la fonction Cross Entropy pour le calcul de la Loss. Voir Annexe 3.

3.3.2 Convolutional Neural Network (CNN) : Les modèles CNN n'ont pas présenté de résultats intéressants lors des tests sur les données que nous possédons. Nous nous sommes donc concentrés sur le modèle précédent.

3.4 Comparaison des modèles

Les données ont été prétraitées telles que présentées en partie 2, puis séparées en ensemble d'entraînement et de test, avec une attention portée à la prévention du data leakage et à la gestion du déséquilibre des classes. Chacun de ces modèles a été optimisé en recherchant les hyperparamètres qui maximisent le score de validation, à l'aide de Grid Search. Ces modèles sont sauvegardables et exportables après entraînement.

Le modèle Random Forest a été sélectionné comme modèle principal en raison de sa robustesse et de son habileté à généraliser sur des données bruitées et déséquilibrées. Une fois optimisé, il obtient d'excellentes performances (proche de 100%) sur la détection d'intrusion réseaux. Par rapport au Decision Tree, le Random Forest réduit la variance des arbres individuels, produisant des prédictions plus stables ; par rapport au KNN, il est plus rapide

à entraîner, moins sensible à l'échelle des features et mieux adapté au grand jeu de données que nous possédons. Le modèle permet d'identifier les features les plus influentes sur le modèle, ce qui n'est pas impossible sur KNN, offrant ainsi un avantage pour l'analyse des méthodes d'attaque à développer par la suite.

Sur un subset de 400 000 données :

	Avant preprocess		Après preprocess		Après optimisation	
	Précision	Temps entraînement	Précision	Temps entraînement	Précision	Temps entraînement
MLP	0.853	5-10 min	0.976	5-10 min	0.982	5-10 min
KNN	X	X	0.973	3s	0.985	4s
Decision Tree	0.475	10min	0.965	3s	0.998	30s
Random Forest	0.521	+1h	0.967	3s	0.999	15min

FIGURE 1 – *
Comparaison des précisions et temps d'entraînement des différents modèles à différentes étapes du pipeline.

4 Les différentes Attaques

4.1 Accès au modèle (boîte blanche)

L'attaque FGSM (sur les réseaux profonds PyTorch)

Le but est de se rapprocher le plus possible de la frontière de décision du modèle. Pour cela nous utilisons l'auto-différenciation et la backpropagation des réseaux PyTorch : le gradient est calculé individuellement sur chaque donnée d'entrée perturbable et il est utilisé pour connaître la direction à suivre pour trouver la frontière entre la classe actuelle de l'entrée et la classe ciblée par l'attaquant (la classe bénin).

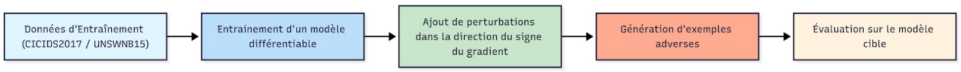


FIGURE 2 – *
Attaque FGSM sur les réseaux PyTorch : exemples adversariaux générés.

Il est important de noter que de nombreuses caractéristiques des entrées sont de seconds niveaux, c'est-à-dire qu'elles sont calculées à partir des valeurs d'autres caractéristiques dites de premiers niveaux.

4.2 Pas d'accès au modèle (boîte noire)

Toutes les attaques ne pouvaient pas être appliquées à tous les types de modèles que nous avons implémentés. Certaines attaques utilisent le calcul du gradient pour la perturbation des données et les modèles type KNN et Random Forest ne possèdent pas de calcul de gradient. Pour cela, nous avons mis en place des méthodes pour permettre un calcul de gradient sur ces types d'algorithmes.

La méthode Surrogate

Un MLP de faible taille est entraîné sur les mêmes données que celles d'entraînement du modèle attaqué. Le but est de simuler le comportement d'un modèle dont on ne connaît pas les poids ou l'architecture. Des données sont ensuite perturbées par attaque FGSM sur le MLP dit surrogate puis utilisées pour tromper le modèle attaqué.

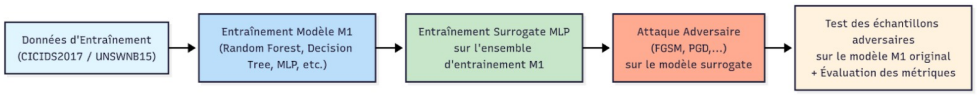


FIGURE 3 – *
Méthode Surrogate : visualisation des perturbations générées.

La méthode Substitut

Cette méthode repose sur la transférabilité des attaques adverses. Un MLP (dit substitut) de faible taille est entraîné sur des données étiquetées par le modèle attaqué. Le substitut est entraîné à fournir les mêmes prédictions que le modèle attaqué pour les mêmes données. Les données sont perturbées par attaque sur le substitut (attaque FGSM par exemple) puis renvoyées au modèle attaqué. Les deux modèles ayant été entraînés sur des tâches similaires à haute dimension, la perturbation du substitut permet la perturbation du modèle original.

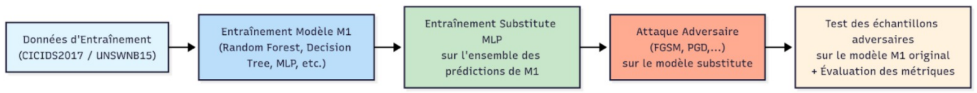


FIGURE 4 – *
Méthode Substitut : perturbation et transfert des attaques.

Exemples dans le cas du Random Forest pour les données d'attaques non bénignes :

	RF **avant attaque**	Substitut **avant attaque**	Substitut **après attaque**	RF **après attaque**
Précision	0.9982	0.9649	0.9996	0.9966
Recall	**0.9956**	**0.9897**	**0.0329**	**0.0427**

FIGURE 5 – *
Random Forest : performance et robustesse après attaque.

Voir Annexes 2.1 à 2.4

L'attaque HopSkipJump

Une méthode d'attaque adversariale sans gradient qui permet de générer des exemples adversariaux en perturbant itérativement une entrée initiale. Elle fonctionne en estimant la direction du gradient à la frontière de décision du modèle cible à l'aide des prédictions, ce qui la rend efficace en requêtes et applicable même aux modèles non différentiables.

Lien du papier : `https://arxiv.org/abs/1904.02144`.

Cette méthode s'avère efficace sur les modèles les moins robustes, en particulier sur Decision Tree. (Annexe 1.1 et 1.2)

Modèle	Temps d'exécution	Précision	Rappel
Decision Tree	10 s	0.9947	0.3824
Random Forest	35 min	0.9974	0.9452
KNN	20 min	0.9745	0.3953

TABLE 1 – Métriques moyennes de l'attaque HopSkipJump sur les modèles non différentiables (2000 attaques).

Par souci de réalisme, nous appliquons des contraintes sur les données adversariales (Annexe 6.1) :

— **Contrainte de bordure** : ne pas dépasser les valeurs maximales des features du dataset pour éviter les valeurs aberrantes.

— **Contrainte de type** : garder la sémantique et la cohérence des données en imposant un type particulier pour la feature de l'échantillon (exemple : 0.33 paquet n'a aucun sens informatique).

Il est intéressant de noter que le **Random Forest** était le plus résistant aux attaques lors de nos tests. La méthode **substitut** a cependant permis de le perturber là où la méthode **surrogate** a échoué.

5 Résister aux Attaques

Nous avons utilisé les attaques que nous avons mises en place pour tester des améliorations de robustesse de nos modèles.

Une première idée a été d'entraîner un modèle avec des données à la fois extraites des datasets et générées par nos attaques adversariales. Le résultat est grandement dépendant de la répartition entre donnée "classique" et donnée "adversariales". Une trop grande quantité des premières noyait les secondes et ne présentait pas une amélioration significative de robustesse. Un déséquilibre inverse permettrait une résistance aux attaques adversariales mais rendrait le modèle perdu face à des données non malveillantes, augmentant les mauvaises classifications.

Des différents modèles que nous avons utilisés, le Random Forest s'est montré le plus robuste par sa difficulté à être perturbé : l'augmentation du nombre d'arbres composant la forêt permet une amélioration directe de la robustesse du modèle.

	RF simple après attaque	Robuste RF après attaque
Précision	0.8846	0.9295
Recall	**0.1779**	**0.7479**

FIGURE 6 – *
Analyse des sensibilités : influence des features sur le modèle.

Une augmentation du nombre d'arbres dans la forêt ainsi qu'une partie des données d'entraînement ayant pour origine une attaque adversariale permet au modèle de garder une capacité de classification proche de la vérité. Voir annexe 5.1 à 5.3.

Les attaques adversariales se font en jouant sur les frontières de classifications des algorithmes de détection, leur efficacité provient de la finesse de ces frontières. Des frontières moins fines seraient moins vulnérables aux attaques, mais également moins efficaces dans leur tâche de classification.

6 Conclusion

L'étude menée au cours de ce projet sur les différents modèles de détection d'attaque réseau a permis de mettre en lumière la faiblesse de certains de ces modèles face à certains types d'attaques adversariales ainsi que la résistance relative d'autres tels que le Random Forest. Cependant, comme illustré dans la partie III, plus une attaque adversariale est sophistiquée, plus ses chances de succès sont hautes. La prise en compte de ces attaques pendant la mise en place des modèles de détection est un premier pas vers des modèles de détection adaptables à tout scénario. Même si la protection parfaite n'existe pas, il est possible d'imaginer des modèles à plusieurs niveaux dont certains prennent en compte l'existence connus des attaques types adversariales.

Annexes

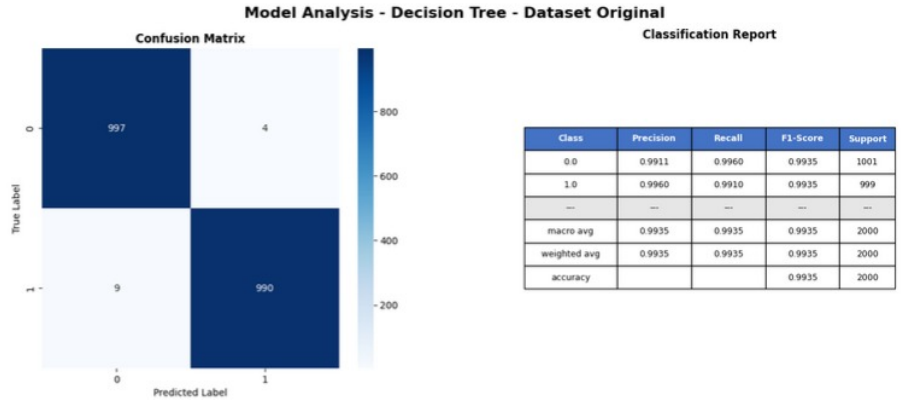


FIGURE 7 – *
Annexe 1.1 : Dataset CICIDS2017 Avant Attaque HSJ Decision Tree

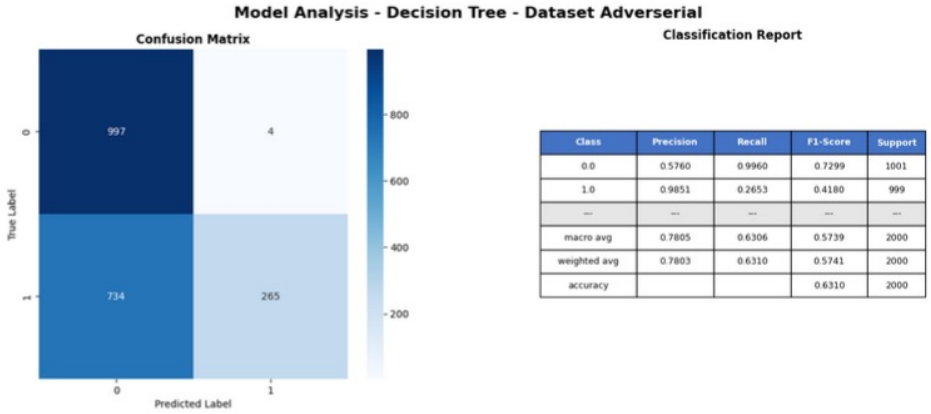


FIGURE 8 – *
Annexe 1.2 : Dataset CICIDS2017 Avant Attaque HSJ Decision Tree

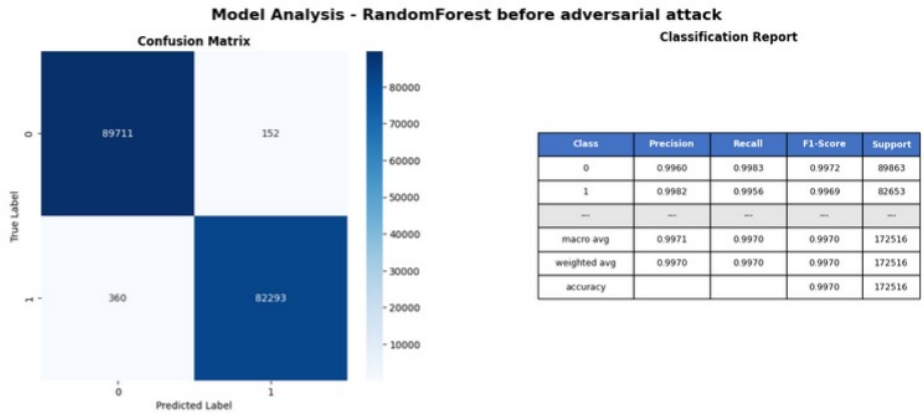


FIGURE 9 – *
Annexe 2.1 : Dataset CICID-2017_RF_Avant_Attack

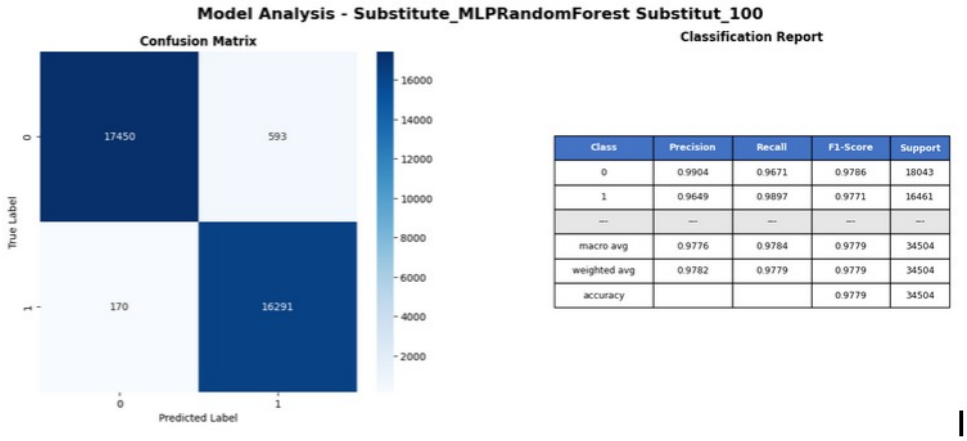


FIGURE 10 – *
Annexe 2.2 : Dataset CICID-2017_Substitut_Avant_Attack

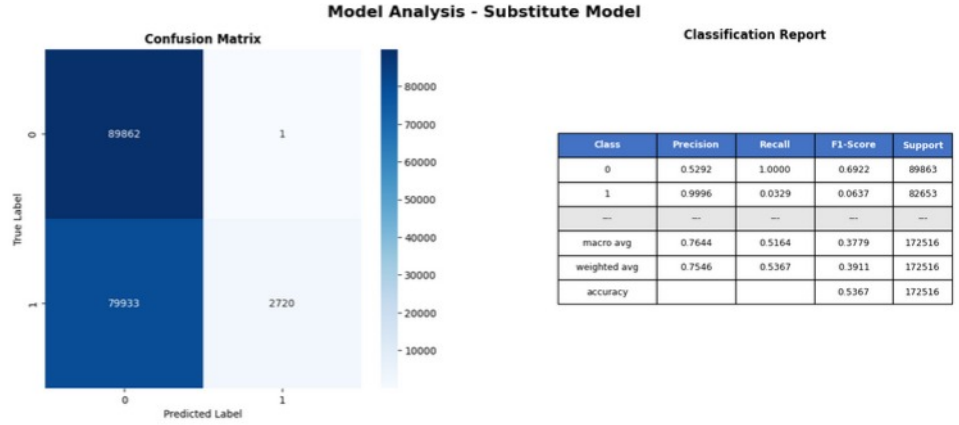


FIGURE 11 – *
Annexe 2.3 : Dataset CICID-2017_Substitut_Apres_Attack

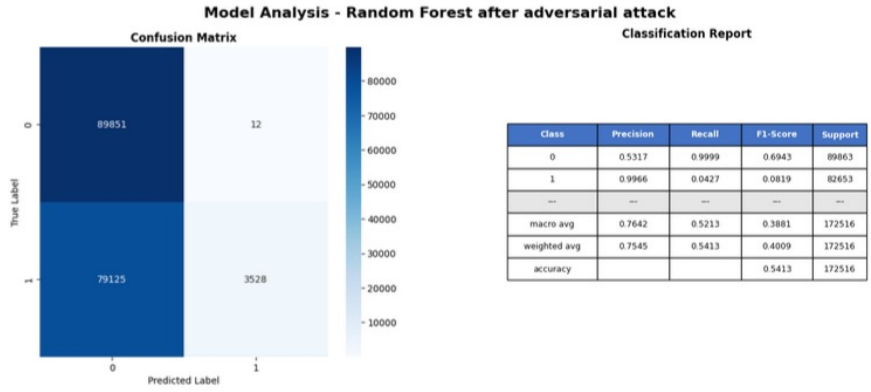
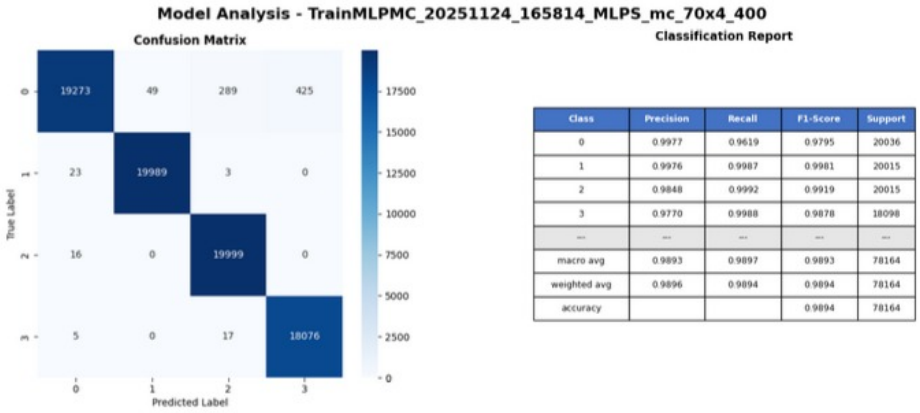
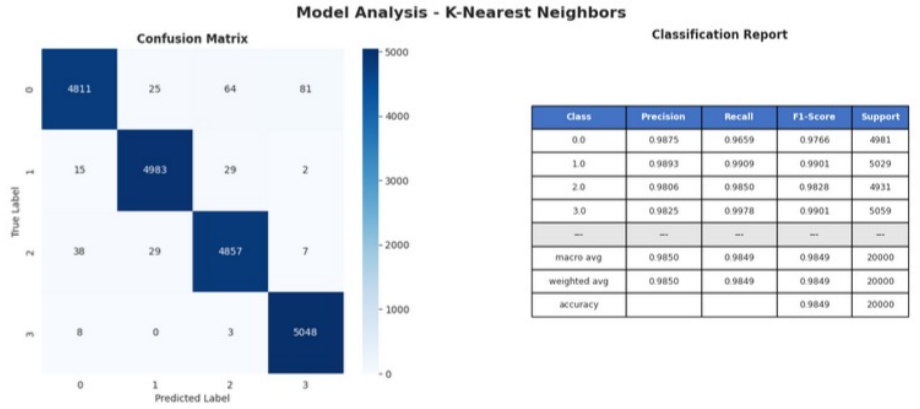


FIGURE 12 – *
Annexe 2.4 : Dataset CICID-2017_RF_Apres_Attack



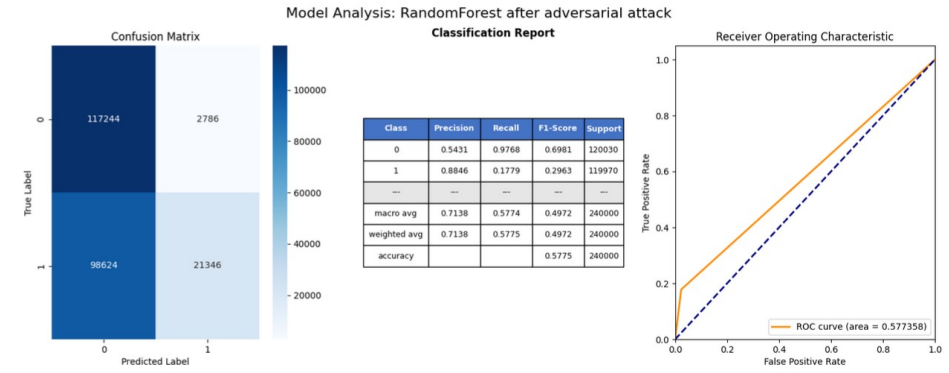
Class	Precision	Recall	F1-Score	Support
0	0.9977	0.9619	0.9795	20036
1	0.9976	0.9987	0.9981	20015
2	0.9848	0.9992	0.9919	20015
3	0.9770	0.9988	0.9878	18098
...	...	...	...	...
macro avg	0.9893	0.9897	0.9893	78164
weighted avg	0.9896	0.9894	0.9894	78164
accuracy			0.9894	78164

FIGURE 13 – *
Annexe 3 : Dataset CICID-2017_MLP_Multi_classes



Class	Precision	Recall	F1-Score	Support
0.0	0.9875	0.9659	0.9766	4981
1.0	0.9893	0.9909	0.9901	5029
2.0	0.9806	0.9850	0.9828	4931
3.0	0.9825	0.9978	0.9901	5059
...	...	...	...	...
macro avg	0.9850	0.9849	0.9849	20000
weighted avg	0.9850	0.9849	0.9849	20000
accuracy			0.9849	20000

FIGURE 14 – *
Annexe 4 : Dataset CICID-2017_KNN_Multi_classes



Class	Precision	Recall	F1-Score	Support
0	0.5431	0.9768	0.6981	120030
1	0.8846	0.1779	0.2963	119970
...	...	...	...	...
macro avg	0.7138	0.5774	0.4972	240000
weighted avg	0.7138	0.5775	0.4972	240000
accuracy			0.5775	240000

FIGURE 15 – *
Annexe 5.1 : Dataset CICID-2017_RF_apres_attack

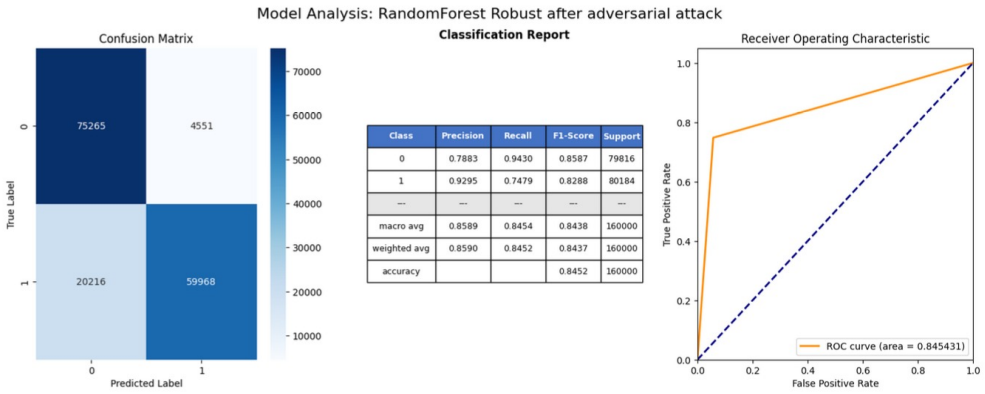


FIGURE 16 – *
Annexe 5.2 : Dataset CICID-2017_Robust_RF_apres_attack

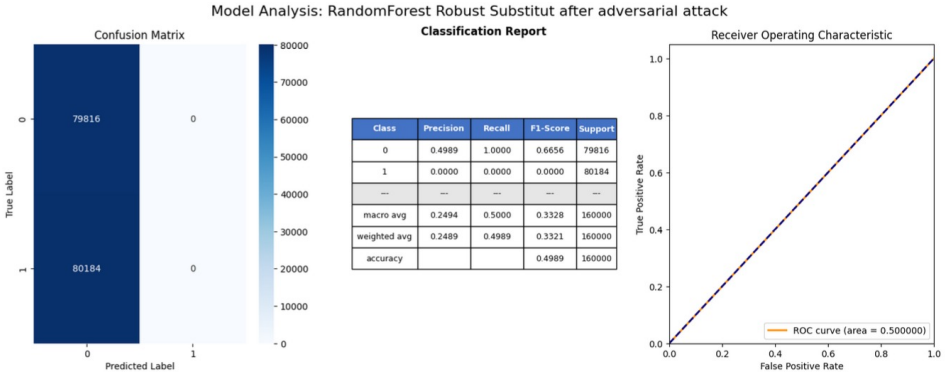


FIGURE 17 – *
Annexe 5.3 : Dataset CICID-2017_Robust_RF_substitut_apres_attack

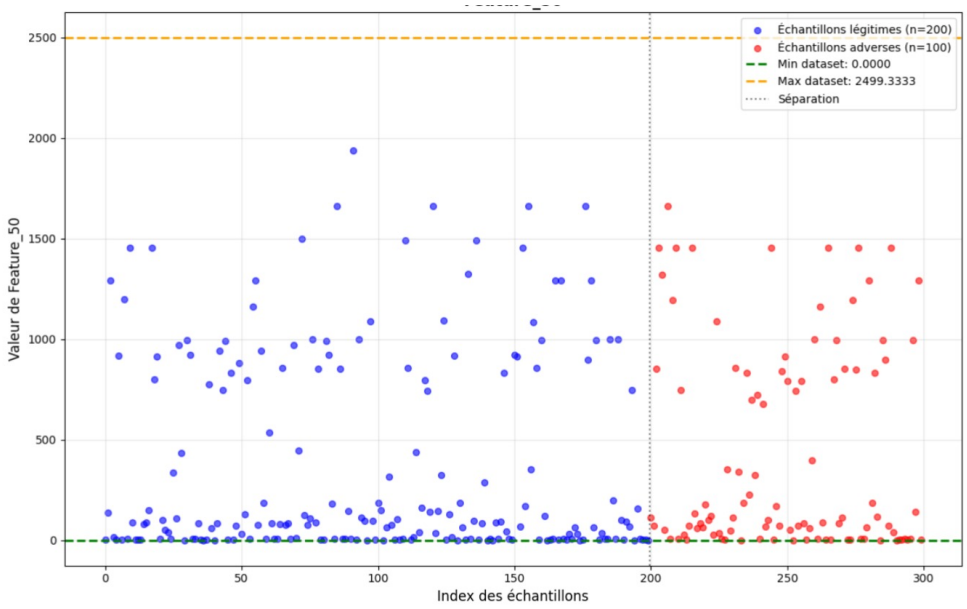


FIGURE 18 – *
Annexe 6.1 : Visualisation des contraintes réalistes sur 200 échantillons légitimes et 100 échantillons adverses sélectionnés aléatoirement