

Concordia University

A Study on the Impact of Data Poisoning on Machine Learning Intrusion Detection Systems

Github repository

SOEN 321 – Information System Security

Fall 2025

Professor Amr Youssef

Submitted by:

Shayan Goldstein

Hiba Talbi

Hossam Khalifa

Varak Mikirditsian

Honorine Ishimwe Twahirwa

Hayk Petrosyan

Contents

Abstract 3
1. Introduction 3
2. Methodology 4
2.1. Dataset Overview: 4
2.2. Preprocessing 6
2.3. Training and Testing Setup 6
3. Results 7
3.1 Types of attacks 7
3.2. Attack Hiding 8
3.3. DoS Poisoning 8
3.4. Random Label Flip 8
3.5. Summary 9
4. Conclusion 9
5. References 10

List of Tables

Table1: Feature Categories and Descriptions 5
Table2: Clean Model vs Poisoned Model result comparisons 11
Table3: Clean Model vs (Attack hiding 5% 10%) result comparisons 12

List of Figures

Figure 1: Attack hiding 5% Confusion Matrix 12
Figure 2: Attack hiding 10% Confusion Matrix 12
Figure 3: DOS 10% Confusion Matrix 13
Figure 4: Random Flipping 10% Confusion Matrix 13

Abstract

Since the dawn of the internet, servers and interconnected devices have been tightening their systems security, but hackers have also been coming up with more creative ways to infiltrate them. A crucial part in networks system security is the Intrusion Detection System (IDS). They are built using machine learning algorithms to detect suspicious activities by analyzing network data and report them. Even though they are efficient in protecting network infrastructures, what happens if the data they were trained on was poisonous to begin with? The purpose of this study is to examine the effects of using a poisoned dataset to train an Intrusion Detection System. By using a popular network traffic dataset known as NSL-KDD, we've crafted several sets of network data flow with each containing a different attack to be tested. We've then trained a model for each dataset and compared the results with a clean model.

Firstly, our clean model, trained with an accurate dataset, had an accuracy of about 86%, meaning 86% of all its categorizations were correct. After running tests for data poisoning on our system using 3 categories of label-flipping (Attack-Hiding, Dos Poisoning, Random Label Flipping) and flipping 10% of the dataset labels for each method, Attack-Hiding was found to be the most effective one, decreasing the accuracy by 4.1%, and increasing missed attacks from 2027 to 2753. The model trained on the poisoned Attack-Hiding dataset, which consists of manipulating malicious data traffic to make it look like normal, lowered the detection accuracy rate and increased its missed attacks significantly. The DoS Poisoning attack, which consists of flipping normal data to be labeled as attacks, had a marginal effect on the accuracy relatively decreasing the accuracy only by 1%, and having 2133 missed attacks. Finally, the Random Label Flipping attack, seemed to be more effective than the DOS but less than attack hiding by adding random noise to the training set and simply resulting in worsening its overall performance. The latter had decrease in accuracy by 2% and had 2267 missed attacks.

After discovering that Attack-hiding was the most effective method, a further test was ran using 5% attack-hiding to further observe the effect it had on the model and what trend, if any, correlates between the % poisoned and the resulting accuracy differences. The resulting trend was almost linear with increased data poisoning via attack hiding causing a linear accuracy loss for our model.

The model and testing were done on this GitHub: https://github.com/Yuioytffhio/SOEN-321-Adversarial-Attack-on-ML

1. Introduction

Intrusion Detection Systems (IDS) play a very important role in Cybersecurity because it identifies malicious attacks within networks. Machine language has emerged as a modern technology to help build IDS. Relying solely on manually created rules becomes inadequate as the size and complexity of contemporary networks increase. Rather, these models are now able to recognize attack patterns and adapt to new threats. The main goal of this project is to investigate how a machine learning model can categorize network traffic as either normal or attack, as well as how its behavior is affected when training data is purposefully altered.

For this project to train and evaluate the model we used the NSL-KDD dataset that is widely used in IDS research and is widely known for its thousand of network connections record that are described by 41 attributes (such as protocol type, services used, number of unsuccessful logins, packet statistics, etc.). We selected this dataset because it is realistic, organized, and created especially for cybersecurity study. Additionally, it has both numerical and categorical characteristics, which makes it a good candidate for researching adversarial manipulation, model performance, and preprocessing.

One most important characteristic of the dataset is class imbalance which depends on having more attack samples than regular ones. A model may be biased in favor of the majority class due to imbalanced data. To address this, we down sampled both classes to equal size in order to create a 50/50 balanced training set. This facilitates the learn fairly and be able to really observe the effects of data poisoning.

Our Machine Learning Model used the Deep Neural Network (DNN). This model can capture non-linear correlations between features and has a high recognition for working well with high-dimensional one-hot encoded data. We built the network with sigmoid output for binary classification, dropout for regularization, and dense layers with ReLU activation. The main aim is to train the model to recognize patterns that differentiate between normal and malicious traffic.

Contributions

- Developed a complete Intrusion Detection System (IDS) Deep Neural Network (DNN) machine learning model.
- Trained the model on the NSL-KDD dataset, which contains real-world-like network traffic connection behaviors as well as attack behaviors.
- Preprocessed data and handled categorical features, by applying One-Hot Encoding, and standardizing numerical features using Z-score normalisation.
- Balanced the dataset to a 50/50 normal and attack distribution, to address class imbalance and ensure fair model learning.
- Constructed a poisoned version of the dataset to simulate adversarial attacks using different label flipping attacks like attack hiding, DoS poisoning, and random label-flipping.
- Trained both a clean model and multiple poisoned models for comparison.
- Evaluated model performance using various evaluation metrics, including accuracy, confusion matrices, precision, recall, F1-score, and detailed error analysis.

Notable Findings

- Our clean DNN model performing well compared to the poisoned models, by achieving 86% in accuracy, and achieving better scores overall, across the multiple metrics used as measurement.
- Attack-hiding, where attack traffic is mislabeled as normal, produced the most damaging effect on the model. The model showed a large drop in recall and F1-score for the attack class, which shows that IDS models are vulnerable when malicious data is disguised as benign.
- DoS poisoning, where normal traffic is mislabelled as attacks, had a smaller impact on accuracy but dramatically increased false positives, making the model unreliable and less practical in real-world situations.
- Random label flipping method introduced some untargeted noise that affected the metrics but still caused less damage than the targeted attack-hiding.
- Overall, our results show that even small amounts of data poisoning can present a threat to IDS models, which are vulnerable to data poisoning attacks, specifically attack-hiding.

2. Methodology

In this section, we describe the dataset used in our intrusion detection experiment, the types of features it contains, the machine learning model trained on this data, and the attack performed on said model.

2.1. Dataset Overview:

The NSL-KDD dataset is a widely recognized benchmark dataset, used for the purpose of intrusion detection research (IDS). The data originates from the DARPA 1988 network intrusion evaluation program,

where network traffic was simulated and captured through a controlled environment developed for testing purposes (Bala and Nagpal, 2019).

This dataset contains a collection of network connections, with each record consisting of 43 columns:

- 41 features, describing the network connections,
- 1 label column, indicating whether a connection is *normal* or an attack. This allows to distinguish between benign connections and malicious ones.
- 1 difficulty score, representing how hard the instance is to classify (Kayode, 2025). This field is not relevant to our training.

The 41 features describe properties of the TCP/IP connection, they include basic information found in the TCP header fields, as well as statistical data measured on the traffic. The are grouped into 4 categories: Basic Features, Content Features, Time-Based Traffic Features, and Host-Based Traffic Features.

Table1: Feature Categories and Descriptions (Kayode, 2025)

Category	Description	Features
Basic Features	Extracted directly from the TCP/IP connection header, they allow to detect anomalies in connection setup, which useful for detecting port scan floods.	Duration, protocol, service, flag, src_bytes, dst_bytes, land, wrong_fragment, urgent
Content Features	Derived from the packet content, they capture behaviors used to reveal unauthorized access attempts.	hot, num_failed_logins, logged_in, num_compromised, root_shell, su_attempted, num_root, num_file_creations, num_shells, num_access_files, num_outbound_cmds, is_host_login, is_guest_login
Time-Based Traffic Features	Computed within a two-second time window these features will show if there's been a rapid connection, or one that's been repeated multiple times over the course of a short time.	count, srv_count, serror_rate, srv_serror_rate, rerror_rate, srv_rerror_rate, same_srv_rate, diff_srv_rate, srv_diff_host_rate
Host-Based Traffic Features	Computed over a window of the last 100 connections. These features will show if there's been coordinated attacks across multiple hosts.	dst_host_count, dst_srv_count, dst_host_same_srv_rate, dst_host_diff_srv_rate, dst_host_same_src_port_rate, dst_host_srv_diff_host_rate, dst_host_serror_rate, dst_host_srv_serror_rate, dst_host_rerror_rate, dst_host_srv_rerror_rate

The attacks in this dataset, fall into four categories: DoS, Probe R2L and U2R (Chadia E. L. Asry et al., 2024). Each of these attacks exhibits different behaviors which can be categorized by certain properties. For example, Neptune, a common DoS attack, is a SYN-flood where the attacker sends a large number of incomplete TCP handshakes. These types of attacks are distinguishable by their high *serror*_rate, and high *dst_host_serror*_rate (Marat Zhanikeev and Tanaka, 2007). A similar process is used to identify the multiple various attack types found on this dataset.

2.2. Preprocessing

Before training our model, several preprocessing steps were applied to ensure that the data was clean, consistent and properly formatted. This is essential when working with a large dataset such as NSL-KDD. To begin, the dataset was balanced so that it contained an equal proportion of attack and normal network connections (50%-50%). This data-balancing step was performed on both the training data and the testing sets.

The original training file, *KDDTrain+.txt,* contains approximately 125,000 samples, while the original testing file, *KDDTest+.txt,* contains about 22,500 samples. After balancing, the balanced datasets were loaded in *KDDTrain_balanced.txt,* which contained about 117260 samples, and *KDDTest+_balanced.txt*, which was reduced to 19,423 samples. The balanced datasets had less samples since they were balanced via reduction (reducing the number of samples the label with extra samples had randomly until it reached the number of samples matching the limiting label)

As mentioned in the previous section, each record contains a *label* field, indicating the connection type (attack or normal). This label serves as our reference, to evaluate our model's predictions. To tackle this problem, a binary classification task was performed, where all attack types were grouped under a single class:

- 0 – normal network connections,
- 1 – all malicious attack network connections.

One-Hot Encoding

Among the 41 features in the dataset, some are numerical, while others are categorical. The categorical features were indexed, by identifying them to their column number, and then transformed, using One-Hot Encoding. This is necessary to allow the data to be fed to the model. Since machine learning models requires numeric input variables, categorical fields must be transformed into binary vectors (Brownlee, 2020).

For a categorical variable x with the categories $C = \{c_1, c_2, \ldots, c_k\}$, the following vector will be produced:

$$\text{one_hot}(x) = [I(x = c_1), \ldots, I(x = c_k)]$$

where,

$I(x = c_i) = 1$ *if* $(x = c_i)$*, else 0.*

Z-Score (Standardization)

The remaining numerical data were standardized using Z-score normalization, which is the process of scaling values to have a mean of 0 and a standard deviation of 1. This avoids for disproportionate features that would impact the model's performance (Google, 2024).

$$z = \frac{x - \mu}{\sigma}$$

2.3. Training and Testing Setup

The clean data used for training and testing consists of the two balanced datasets described above. Both files contain clean, correctly labeled samples, which allowed to train the model to distinguish malicious network connections from normal ones.

To develop the model, the preprocessed training data was used to fit the neural network model, while the testing data was used to make predictions on unseen data, and evaluate the model based on multiple metrics, such as accuracy, precision, recall and confusion matrices.

Epoch in Training

An *epoch* is a critical concept in machine learning models; it's a measure of the number of times an algorithm will pass through the entirety of the training dataset. During the training, a number of epochs is set that the model will have to go through, in our case, 15 epochs were used. During each epoch, the model adjusts its weight based on the loss it calculated from the previous predictions that were made (Lyzr Team).

We initially trained our model for 50 epochs, however this caused severe overfitting, and we had to deal with long unnecessary computation time. After experimenting with 30, then 20 epochs, we observed that 15 epochs achieved the best balance between training efficiency and generalization.

Poisoning the Model

To study the effect of data poisoning, we created four manipulated versions of the balanced training set:

- *KDDTrain+_balanced_to_poisoned_attackhiding 10%.txt*
- *KDDTrain+_balanced_to_poisoned_attackhiding 5%.txt*
- *KDDTrain+_balanced_to_poisoned_DoS 10%.txt*
- *KDDTrain+_balanced_to_poisoned_randomflipping 10%.txt*

In total, three types of data-poisoning adversarial attacks were performed on the model:

- **Attack Hiding:** 5% of the total samples were flipped from labeled as *attack* to *normal,* then the same logic was applied on 10% of the set.
- **DoS Poisoning:** 10% of total dataset was flipped from being labeled as *normal* to *attack.*
- **Random Label Flipping:** 10% of all rows were randomly flipped between *attack* and *normal.*

Similarly to the clean model, the same testing data was interpreted by the 3 models that were trained on each of the 3 poisoned data sets to make predictions on network traffic data. The accuracy of these 3 poisoned models was then compared against the initial model trained on clean data.

2.4. Baseline Model

A Deep Neural Network (DNN) model was implemented as our baseline model. DNNs are suitable for high-dimensional tabular datasets such as NSL-KDD because they can model complex, non-linear feature interactions across many attributes. They are models that learn from experience and are able to adapt themselves based on the pattern that they're analyzing. The more hidden layers a model has, the better it'll be able to process and adapt itself to new data (Zemim, 2024).

Deep Neural Network Models

Deep Neural Networks consist of an input layer, multiple hidden layers, and an output layer. The inner, hidden layers are typically dense, containing what we call **neurons**, that will perform computations on the data that they're fed. Each neuron in a dense layer is connected to every neuron in the previous dense layer (Zemim, 2024).

They function by accepting input data through the input layer, where each neuron will assign weight to received raw data, this weight will serve as an indication of how relevant the data is to the task (Mercier, 2025).

Dense Layers

Dense layers will compute the operation:

output = activation(dot(input, kernel) + bias) (Thor, 2024)

where:

1- **input -** refers to the input layer of the model. This is the first layer where the shape of the model will be defined. Subsequently, the future layers' shapes will automatically follow that of the input layer.
2- **kernel -** refers to the weight of the matrix of the layer.
3- **dot (input, kernel) -** dot product of the input with the kernel.
4- **bias -** vector that is added to the result of the dot product. This allows to add flexibility to the model.

5- **activation -** function that is applied to the result of all the previously explained operations. The activation function that was used for our model is ReLU.

Dropout Layers

In addition to dense layers, the model contains dropout layers, as well. These layers will take the *rate* as an argument, representing the fraction of the input that should be dropped. For example, setting the rate to 0.1 means 10% of the neurons in this layer will be randomly dropped in each epoch. This serves as prevention for overfitting. The rest of the 90% of the neurons that weren't dropped, will be the ones trained on the input data (Yadav, 2022)

Activation Functions

ReLU

The Rectified Linear Unit activation function is defined as:

$$\text{ReLU}(x) = \max(0,x).$$

where, the output is the inputted value, unless it is a negative number, in which case 0 will be outputted.

This function has become the default activation function for hidden layers in the context of DNN models. This is because it's computationally efficient and easy to implement, in comparison to other functions, and it helps mitigate the vanishing gradient problem in neural networks, which in turn allows for the model to be more efficient by learning from the backpropagation process (Thor, "Activation Functions") (Nair and Hinton, 2010)

Sigmoid

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

The sigmoid function is used for the output later in binary classification tasks, producing a value between 0 and 1, which can later be interpreted as a probability (Thor, "Activation Functions")

Architecture

Our model possesses the following architecture:

- **Input Layer –** accepts the processed vector after standardization and one-hot encoding. It defines the shape of the model.
- **Dense Layer (128 units, ReLU) –** extracts high-level feature interactions important for classification.
  This layer has 128 neurons and uses ReLU as an activation function.
- **Dropout Layer (0.3) –** randomly deactivates 30% of the neurons on that layer in order to prevent overfitting.
- **Dense Layer (64 units, ReLU)**
- **Dropout Layer (0.3)**
- **Output Layer (1 unit, sigmoid) –** contains a single neuron and uses Sigmoid as an activation function.

Why This Architecture Was Chosen

A deep neural network model was chosen because the NSL-KDD dataset is a high tabular set, with 41 attributes. DNNs are models known to perform well with a large feature space due to their multiple layers and depth neurons used for processing a wide range of data. Unlike classical models, like Decision trees

for example, DNNs scale better with thousands of samples ("Deep Neural Network - an Overview | ScienceDirect Topics")

Many of the records in the dataset are categorical, such as service types and protocols, which ReLU-based networks can model effectively.

In addition, DNNs are widely used in IDS research, they're good for learning complex attack pattens and they show robustness to noise and adapt to training changes, which also correlates with the choice of our attack, data-poisoning, where data change is the viable part of it. Hence, why it was interesting to choose DNN to study data-poisoning on.

2.5 Data Poisoning

Data poisoning is a genre of adversarial attack where an attacker intentionally changes data used to train a machine learning model. It works by either inserting new training samples or modifying existing ones to corrupt them so that the model internalizes false patterns. When executed successfully, the poisoned models are expected to show degraded accuracy, unpredictable behaviour, or controlled misbehaviour that can benefit the attacker. By using label flipping to change the correct labels of training samples to incorrect labels, we were able to simulate the effects of attack hiding, DoS, and random flipping attacks.

Attack Hiding

Attack hiding refers to poisoned data that looks clean, to the point where a standard inspection of the data looks properly labeled, giving the model and anyone looking at it the illusion that altered data is correct. To imitate this, 10% of the network traffic that had the 'attack' flag was modified to have the 'normal' flag before training our model.

DoS Poisoning

DoS poisoning parallels the concept of a regular DoS attack by applying it to the training data of a ML model. An attacker using DoS poisoning would aim to break the overall model's performance by injecting large amounts of misleading data. To simulate DoS poisoning, we trained one of the models on a data set where 10% of the data's *label* variable from the initial data was changed from 'normal' to 'attack'.

Random Label-Flipping

Random flipping is an attack where the labels of data from legitimate samples are flipped. This was replicated by swapping the labels of 10% of the training data set from 'normal' to 'attack' and from 'attack' to 'normal' and then training a ML model on this data.

2.6. Evaluation Metrics

To evaluate intrusion detection system (IDS) properly, it's not sufficient to only rely on accuracy. Although accuracy provides an overview of how well the model performs, IDS systems require a more in dept analysis that encapsulate false negatives, false positives, and F1-score. These metrics ae what will allow us to determine whether the system would perform practically in a realistic environment.

The NSL-KDD dataset, in particular, is challenging due to its many characteristics and the multiple data it contains on various attacks, with some of them having overlapping properties. Certain attacks are harder to detect, because of their subtleness. In addition, the training set consists of data that presents clearer patterns, making it easier. In contrast, the testing set is intentionally harder and contains never-before seen cases that the model will have to predict. This allows for a far more realistic environment where the model will have to deal with tricky samples. Because of these difficulties, multiple metrics are essential to characterize the true performance of the model.

Accuracy – measures the proportion of all the correctly classified samples.

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN}$$

Attack Recall (True Positive Rate) **–** shows the proportion of attacks that were successfully detected.

$$\text{Recall} = \frac{TP}{TP+FN}$$

This is one of the most important metrics because missed attacks can be critical, in the context of an IDS, where security should be robust.

High recall shows that the model is good at catching malicious activities.

Attack Precision – shows the proportion of predicted attacks that were correctly classified as attacks.

$$\text{Precision} = \frac{TP}{TP+FP}$$

Low precision means that the IDS is generating many false alarms, this isn't ideal since it makes the model unreliable and wastes resources.

False Negatives – shows how many attacks were missed.

This metric is also a notable measure because it could be devastating for a security system to bypass attacks. This would show that the IDS is vulnerable, and prone to label-flipping.

False Positives (False Alarms) **–** shows the proportion of benign traffic that was wrongly flagged as malicious.

False alarms can be devastating to an IDS. They could result in unnecessary blocking, wasted resources in analyzing non-malicious traffic instead of focussing on attacks, and the lack of robustness making the system unreliable.

F1-score – mean measure of precision and recall.

$$\text{F1} = 2 \cdot \frac{Precision \cdot Recall}{Precision+Recall}$$

F1-scores provide a balanced view of how well the model is able to perform detections.

A high F1-score means that it has high recall and high precision, meaning when the model detects attacks, they're not false alarms and they're correctly classified.

A model can have high precision but low recall, which would mean that it barely raises false alarms, but it does miss many attacks. Or, it can also have high recall but low precision, meaning it catches most attacks, but it produces a lot of false alarms.

Confusion Matric and Interpretation

To evaluate all the above metrics, a matrix is generated to summarize the different results. The matrix has the following form:

	Predicted *Normal*	**Predicted *Attack***
Actual *Normal*	True Negative (TN)	False Positive (FP)
Actual *Attack*	False Negative (FN)	True Positive (TP)

where,

- **TN**: benign traffic
- **TP**: attacks that were correctly detected

- **FP**: false alarms
- **FN**: missed attacks

The confusion matrix allows to easily visualize and analyze how different label-flipping data-poisoning attacks will affect the model and whether predictions are correctly classified.

3. Results

3.1 Types of attacks

The table below illustrates the association between the types of poisoning applied and declines in accuracy and performance of the model.

Table2: Clean Model vs Poisoned Model result comparisons (all types)

Metric	Clean	Attack-Hiding (10%)	DoS Poisoning (10%)	Random Label Flip (10%)
Accuracy	86.0%	81.9%	85%	84%
Attack Recall	79.1%	71.7%	78%	77%
Attack Precision	91.2%	90.2%	91%	90%
F1-score	84.72%	79.89%	84.00%	82.99%

All these attacks consist of flipping dataset labels.
Attack-Hiding: labels normal packet as attack packet
DoS: labels attack packet as normal packet

Our preliminary clean model categorizes the data with an accuracy of 86% and attack precision of 91.2%. When applying *Attack Hiding*, we can see a significant increase in missed attacks (726 more) and an overall drop in accuracy (by 4%). Meanwhile, when applying *DoS Poisoning* and *Random Label Flip*, we see less drops in accuracy (1% for DoS Poisoning and 2% for Random Label Flip). However, DoS Poisoning results in the highest number of false alarms (45 more). Why does this happen, and how is it possible that different types of poisoning affect different properties of the model?

3.2. Attack Hiding

Attack Hiding manipulates malicious samples, so they look more like benign samples. This is the only attack specifically designed to reduce the model's ability to catch attacks, which is why the model misses them, being the cause of low recall and low accuracy. From these results, we can conclude that Attack Hiding is the most effective method for degrading model performance when the attacker's goal is to undermine detection accuracy. This is shown in the above table, as it is the one that causes the model to produce the lowest overall accuracy and the highest number of missed attacks. To understand this effect more deeply, we further analyzed the impact of different intensities of Attack Hiding. The table below shows the effects of 5% Attack Hiding vs. 10%.

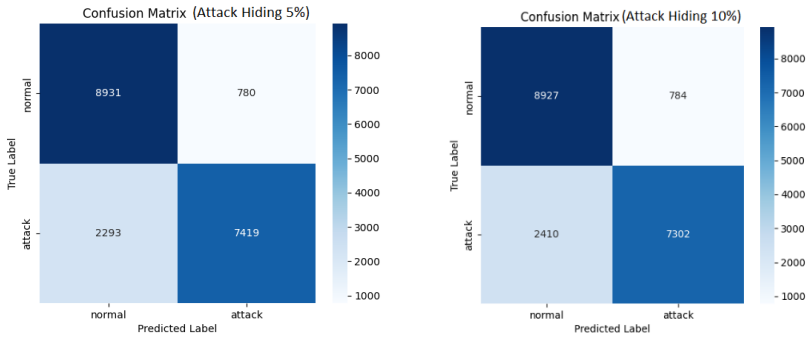
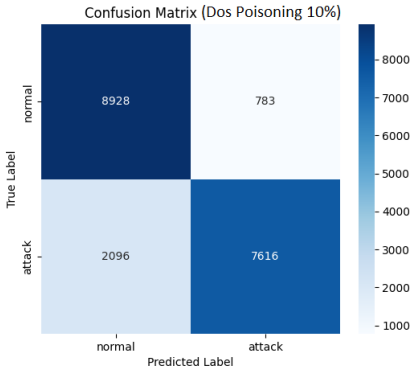


Table3: Clean Model vs (Attack hiding 5% 10%) result comparisons

Metric	Clean	Attack-Hiding (5%)	Attack-Hiding (10%)
Accuracy	86.0%	84%	81.9%
Attack Recall	79.1%	75%	71.7%
Attack Precision	91.2%	91%	90.2%
F1-score	84.72%	82.23 %	79.89%

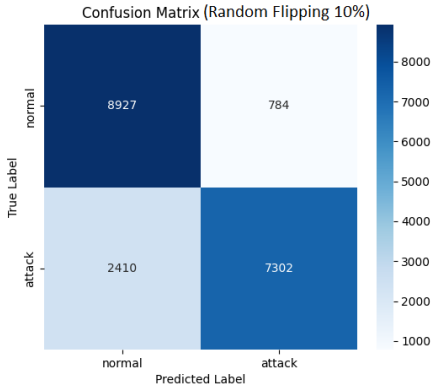
3.3. DoS Poisoning

DoS poisoning adds mislabeled data which makes the model overly sensitive to new data. This is what leads to having the highest number of false positives. The model over flags everything, but still catches most attacks, so precision and accuracy don't collapse. Our model appeared to have evolved into a conservative model, which is why DoS poisoning had no impact; it needed substantial evidence to classify an event as an attack, hence preserving the services of legitimate traffic despite the compromised training data. In short, DoS poisoning pushes the model toward an over-defensive stance and raising false alarms but still maintaining attack detection performance.



3.4. Random Label Flip

Random Label Flip introduces noise to the data. Some attack data looks harmless; some harmless looks malicious. Furthermore, the attack is not targeted. This is why the performance of the model drops a bit everywhere but not as dramatically compared to Attack Hiding. Because the noise is random and not adversarially optimized, the model still sees the correct underlying distribution. In our case only a small fraction (10%) is corrupted. The remaining 90% remains accurate, so the model still learns the true patterns and does not get pushed toward any specific wrong boundary. In a targeted attack (like Attack Hiding), the poisoned samples push the decision boundary toward a specific wrong region. RLF pushes the boundary in all directions at once, causing net movement to be small.



3.5. Summary

Overall, the three different types of attack methods each impact the model differently because everyone disrupts a different part of the learning process. Attack Hiding directly interferes with the model's ability to distinguish attacks from healthy data. DoS Poisoning mostly affects the model's sensitivity, causing it to create more false alarms. Random Label Flip introduces untargeted noise, which slightly lowers performance across all metrics but does not significantly distort the model's decision boundary. These differences highlight that the effect of poisoning depends not only on the amount of corruption, but on how the corrupted data influences the patterns the model learns.

4. Conclusion

To conclude, in this research we evaluated label flipping poisoning methods against a neural-network IDS trained and tested on NSL KDD benchmark datasets. To ensure that the results and model were unbiased, both the training and the testing data sets were balanced to contain 50/50 between normal traffic and "attack"/harmful traffic. The training balancing was to improve our model's detection capabilities, and the test data balance was to prevent any bias caused by a model possibly selecting 1 answer and getting a higher than 50% accuracy. While the NSL KDD dataset originally had many diverse options for the attack traffic, we opted to train the model on a binary "attack" or "normal" traffic labels by simply reducing all attack types down to the label "attack". Our experiments were on checking how three types of label flipping poisoning (attack-hiding, DOS, and random flipping) of the training dataset affected the model's performance.

The results of our experiments showed favor for attack-hiding as the most effective poisoning method. The result was more than a 4% decrease of accuracy and more than an additional 700 missed attacks by the IDS. Contrasting that, the DOS method showed a lack of demonstrable effect causing only 1% decrease of accuracy and 45 false alarms. Our inference was that this model developed a conservative approach to declaring attacks due to the originally large variety that were turned to be labeled as "attack" in the training dataset. Due to that conservative approach, only few additional normal traffic was flagged as harmful. And finally, random flipping which was just a method to introduce noise into the training dataset did demonstrate an effect on the model but not as noticeable as attack hiding (2% accuracy loss and more false alarms and missed attacks overall)

Overall, these research tests showed that even a small amount of training data poisoning 5-10% via label flipping, as our example, can have a noticeable effect on the performance of an IDS especially when focused

on hiding harmful traffic. This shows the importance of data integrity when developing ML based IDSs and shows even more the additional importance for focus on information systems security development.

5. References

Bala, Ritu, and Ritu Nagpal. “A REVIEW on KDD CUP99 and NSL-KDD DATASET - ProQuest.” *Proquest.com*, 2019, www.proquest.com/openview/ef14e2e71dbbe9149043b33020b30c18/1?pq-origsite=gscholar&cbl=1606379. Accessed 28 Nov. 2025.

Kayode, Bisola Faith. “Temporal-Spatial Attention Network (TSAN) for DoS Attack Detection in Network Traffic.” *Arxiv.org*, 21 Mar. 2025, arxiv.org/html/2503.16047v2. Accessed 28 Nov. 2025.

Chadia E. L. Asry, et al. “Attacks Categories in the NSLKDD Dataset.” *Figshare*, 24 Jan. 2024, plos.figshare.com/articles/dataset/Attacks_categories_in_the_NSLKDD_dataset_/25056703?file=44215804, https://doi.org/10.1371/u002Fjournal.pone.0295801.t003. Accessed 28 Nov. 2025.

Marat Zhanikeev, and Yoshiaki Tanaka. “Detection and Identification of Neptune Attacks and Flash Crowds.” *Lecture Notes in Computer Science*, vol. 4773, 17 Sept. 2007, pp. 571–574, https://doi.org/10.1007/978-3-540-75476-3_68.

Brownlee, Jason. “Ordinal and One-Hot Encodings for Categorical Data.” *Machine Learning Mastery*, 11 June 2020, machinelearningmastery.com/one-hot-encoding-for-categorical-data/.

Google. “Numerical Data: Normalization.” *Google for Developers*, 2024, developers.google.com/machine-learning/crash-course/numerical-data/normalization.

Lyzr Team. “What Are Epochs in Machine Learning?” *Lyzr*, 26 Nov. 2024, www.lyzr.ai/glossaries/epochs/. Accessed 28 Nov. 2025.

ZEMIM. “Deep Neural Network (DNN) Explained - ZEMIM - Medium.” *Medium*, 14 Apr. 2024, medium.com/@zemim/deep-neural-network-dnn-explained-0f7311a0e869. Accessed 28 Nov. 2025.

Mercier, Marc. “What Is a Deep Neural Network? Learn More about Deep Neural Networks and How Deep Learning Works | Botpress Blog.” *Botpress.com*, 14 Jan. 2025, botpress.com/blog/deep-neural-network. Accessed 28 Nov. 2025.

Thor, Wei-Ming. “Common Layer Types: Dense.” *Apxml.com*, ApX Machine Learning, 2024, apxml.com/courses/deep-learning-fundamentals-keras/chapter-2-building-networks-keras/dense-layers. Accessed 28 Nov. 2025.

Yadav, Harsh. “Dropout in Neural Networks | towards Data Science.” *Towards Data Science*, 5 July 2022, towardsdatascience.com/dropout-in-neural-networks-47a162d621d9/. Accessed 28 Nov. 2025.

Thor, Wei-Ming. “Activation Functions.” *Apxml.com*, ApX Machine Learning, 2016, apxml.com/courses/deep-learning-fundamentals-keras/chapter-2-building-networks-keras/activation-functions. Accessed 28 Nov. 2025.

Nair, Vinod, and Geoffrey Hinton. *Rectified Linear Units Improve Restricted Boltzmann Machines*. 2010.

“Deep Neural Network - an Overview | ScienceDirect Topics.” *Www.sciencedirect.com*, www.sciencedirect.com/topics/computer-science/deep-neural-network. Accessed 28 Nov. 2025.