Information Systems Security

SOEN 321 Section GG

Final Project: Attacks on Machine Learning

https://github.com/Guillaume-Lachapelle/SOEN321-Project

By:

Ann-Marie Czuboka (40209452)
Isabelle Czuboka (40209525)
Guillaume Lachapelle (40203325)
Oliver Vilney (40214948)
Nicholas Piperni (40207764)
Valeria Dolgaliova (40212218)
Tanzir Hoque (40210275)

Presented To:

Dr. Amr Youssef

Department of Computer Science & Software Engineering

Concordia University

Montreal, Canada

December 2nd, 2024

Abstract

Machine learning has become an important role in many domains today like cybersecurity by detecting and preventing potential threats. But, machine learning (ML) is not an impenetrable wall and can be easily exploited leading to failures that can be incredibly harmful. As ML is foundationally based on its datasets, small changes in this data can impact the accuracy guiding the model to wrong predictions. This project will investigate four types of attacks: Adversarial Attacks, Membership Interference Attacks, Trojan/Backdoor Attacks and Denial-of-Service Attacks on a model trained with the NSL-KDD dataset, found on Kaggle [1]. The methodology includes processing the dataset, training a baseline model, applying attack techniques and gathering data on its effects on the model.

For each 4 of the attacks, the results showed significant reduction in accuracy between 20 and 45% or a system crash. These results prove the vulnerability of AI models and the strength of these attacks.

Our findings highlight the challenges of keeping a secure and reliable ML system. There is a crucial need for solid defense mechanisms, such as adversarial training and the detection of anomalies, to improve model security. This report aids in understanding the implications of these vulnerabilities and offers insights into developing ML systems capable of withstanding various types of attacks.

Table of Contents

Introduction

The rapid advancement of Machine Learning (ML) and Deep Learning (DL) has transformed the field of cybersecurity by providing a wide range of tools to detect and mitigate threats and other malicious activities. Despite these models being immensely powerful, they are prone to exploitation. Adversaries can exploit vulnerabilities in ML systems, which can lead to failures and reduced performance.

Interesting results were found in each attack. For the Adversarial Attack, we notice a reduction in accuracy from 100% to 52.41%. For the Membership Inference Attack, we observe a reduction in accuracy from 100% to 79.58%. For the Trojan/Backdoor Attack we get a reduction in accuracy from 100% to 70.07%. The final attack, the denial-of-service attack, started with an accuracy of 61.2% and the result of an attack is a system crash. All of these results are what we expected to see, either a reduction of accuracy or a system failure.

Methodology

Dataset Overview

The NSL-KDD dataset is widely used for network-based intrusion detection systems. It contains a diverse range of network traffic patterns, including normal and abnormal behaviour labelled in the dataset. This dataset is an improvement from its initial attempt in 1999, by reducing multiple types of abnormal behaviours into 1 type [2]. The dataset consists of many data rows, a few dozen thousand for the training and testing data each, making it large and complex. Every item has its last attribute class labeling it as ‘normal’ or an ‘anomaly’. Furthermore, each data row is represented by 41 features which include duration, protocol_type, service, src_bytes, among others. Although the dataset has been extensively used to train and evaluate intrusion detection models, its susceptibility to attacks has not been evaluated thoroughly.

In this report, we investigate four common types of attacks on a ML model trained with the NSL-KDD dataset:

- **Adversarial Attacks:** These attacks involve making subtle modifications to input data that are often imperceptible to humans but can mislead models into making incorrect predictions [3].
- **Membership Inference Attacks:** These attacks aim to exploit the outputs of ML models to determine whether specific data points were part of the training dataset. By observing model confidence scores or prediction patterns, an attacker can infer the inclusion of sensitive data in the training process [4].
- **Trojan/Backdoor Attacks:** These attacks introduce hidden triggers in the training data, which when activated, cause the model to behave maliciously [5].
- **Denial-of-Service (DoS) Attacks**: DoS attacks aim to overwhelm the machine learning system with excessive or malformed inputs, which renders it unable to function properly [6].

Before performing any attacks on the system, a few steps are required to prepare the dataset.

The first step is to select a dataset worthy of exploration. This has been chosen as the NSL-KDD dataset from Kaggle. It has been used for many projects already which can be observed in the 'Code' section in Kaggle. The dataset has also been improved over the years so it is an ideal dataset to perform attacks.

The second step is to build the model. We have done this in the following way for the Adversarial attack, Trojan/Backdoor attack, and Membership Inference attack:

```
# Train the model
# [11] https://www.tensorflow.org/guide/keras/training_with_built_in_methods
def train_model(X_train, y_train):
    model = Sequential([
        Dense(64, activation='relu', input_dim=X_train.shape[1]),
        Dense(32, activation='relu'),
        Dense(1, activation='sigmoid')  # Binary classification
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    model.fit(X_train, y_train, epochs=5, batch_size=64, validation_split=0.1, shuffle=False)
    return model
```

For the DoS attack, we have set up the model the following way:

```
class flaskNN(nn.Module):
    def __init__(self):
        super(flaskNN, self).__init__()
        self.fc1 = nn.Linear(X_train.shape[1], 128)
        self.relu = nn.ReLU()
        self.dp = nn.Dropout(p=0.5)
        self.bn1 = nn.BatchNorm1d(128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, len(torch.unique(y_train)))

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.bn1(x)
        x = self.dp(x)

        x = self.fc2(x)
        x = self.relu(x)

        x = self.fc3(x)
        return x
```

The third step is to gather preliminary information on the model. Like its feature types, ground truth column and model's accuracy. We have done this in the following way.

On Kaggle [1], the information for the different feature types are displayed, informing the team of which columns need encoding, for example prototype_type, service and flag need encoding.

```
@relation 'KDDTest'
@attribute 'duration' real
@attribute 'protocol_type' {'tcp','udp', 'icmp'}
@attribute 'service' {'aol', 'auth', 'bgp', 'courier', 'csnet_ns', 'ctf', 'daytime', 'discard', 'domain',
@attribute 'flag' { 'OTH', 'REJ', 'RSTO', 'RSTOS0', 'RSTR', 'S0', 'S1', 'S2', 'S3', 'SF', 'SH' }
@attribute 'src_bytes' real
@attribute 'dst_bytes' real
@attribute 'land' {'0', '1'}
@attribute 'wrong_fragment' real
@attribute 'urgent' real
@attribute 'hot' real
@attribute 'num_failed_logins' real
@attribute 'logged_in' {'0', '1'}
@attribute 'num_compromised' real
@attribute 'root_shell' real
@attribute 'su_attempted' real
@attribute 'num_root' real
@attribute 'num_file_creations' real
@attribute 'num_shells' real
@attribute 'num_access_files' real
@attribute 'num_outbound_cmds' real
@attribute 'is_host_login' {'0', '1'}
@attribute 'is_guest_login' {'0', '1'}
@attribute 'count' real
@attribute 'srv_count' real
@attribute 'serror_rate' real
@attribute 'srv_serror_rate' real
@attribute 'rerror_rate' real
@attribute 'srv_rerror_rate' real
@attribute 'same_srv_rate' real
@attribute 'diff_srv_rate' real
@attribute 'srv_diff_host_rate' real
@attribute 'dst_host_count' real
@attribute 'dst_host_srv_count' real
@attribute 'dst_host_same_srv_rate' real
```

Below is how the model's accuracy is calculated

```
# Evaluate the model
_, test_accuracy = model.evaluate(X_test, y_test) # loss and metrics being returned
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

The fourth step is to perform the attacks on the datasets and gather its effect on the model. We have performed the following attacks below. In each category, we will describe how these attacks were performed and the data we gathered from the modified models.

Adversarial Attack

An easy example of an adversarial attack is changing the detection of a stop sign for a speed limit sign. You can see how this can cause major accidents in self-driving cars. Here is how we applied this kind of attack to our dataset.

A neural network was constructed using the TensorFlow Keras API to classify the dataset. The model was designed as follows:

Architecture:

- Input layer: A dense layer with 64 units and ReLU activation.
- Hidden layer: A dense layer with 32 units and ReLU activation.
- Output layer: A dense layer with sigmoid activation for binary classification.

Hyperparameters:

- Optimizer: Adam
- Loss Function: Binary Cross-Entropy
- Epochs: 5
- Batch size: 64
- Validation Split: 20% of the training data. The model was trained on the preprocessed training dataset, and its initial test accuracy was observed to be 100%.

The baseline model was evaluated on the test dataset to gather its initial performance metrics. The accuracy achieved was 100%, indicating the model was able to effectively perform on its test data demonstrating that a neural network model works very well with this dataset.

The following steps were taken to execute the attack:

- Noise was injected into specific numeric feature columns on a copy of the training data to mislead the model.
- 50% of the labels in the newly added data were flipped to simulate a targeted attack.
- The manipulated data was joined with the original training data to train a new model, referred to as the attack model.
- The attack model's performance on the test dataset was evaluated, achieving an accuracy of 52.41%.

Trojan/Backdoor Attack

An example of a Trojan backdoor attack is when there is a specific pattern or value in the input features which consistently causes misclassification. Here is how we applied this type of attack to our dataset:

The architecture and the hyper parameters were the same as the adversarial attack. It has the exact same model as the adversarial attack so we know that the model itself is efficient. The following steps were done to execute the attack:

- 5% of the training data was assigned a trigger to inject into the model.
- The model was trained with the altered data
- The model was evaluated with unaltered test data and has a accuracy of 100%
- The model was evaluated with test data where 30% of the rows had the trigger
- The accuracy was only 70.07% .

The trigger data was mislabeled while the rest was not.

Membership Inference Attack

Membership Inference Attack is all about detecting the data, preferably the sensitive data in the dataset violating privacy and security. The goal is to identify whether data was part of the original training dataset. Here is how we applied this type of attack to our dataset.

The architecture and the hyper parameters were the same as the adversarial attack and trojan/backdoor attacks and uses the same base model for the KDD dataset. The following steps were done to execute the attack:

- Generated confidence scores for the training and testing data.
- Created shadow dataset for membership inference.
- Training a membership inference attack model using the shadow datasets.
- Evaluate the accuracy of the attack model.

By using the confidence scores of the original training and testing data for the shadow datasets, the attack model is able to predict whether new samples were part of the original model's training set based solely on their confidence scores. The goal is to leverage the differences in how the original model behaves with the original training data compared to new data to infer membership.

Denial-of-Service Attack

An example of DoS attack is excessive traffic, like purposefully inquiring the system multiple times. Here is how we applied this type of attack to our dataset:

We created a simple neural network model that was trained with the NSL-KDD dataset. The model was slightly different from the other models, as this one required to be part of a server that is run locally through Flask. The attack consisted of having a single thread host the server and the neural network, while having several other threads send a large volume of requests for the same prediction.

How it affected the model: For low counts (1000 requests), the model is significantly slowed down but manages to evaluate all 1000 requests. For high counts (5000), the model gets overwhelmed and crashes, causing subsequent requests to cause exceptions.

Results

After conducting four different types of attacks on the NSL-KDD model, we have summarized our findings in the below table.

Attack Type	Accuracy Before Attack	Accuracy After Attack	Observations
Adversarial Attack	100%	52.41%	The model was fooled into misclassifying manipulated inputs.
Membership Interference	100%	79.58%	The new model determined which data belonged to the original training dataset.
Trojan/Backdoor Attack	100%	100% (without trigger) 70.06% (with trigger)	The data that had the trigger was mislabeled.
Denial-of-Service	61.2%	N/A	The model gets overwhelmed with requests and crashes.

Table 1: Summary of Findings

For the adversarial attack, the team conducted evaluations based on the percentage of flipped labels and observed an interesting trend. We tested the model with 25%, 35%, 40%, 45%, and 50% flipped labels in the newly added data, which is added to the original data, resulting in accuracy scores of 99.97%, 99.76%, 99.68%, 98.48%, and 52.41%, respectively. In other words, modifying 25%, 35%, 40%, 45%, and 50% of the flipped labels in the newly added data will be equivalent to modifying 12.5%, 17.5%, 20%, 22.5% and 25% of the total dataset used to evaluate the model. These results show that the accuracy only drops significantly when 50% of the added data's labels are modified/poisoned, otherwise, the impact is minimal. This suggests that the model maintains strong performance unless at least 25% of the dataset is affected. It was also observed that running the model with only the added injected data and no flipped labels results in an unchanged accuracy. Therefore, it can be concluded that modifying the ground truth column (referred to as the `class` column in Kaggle) is essential in attacking the model successfully.

Conclusion

These results teach us that Machine Learning models are still very much susceptible to hackers and dangerously wrong predictions. However, there are many strategies to prevent these negative behaviors. Some of these include Adversarial Learning which is to train the model with altered data so that the model learns to correct these predictions. An example to prevent data poisoning is Audit Trails which leave records of all changes in the dataset to better track down malicious activity [7]. Also, Access Control is a procedure which prevents unwanted users access to sensitive data [7]. Moreover, to defend against trojan attacks one could ensure the security and cleaness of the entire data pipeline [18]. Many more preventional actions can be explored.

References in Report

[1] M. H. Zaib, "NSL-KDD," *Kaggle*, Apr. 25, 2019. [Online]. Available: https://www.kaggle.com/datasets/hassan06/nslkdd/data. (accessed Nov. 24, 2024)

[2] "NSL-KDD-Data_Analysis-and-Modeling," *GitHub*, Data-Profiler, 2022. [Online]. Available: https://github.com/Data-Profiler/NSL-KDD-Data-Analysis-and-Modeling. (accessed Nov. 24, 2024)

[3] Sciforce, "Adversarial attacks explained (and how to defend ML models against them)," *Medium*, Sep. 7, 2022. [Online]. Available: https://medium.com/sciforce/adversarial-attacks-explained-and-how-to-defend-ml-models-against-them-d76f7d013b18.(accessed Nov. 24, 2024)

[4] *TechTalks*, "Machine learning: What are membership inference attacks?" Apr. 23, 2021. [Online]. Available: https://bdtechtalks.com/2021/04/23/machine-learning-membership-inference-attacks/. (accessed Nov. 24, 2024)

[5] Liu, Y. (n.d.). *Trojan attacks and defenses on Deep Neural Networks*. Purdue e-Pubs. [Online]. Available: https://docs.lib.purdue.edu/dissertations/AAI30506242/#:~:text=We%20call%20it%20trojan%20attack,special%20pattern%20called%20trojan%20trigger. (accessed Nov. 17, 2024)

[6] *What is a denial of service (dos) attack?*. Palo Alto Networks. 2024. [Online]. Available: https://www.paloaltonetworks.ca/cyberpedia/what-is-a-denial-of-service-attack-dos (accessed Nov. 17, 2024)

[7] Oleszak, M. (2024, September 27). *Adversarial machine learning: Defense strategies*. neptune.ai. [Online]. Available: https://neptune.ai/blog/adversarial-machine-learning-defense-strategies#:~:text=Different%20attack%20types%20exist%2C%20including,improve%20robustness%20against%20adversarial%20attacks. (accessed Nov. 17, 2024)

References in Code

Adversarial Attack

[8] “Weights & Biases,” W&B, 2024. [Online]. Available: https://wandb.ai/sauravmaheshkar/RSNA-MICCAI/reports/How-to-Set-Random-Seeds-in-PyTorch-and-Tensorflow--VmlldzoxMDA2MDQy (accessed Nov. 29, 2024).

[9] “random.seed( ) in Python,” GeeksforGeeks, May 02, 2019. [Online]. Available: https://www.geeksforgeeks.org/random-seed-in-python/ (accessed Nov. 29, 2024).

[10] “LabelEncoder,” scikit-learn, 2024. [Online]. Available: https://scikit-learn.org/dev/modules/generated/sklearn.preprocessing.LabelEncoder.html (accessed Nov. 29, 2024).

[11] “Training & evaluation with the built-in methods | TensorFlow Core,” TensorFlow. [Online]. Available: https://www.tensorflow.org/guide/keras/training_with_built_in_methods (accessed Nov. 29, 2024).

[12] “Normal (Gaussian) Distribution,” www.w3schools.com. [Online]. Available: https://www.w3schools.com/python/numpy/numpy_random_normal.asp (accessed Nov. 29, 2024)

[13] “NumPy Joining Array,” www.w3schools.com. [Online]. Available: https://www.w3schools.com/python/numpy/numpy_array_join.asp (accessed Nov. 29, 2024).

[14] GeeksforGeeks, “Fixing Accuracy Score ValueError: Can’t Handle Mix of Binary and Continuous Target,” GeeksforGeeks, Jul. 24, 2024. [Online]. Available: https://www.geeksforgeeks.org/fixing-accuracy-score-valueerror-cant-handle-mix-of-binary-and-continuous-target/ (accessed Nov. 29, 2024).

DoS Attack

[15] “How To Make a Web Application Using Flask in Python 3 | DigitalOcean.” Accessed: Nov. 29, 2024. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3

[16] V. Durvasula, “Sending JSON Data to a Flask Using Requests in Python - AskPython.” Accessed: Nov. 29, 2024. [Online]. Available: https://www.askpython.com/python-modules/flask/semd-json-data-flask-app

[17] “How to use ThreadPoolExecutor in Python3 ?,” GeeksforGeeks. Accessed: Nov. 29, 2024. [Online]. Available: https://www.geeksforgeeks.org/how-to-use-threadpoolexecutor-in-python3/

Trojan Backdoor Attack

[18] “What is trojai AI,” What Is TrojAI - TrojAI 1.0.0 documentation, https://pages.nist.gov/trojai/docs/about.html (accessed Nov. 29, 2024).

Membership Inference Attack

[19] YouTube, https://youtu.be/rDm1n2gceJY (accessed Nov. 29, 2024).

[20] B. Dickson, “Machine learning: What are membership inference attacks?,” TechTalks, https://bdtechtalks.com/2021/04/23/machine-learning-membership-inference-attacks/ (accessed Nov. 29, 2024).

[21] Dr. E. Lee, “AI security: Membership Inference attacks and mitigating them with differential privacy with code...,” Medium, https://drlee.io/ai-security-membership-inference-attacks-and-mitigating-them-with-differential-privacy-with-code-78bf3f7af5d8 (accessed Nov. 29, 2024).

[22] P. Irolla, “Demystifying the membership inference attack,” Medium, https://medium.com/disaitek/demystifying-the-membership-inference-attack-e33e510a0c39 (accessed Nov. 29, 2024).