Alenna Zweiback
Project #3

Introduction and Background

In this project, I built a neural network-based network intrusion detection system (NIDS) that classifies network traffic as either benign or malicious for integrity attacks. I also tested how well the system could handle adversarially perturbed inputs. Perturbed inputs are traffic that had been slightly modified to try and evade detection. My goal was to create a baseline model, try out different defense strategies, and then evaluate how well each one worked. This report will outline my process and attempts for doing Task 1 and Task 2.

Exploring the Data

The dataset I used had labeled network traffic, including categories like Benign, Web Attack, DDoS, Infiltration, Port Scan, DoS, and Brute Force. I started by dropping any missing values and cleaning the string labels in the last column. Then I converted everything into a binary classification problem where 0 = benign and 1 = malicious. Since the dataset was already balanced, I did not need to oversample. I did not need to encode since I had converted my label column into binary and was not using categorical variables. Even though the data was normalized, I standardized all the feature values using StandardScaler so my neural network could learn more effectively and converge faster.

Architecture and Design Choices

For my baseline model, I used an MLPClassifier with three hidden layers containing 128, 64, and 32 neurons. I used ReLU activation, the Adam optimizer, and adaptive learning rate to help the model adjust as it trained. I set max_iter=300 to give it enough time to learn the patterns in the data and added a random state for reproducibility. By creating this baseline mode, it tested how it would perform on clean data before testing my new testing defenses.

For Task 2, I tried two different approaches. First, I built an adversarially trained MLP with two hidden layers (64 and 32 neurons) and ReLU activation. I retrained this model using a mix of clean and perturbed malicious samples. Second, I built an ensemble model by combining my baseline MLP with a Random Forest classifier (with 100 estimators). I used soft voting to average the prediction probabilities from both models. My thought process was that the neural network could handle complex patterns, and the Random Forest could help with robustness and stability and build upon those strengths to correctly detect malicious samples.

Attack Methods(s) Used

For the adversarial training approach in Task 2, I simulated and augmented the training data to help the model learn how to handle evasion attacks. I started by splitting the dataset into training and validation sets. Then I filtered out just the malicious samples from the training set and applied a noise generator that slightly changed the feature values. This noise was directional and controlled using an epsilon value which allowed me to adjust the strength of the perturbation. I tested three different epsilon values: 0.01, 0.05, and 0.1. These perturbed malicious samples were then combined with the clean data to retrain the MLP. I also used the same noise generator during testing to evaluate how well the model could handle adversarial inputs it had not seen during training.

For the ensemble method, I took a different approach. I did not retrain the models using any adversarial data. Instead, I combined my baseline MLP and a Random Forest classifier using soft voting. This means I averaged the predicted probabilities from both models and used that to make the final prediction. My goal was to see if combining two models that were trained only on

clean data could still perform well on adversarial inputs. I tested the ensemble on the same perturbed malicious samples that I used to test the adversarially trained model. This gave me a good way to compare the two defense strategies. I tried one that required retraining with adversarial data, and one that did not.

Defense Mechanisms

I wanted to test two different defense approaches to see how each would handle adversarial attacks. The first was adversarial training, where I retrained a neural network using both clean and perturbed malicious samples. My goal was to teach the model how to recognize subtle changes in malicious inputs and stay accurate even when those inputs were slightly modified. This approach gave the model a better chance at catching adversarial and integrity attacks, especially when they were hard to detect, and attackers become more sophisticated.

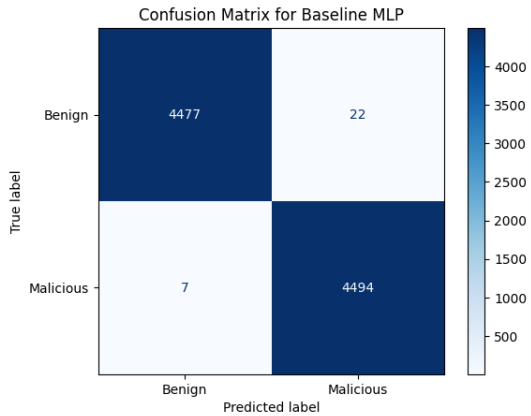
The second defense I used was ensemble learning. I combined my baseline MLP with a Random Forest classifier and used soft voting to make the final predictions. I chose soft voting because it averages the prediction probabilities from both models instead of just picking whichever one has the majority vote. I felt this would give more reliable results, especially in cases where one model is more confident than the other. What I really liked about this method is that it gave me the best of both worlds: strong performance on clean data and surprisingly good results on adversarial inputs, even without additional training. It added stability without adding complexity, and in the end, it was the most balanced defense overall.

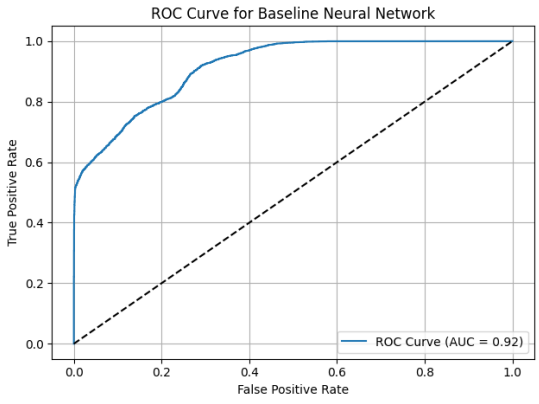
Experimental Results

I tested three models in this project: the baseline MLP, the adversarially trained MLP, and the ensemble. I evaluated each model on clean benign samples, clean malicious samples, and perturbed malicious samples using different epsilon values.

Baseline Model Performance

The baseline MLP was trained on clean data and performed well on both benign and malicious samples and achieved a 99.61% accuracy with a recall of 0.9971 and a precision of 0.9951. However, it showed a slightly higher false positive rate of 0.0049 compared to the other models. The confusion matrix and ROC curve below provide more detail on how the baseline model handled classification tasks. The ROC curve for the baseline neural network had an AUC score of 0.92, showing that the model was strong at distinguishing between benign and malicious traffic but still had some room for improvement under more difficult conditions.

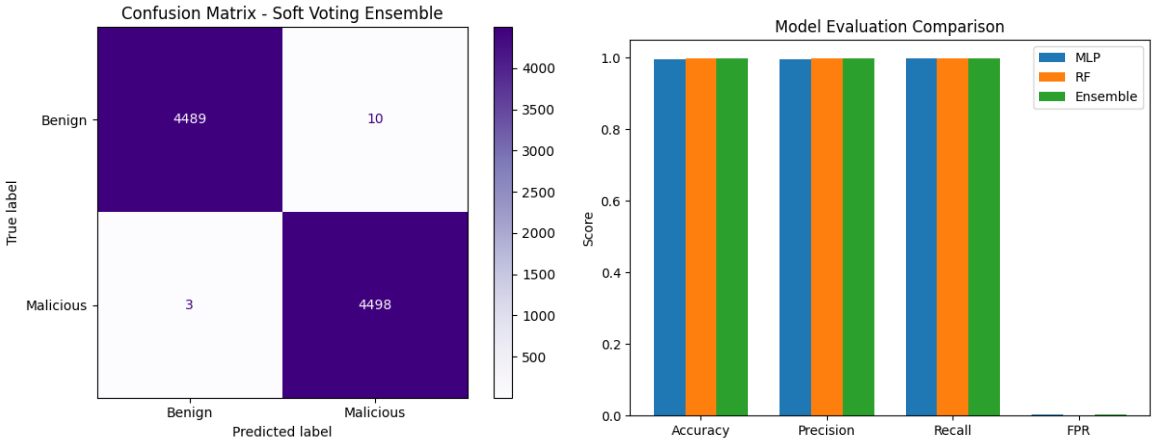




Alenna Zweiback
Project #3

Ensemble Performance

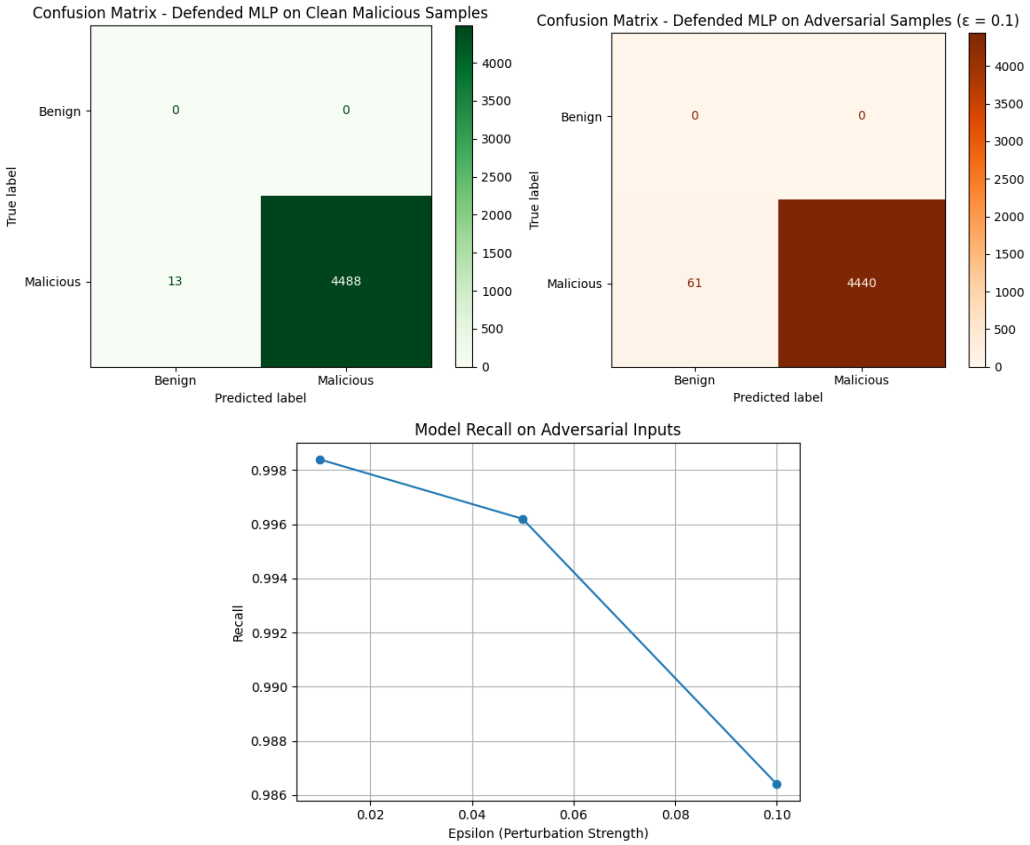
The Random Forest classifier outperformed the baseline in every metric with an accuracy of 99.89% and a really low FPR of 0.0020. The ensemble model, which used soft voting between the MLP and Random Forest, achieved a good balance with an accuracy of 99.86%, precision of 0.9978, recall of 0.9993, and FPR of 0.0022. The confusion matrix for the ensemble method shows a very small number of misclassifications. I also added a comparison chart showing accuracy, precision, recall, and false positive rate across the baseline MLP, Random Forest, and ensemble model. It helped me see that while all models performed well, the ensemble maintained high performance across every metric with the added benefit of robustness without requiring adversarial training. This made me realize that in some cases, a well-designed ensemble can offer a strong defense strategy without the added complexity of retraining on adversarial data.



Adversarial Training Performance

The adversarially trained MLP showed strong robustness against attacks. On clean benign samples, it misclassified 74 inputs as malicious, which raised the false positive rate to 1.64%. It is not ideal, but on the malicious and perturbed malicious samples, the model was almost perfect. It had 100% precision and extremely high recall across all three perturbation levels (epsilon = 0.01, 0.05, 0.1). This shows that adversarial training really helped it recognize malicious behavior even when it was slightly disguised. The table below summarizes the results for the adversarially trained MLP, along with the confusion matrix results. I also included a line chart that shows how recall changed as the level of noise increased. The chart helped me see that the model's ability to catch attacks dropped slightly as the perturbations became stronger. Even though overall accuracy stayed high, the recall dropped just a little, which means the model started to miss a few more malicious inputs as the attacks got harder to detect.

Test Set Type	Accuracy	Precision	Recall	FPR	Confusion Matrix
Clean Benign	0.9836	0.0000	0.0000	0.0164	[[4425, 74], [0, 0]]
Clean Malicious	0.9982	1.0000	0.9982	0.0000	[[0, 0], [8, 4493]]
Perturbed (ε = 0.01)	0.9984	1.0000	0.9984	N/A	[[0, 0], [1, 4500]]
Perturbed (ε = 0.05)	0.9962	1.0000	0.9962	N/A	[[0, 0], [1, 4500]]
Perturbed (ε = 0.1)	0.9864	1.0000	0.9864	N/A	[[0, 0], [4, 4497]]



Insights or limitations observed

This project helped me understand how adversarial training can make a model much more resilient to evasive attacks. I was surprised at how well the defended MLP handled perturbed samples as it basically never missed a malicious input. But at the same time, it misclassified a lot more benign traffic than I expected. That tradeoff between catching every attack and not flagging innocent data is something I did not realize would be so challenging. I learned that you cannot always have perfect accuracy across every class when you add defenses like this, no matter how hard you try to train the model to do so.

The ensemble method ended up being a really good middle ground. Even though it was not trained with any adversarial examples, it still performed really well on the perturbed malicious inputs. It kept false positives low and recall high, which made it feel like a more practical and balanced option especially for situations where retraining with adversarial data might not be possible or known. In other courses, we have used gradient descent methods to create noise in neural network models. I did not test them in this project, but I learned they are more advanced than the basic noise method I used. These types of attacks are stronger and can be harder to defend against. I would like to explore them in the future to see how well my models and defenses hold up. I also think it would be interesting to test these strategies on real-time data. This project gave me a much better understanding of how machine learning models can be attacked and ways to create defense mechanisms and the importance of this application to cybersecurity.