Jess Miller, Anthony Lin, Anthony White

Professor Christian Lopez

CS 424: Intro to Machine Learning

May 14, 2025

Executive Report - ML Final Project

We wanted to train a model to analyze a cybersecurity dataset with the goal of detecting various malicious attacks. We achieved this by creating two models, a “Defender” model that classifies the data based on specific parameters into either a benign action or an XSS, Brute Force, or SQL Injection attack, and an “Attacker” model that generates different types of attacks within a specific data range, with the ability to mimic either harmful or malicious events.

The dataset we used was pulled from CIC-IDS-2017, and is a cleaned dataset from the University of New Brunswick, intended to provide a realistic, labeled dataset of network traffic for machine learning-based security research. We focused on the ‘Thursday Working Hours Morning WebAttacks’ file which has 79 numerical features, as extracted by the CICFlowMeter tool from network packets. These features are primarily statistical measures of network flow characteristics, designed to capture patterns related to the various web attack types. Each network flow is labeled with a category: BENIGN (normal traffic) or one of the attack types (Web Attack XSS, Web Attack SQL Injection, Web Attack Brute Force). One thing to note is that the dataset has heavy class imbalance, with far more benign traffic than attack traffic, and the attack types themselves are also imbalanced.

We created 2 models to solve our problem, with both of them being able to feed off of each other. The first one is our defender model, which is a multi-class classification model designed to identify different types of web attacks while also recognizing benign traffic. The goal

is to maximize correct classifications across all classes while minimizing misclassifications between attack types. This model implements data preprocessing to ensure data quality and compatibility, feature scaling to normalize feature ranges, Synthetic Minority Over-Sampling Technique (SMOTE) to try and address the imbalance in classes we have in our data, and implements XGBoost as the classification algorithm. This specific algorithm was chosen due to its ability to handle complex, non-linear relationships in network traffic, its built-in handling of imbalanced data, and its strong performance when dealing with tabular data. The main design decisions we made were splitting the data before preprocessing to prevent data leakage, scaling the data before implementing SMOTE so that it isn't affected by different feature scales, and implementing SMOTE instead of undersampling our benign samples to reflect the data more accurately.

The defender pipeline begins by loading a raw CSV file. We stripped all header whitespace, coerced all features to numeric, replacing infinities with zeros, and clipping extreme values to $\pm 1 \times 10^6$ to ensure that the data is clean and has bounded inputs. A pre-trained label encoder then maps attack types to integer classes, and the data is split 80/20 with stratification to preserve class proportions. A StandardScaler is loaded or fitted and saved, transforming both train and test features to zero-mean and unit-variance. If available, GAN-generated "fake" samples are stacked onto the real training data, after which a cleanup function removes any residual NaNs or infinities. The pipeline then applies SMOTE to oversample the Brute-Force and XSS classes up to the majority size and computes per-sample weights to equalize class influence. Finally, an XGBoost classifier with tuned hyperparameters is trained on the balanced, weighted set. Its performance is then evaluated on the hold-out data, and special probability‑thresholding

is applied to boost recall for the critical XSS attack class, culminating in a confusion matrix that highlights remaining misclassifications.

The second model we created was our attacker model, which is a Generative Adversarial Network (GAN) that's been specialized for crafting adversarial network-flow samples against the XGBoost defender model. The attacker pipeline begins by loading the defender's artifacts (a fitted label encoder, feature scaler, and trained XGBoost model) and then reads the same raw flow data, isolating the attack records, and applying the defender's scaler. To try and trick the defender, it trains a small surrogate binary classifier on the defender's augmented training set so that the generator can learn to not only fool the GAN discriminator but also trick the defender's decision boundary. The code then defines a four-layer Generator and a three-layer Discriminator in PyTorch, force-reinitializes their weights, and runs a 200-epoch adversarial loop. The Discriminator learns to distinguish real from generated flows, while the Generator optimizes a combined loss that both maximizes Discriminator error and maximizes the surrogate's "benign" confidence. We print the discriminator and generator losses and the "fooling rate", which is the fraction of generated flows the XGBoost model misclassified as benign. After training, it evaluates the final generator on 1,000 fresh attack samples, saves both network weights, and dumps a labeled set of synthetic adversarial flows to harden the defender model.

Lastly, we wrapped our defender and attacker into an N-cycle adversarial loop that lets them continuously challenge and then harden one another. In each cycle, the loop first detects whether any new GAN-generated samples have been added; if so, it retrains the XGBoost defender end-to-end (data cleaning, scaling, SMOTE, weighted training), otherwise it simply loads the existing model. Next, the attacker uses the defender's current parameters and raw attack flows to train (or reload) its GAN generator, augmenting its pool with a fresh batch of 1k

adversarial examples. After each cycle, we log the defender's test accuracy and the attacker's fooling rate. Over multiple rounds, this process exposes the defender's weaknesses, patches them with newly generated attacks, and then tests whether the attacker can still slip past, driving both models toward steadily stronger, more robust performance.

After running the defender model, we generated a confusion matrix, which shows that it does an excellent job at classifying Benign traffic and a pretty good job at figuring out if data is malicious, but struggles in actually classifying between different types of attacks. Due to the small percentage of our data that were actually attacks, we expected our model to struggle with differentiating between them, with the biggest culprit being between XSS and Brute force attacks. Overall, while our model is good at detecting when it is being attacked, it currently isn't the best at classifying the attack correctly.

Through running the attacker model, the GAN model learns to trick our XGBoost defender very quickly as its surrogate loss falls from 0.62 to 0.54 in just a few epochs, and by epoch 20 it can fool the defender 100% of the time on its training batch—but this success is brittle. The discriminator and generator losses swing wildly throughout training, and the fooling rate plunges back down (to as low as 9%) before rebounding, showing the model overfits and then forgets. A t-SNE plot confirms that the fake flows cluster separately from real attacks, so although the GAN exploits the defender's blind spots, it does so with unrealistic, unstable samples rather than true adversarial replicas. Overall, the GAN can fool the defender on seen data, but its unstable, unrealistic samples won't hold up in real use.

One limitation that we ran into was due to the data we found. There was an imbalance in attack types. We had access to a lot more data for the benign traffic attacks than we did for the SQL Injection and XSS attacks. This could lead the model to have a biased learning when

thinking about the different types of attacks it may encounter. Another limitation that could possibly harm our model is the possibility of the attacks being too specific. If this is the case, then the defender model may learn how to defend specific signatures, and if a similar attack occurs, it may not perform well as it is not the same exact attack it is used to. We also ran out of GPU overhead space within Colab. This limited our ability to train the models and specifically the adversarial loop to the level we wanted to do.

If we had more time and more sophisticated technology, we would have liked to do the following in order to train the model better. For the defender model specifically, feature engineering could be a huge improvement in order to better differentiate the attacks. Our model has trouble distinguishing between Brute Force and XSS, so this could help address our issue. If given access to more GPU overhead in Colab, we could implement Hyperparameter tuning in order to optimize parameters in each of the models, and run the adversarial loop for more iterations and make it more sophisticated. As for the attacker model, we could use a more stable GAN variant (like WGAN-GP) and a stronger, differentiable surrogate that better mimics the XGBoost boundary, which could yield smoother training and more realistic flows. Additionally, adding a conditional GAN structure would let us tailor adversarial samples to each attack type, and enforcing protocol constraints (e.g. valid port ranges or integer packet counts) would keep generated traffic plausible. Lastly, incorporating some diversity to promote losses or gradient penalties could prevent mode collapse and help the GAN generalize its fooling ability to truly unseen attacks.