Project Progress Report

(Manigandan Ramadasan)

Project Title: Machine learning for anomaly detection in encrypted network traffic

Introduction:

The increasing reliance on network communication and the growing complexity of cyber threats have made anomaly detection a critical aspect of network security which helps to keep the network secure from cyber-attacks. Detecting anomalous patterns in network traffic plays a very crucial role in identifying potential security breaches, preventing cyber-attacks, and ensuring the confidentiality, integrity, and availability of network resources. However, as the need for robust anomaly detection mechanisms grows, so does the concern for preserving the privacy of sensitive network data. This is the case when anomaly detection tasks are outsourced to third-party service providers where protecting the privacy of network traffic becomes even more critical.

Homomorphic Encryption, the holy grail of cybersecurity which had many developments in last decade is a promising solution for preserving the privacy of sensitive data while allowing computations to be performed on encrypted information. By leveraging the power of homomorphic encryption, organizations can outsource anomaly detection tasks to third-party service providers without revealing the underlying network traffic. This approach enables privacy-preserving anomaly detection, allowing organizations to enhance their security and safeguarding sensitive information about their network.

The primary objective of this project is to explore the feasibility and effectiveness of utilizing homomorphic encryption in detecting anomalies based on the statistical measures of network traffic flow. In this progress report, my goal is to provide detailed explanation of the work I have till until now which includes preprocessing the dataset, analysing it, applying machine learning and deep learning algorithms on homomorphically encrypted data for anomaly detection.

Dataset and Libraries:

I chose the CICIDS2017 dataset because it contains a large and diverse collection of network traffic data, making it suitable for evaluating the performance of intrusion detection systems. For performing Homomorphic Encryption, I used the TenSEAL library because this is a stable library when compared to other libraries and it has an implementation of CKKS scheme which is used in the project because the dataset consisted of floating-point numbers. When it comes to deep learning framework, I chose PyTorch over TensorFlow due to TenSEAL's support to its framework.

When working with TenSEAL, I had to choose parameters carefully and try different parameters. The two main hyperparameters are polynomial modulus (N, which defines the security level and size of ciphertext elements since it represents the lattice dimension) and coefficient modulus (q, which indicates the level of the scheme/multiplicative depth) which is a list of prime numbers that represents the bit sizes of the prime numbers that will be generated by TenSEAL. Another parameter is the scaling factor which defines the precision of the computations performed.

General Workflow:

The brief workflow of the project is as follows:

1. Preprocess the CICIDS2017 dataset.
2. Split the dataset into train and test dataset (8:2 respectively).
3. Train a model using PyTorch using plaintext by trying out different combinations of meaningful hyperparameters (optimizer, learning rate, batch size and other hyperparameters particular to the algorithm).
4. Evaluate the model using plaintext.
5. Extract the weights and biases of the model.
6. Encrypt the data using TenSEAL based on the values of N and q.
7. Encrypt the weights and biases (optional, since ciphertext-ciphertext multiplication increases the computation time exponentially which is bad for IDS since there will be a lot of network traffic flow which will have to be evaluated in real time).
8. Evaluate the model using encrypted data.
9. Decrypt the result and apply sigmoid function.

Preprocessing:

In the initial stage of the project, I performed data preprocessing. This is an essential step in any machine learning/deep learning project, as it prepares the raw data for further processing and analysis. In our project, the preprocessing steps were designed to clean the data and convert it into a format that could be effectively used by our machine learning model and the steps are as follows:

- Dropping unnecessary (Destination Port) and repeated columns.
- Handling Null and Infinite Values:
  - I tried out various methods to handle null and infinite values especially for the samples which were classified as malicious (14 malicious classes) since 18% of the dataset only consisted malicious samples. I tried filling the null and infinite values by the mean of column belonging to the particular attack class in order to preserve the characteristics of the attack, but this didn't give good result because there were many outliers which pulled the mean to extreme. I tried the same by using median and this also failed since there were too many diverse values which didn't give a good result (as the column was filled with floating point values). I also tried to impute the values based on KNN algorithm (KNN Imputation). I used elbow method to select the optimal K value and imputed the null and infinite values, but it still gave less accuracy when compared to the model trained when the null and infinite values were dropped. So, I dropped the null and infinite values.
- Next step was to remove duplicate values to avoid bias and overfitting.
- Next, I performed correlation analysis and removed the features with high correlation to avoid multi-collinearity problem.
- Converting multiclass labels to binary class labels (Benign – 0/-1 and Malicious – 1)
- Since there were a lot of features, it's better to remove some to improve the computational efficiency and limit error growth while performing homomorphic encryption operations. To achieve this, I performed feature selection and selected top

feature (40 – trial and error method) using Random Forest regressor. I chose Random Forest Regressor for feature selection because it gave the highest accuracy while classifying the samples.

Logistic Regression:

Logistic Regression is the most common algorithm when it comes to binary classification due to its robustness, but the problem is the sigmoid function. This can be mitigated if sigmoid computation is pushed to the client side. I experimented with different optimizers and different parameters, and I found that LBFGS converges faster and gives better results along with Binary Cross Entropy Loss function. I developed the model using one linear dense layer which requires only one multiplication operation. The number of elements in the q list other than first and last elements defines the number of multiplications that can be performed i.e., multiplicative depth. Since we need only one multiplication operation, there only one element and we also have 20 bits (60 - 40) for the integer part in the q, which should be far more enough since output values aren't that big (even smaller integer works but we should maintain the values in such a way that it must be congruent to 1 modulo 2*polynomial modulus degree). Scaling also played a very important role. There are nonlinear scalers like Quantile Transform, Power Transform which gave better results than linear scalers but since we can only do homomorphic operation on linear function, I had to choose between Min Max Scaler and Standard Scaler. In most of the cases, Min Max Scaler gave better results and since Logistic Regression has multiplicative depth of one, I chose Min Max Scaler, and I maintained the precision. The reason for choosing Standard Scaler for other algorithms will be discussed in further sections.

Name of the Parameter	Optimal Parameter
Learning Rate	0.5
Epoch	300
Optimizer	LBFGS
N	8192
q	[60, 40, 60]
Scaling Factor	2^{40}

Table 1: Parameters for Logistic Regression

Metric	Plain Text Data (in %)	Encrypted Data (in %)
Accuracy	94.37	94.37
Precision	92.98	92.98
Recall	95.98	95.98
F1 - Score	94.46	94.46

Table 2: Evaluation Results

The evaluation for the plain text data and encrypted data is the same which implies that we chose the right parameters, and the noise growth was well handled. The time taken to encrypt one sample/network flow took ~3.41ms and evaluation of one sample took ~7.86ms.

SVM:

SVM is known for its effectiveness in high-dimensional feature spaces, where the number of features is large Our datasets have numerous features, and SVM (especially non-linear SVM) can handle such high-dimensional data efficiently. But this is limited by homomorphic encryption due to which I had to go with Linear SVM which is particularly bad in anomaly detection as we can see in Table 3. I developed Linear SVM with one layer along with Hinge loss function and Stochastic Gradient Descent optimizer which gave the better results compared to other optimizers. Since we have only one multiplication operation where weights are multiplied with feature values and bias is added, I chose a small N and q and due to small multiplicative depth, I chose Standard Scaler.

Name of the Parameter	Optimal Parameter
Learning Rate	0.001
Epoch	100
Batch Size	64
C Value	0.01
Optimizer	SGD
N	8192
q	[60, 40, 60]
Scaling Factor	2^{40}

Table 3: Parameters for SVM

Metric	Plain Text Data (in %)	Encrypted Data (in %)
Accuracy	77.43	76.87
Precision	56.96	56.31
Recall	96.47	96.51
F1 - Score	71.63	71.12

Table 4: Evaluation Results

Even though Homomorphic Encryption noise growth is controlled, the results are not impressive. The time taken to encrypt one sample/network flow took ~4.70ms and evaluation of one sample took ~9.98ms.

ANN/MLP:

The ANN or MLP model developed has 3 layers – one input layer (number of input features = 40, number of output features = 24), one hidden layer (number of input features = 24, number of output features = 8) and output layer (number of input features = 8, number of output features = 1) with square activation functions except the last layer. I arrived on this architecture based on trial-and-error method. Increasing the number of layers degraded the results and increased time taken to evaluate. The number of neurons in each layer was also derived based on trial-and-error method. In this case, Adam optimizer performed well than any other optimizers with Binary Cross Entropy with Logit Loss Function. I chose square activation function because it is a linear function, and it is a simple and well-known activation function used in the field of deep learning for homomorphic encryption. Because of this the multiplicative depth increase to five as the model requires five multiplication operation which is reason for five 40s in q and because of this I had to increase the value of N. Along with that, I had to choose Standard Scaler over Min Max scaler to maintain precision. Since Min Max scaler produces values in the range [0,1], the precision gets lost when the values are square due to which I had to switch to Standard Scaler.

Name of the Parameter	Optimal Parameter
Learning Rate	0.002
Epoch	1000
Optimizer	Adam
N	16384

q	[60, 40, 40, 40, 40, 40, 60]
Scaling Factor	2^{40}

Table 5: Parameters for ANN

Metric	Plain Text Data (in %)	Encrypted Data (in %)
Accuracy	95.89	94.59
Precision	94.55	91.13
Recall	97.36	98.77
F1 - Score	95.93	94.79

Table 6: Evaluation Results

We can notice that there is a slight decrease in the metrics for encrypted data which implies there is a little noise growth but this is due to more multiplication operations and this could be avoided if we choose bigger value of N which allows us to choose big q but this will increase the evaluation time exponentially and would require more computational resources and the size of the ciphertext will also become double which is something to be avoided. The time taken to encrypt one sample/network flow took ~14.12ms and evaluation of one sample took ~528.37ms.

CNN:

The model has one Convolution layer (this is limited by TenSEAL because of im2col operation) and two dense linear layers, with square activation function after every layer and sigmoid for the last layer. The trade-off is that I had to use bigger parameters since the convolution operation in TenSEAL is achieved through im2col technique. Due to this, the multiplicative depth increases so I had to increase the value of t while satisfying the condition that the values in such a way that it must be congruent to 1 modulo 2*polynomial modulus degree which led to increase in the value of N to maintain 128-bit security. In this case also, Adam optimizer performed well than any other optimizers with Binary Cross Entropy with Logit Loss Function. I used Standard Scaler for the same reason mentioned in ANN section.

Name of the Parameter	Optimal Parameter
Learning Rate	0.001
Epoch	10
Batch Size	64
Optimizer	Adam
N	16384
q	[40, 31, 31, 31, 31, 31, 31, 40]
Scaling Factor	2^{31}

Table 7: Parameters for CNN

Metric	Plain Text Data (in %)	Encrypted Data (in %)
Accuracy	96.15	96.20
Precision	95.70	95.00
Recall	96.65	97.52
F1 - Score	96.17	96.25

Table 8: Evaluation Results

We can notice that the evaluation on encrypted data is better than evaluation on plain data which implies that the noise was well handled. The time taken to encrypt and evaluate one sample/network flow took ~5963.20ms which is really high, and it might create a bottleneck situation in IDS in real time.

Other Homomorphic Encryption Operations:

I implemented Inverse Algorithm, Min-Max Algorithm, Comparison Algorithm based on Approximation based on the paper titled “Numerical Method for Comparison on Homomorphically Encrypted Numbers” authored by Cheon, J. H., Kim, D., et al. The Inverse and Mix Max algorithm gave correct results, but the Mix Max Algorithm took more time than expected. The main challenge was with the approximation-based comparison algorithm. It didn't provide the expected output, indicating potential issues with its implementation or the approximation approach.

Future Work:

The foremost goal for is to rectify the issues with the comparison algorithm. If the approximation-based comparison algorithm is successfully corrected and provides the expected results, the next goal is to implement decision trees. Another task is to make a comparison table with different parameter for different algorithm which will be useful for concluding the best algorithm.

Conclusion:

In conclusion, this project progress report highlights the progress made in applying homomorphic encryption for anomaly detection in encrypted network traffic. The results obtained so far demonstrate the potential of homomorphic encryption to preserve data privacy while maintaining effective detection capabilities. However, there are still areas for improvement and further exploration, such as optimizing encryption parameters, addressing noise growth, and refining approximation-based algorithms.