COMP90073 Security Analytics, Semester 2 2020

Project 2: Machine learning based cyberattack detection

Submitted by :
Name : Ribhav Shridhar
Student ID : 1037144
Username : shridharr

Introduction

Cybersecurity is growing and the rate of cyber-crime is rapidly rising. Advanced attacks are considered as the new normal as they are becoming more frequent and widespread. This constant evolution also calls for innovation in the cybersecurity defence.

Network Intrusion Detection and Prevention Systems (IDS/IPS) screen for malicious activity. Signature based IDS relies on recognized signatures and is useful for identifying malwares that fit these signatures. Although Behaviour-based IDS learns what is usual for a server and articles on any activity that deviates from it. Both types, though efficient, have some vulnerabilities.

In this report we are explaining the two tasks performed for anomaly detection. We have used 2 unsupervised machine learning technique to identify anomalous data from the NetFlow data given. Also, how different feature extraction techniques can change the performance of our detection algorithms. In the second task we have used gradient descent-based method to generate adversarial samples against supervised learning models to check the robustness of our model and how does it react to noise. In this task we also check our results for a targeted IP address, and see how the adversarial methods manipulate the data in order to bypass the detection.

Dataset and EDA

We have been provided NetFlow data for a network under cyberattacks. The datasets provided consist of both Botnet and normal traffic data.
The data for Project 2 contains the NetFlow data for a network under cyberattacks. Each line of the dataset includes the following 14 fields: *(1) timestamp, (2) duration, (3) protocol, (4) source IP address, (5) source port, (6) direction, (7) destination IP address, (8) destination port, (9) state, (10) source type of service, (11) destination type of service, (12) the number of total packets, (13) the number of bytes transferred in both directions, (14) the number of bytes transferred from the source to the destination.*

We performed some EDA to analyse the data set and fetch some its most important patterns and characteristics. It was performed using Splunk.

Some of the top metrics are summarized below –

1.

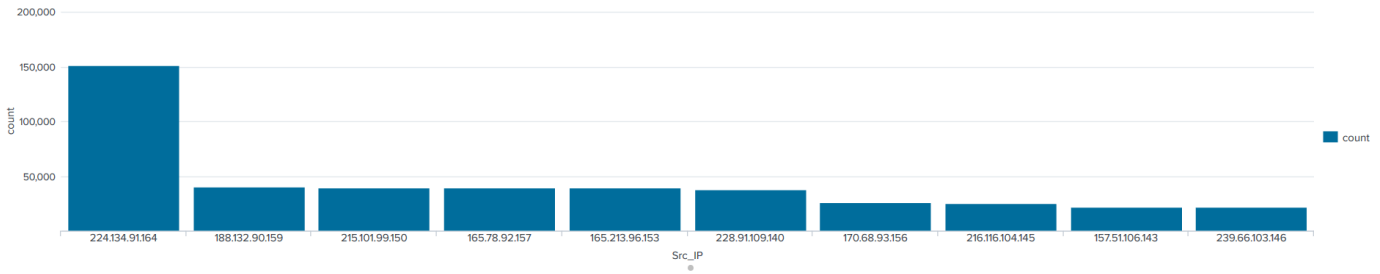

Figure 1. Top Source IP by number of events

2.

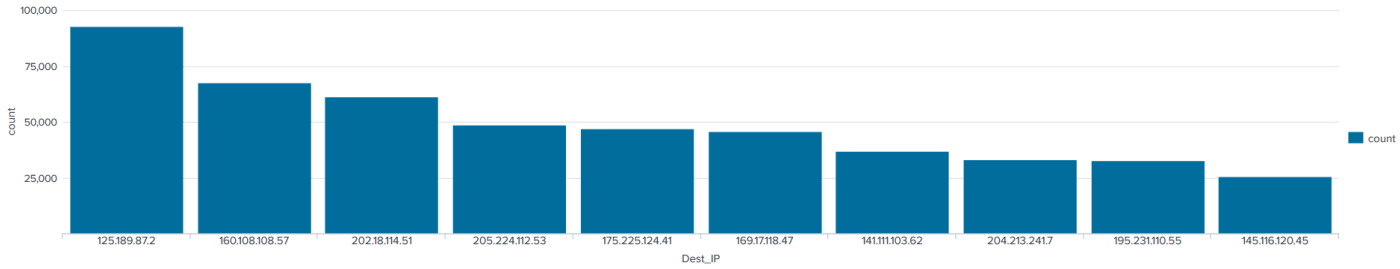

Figure 2 Top Destination IP by number of events

3.

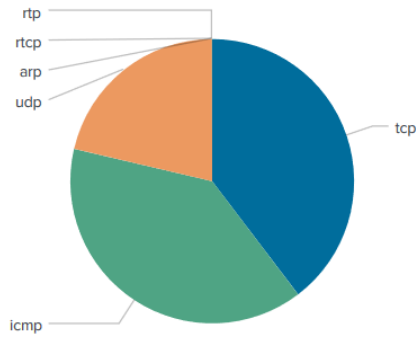

Figure 3 Top Protocol by number of events

4.

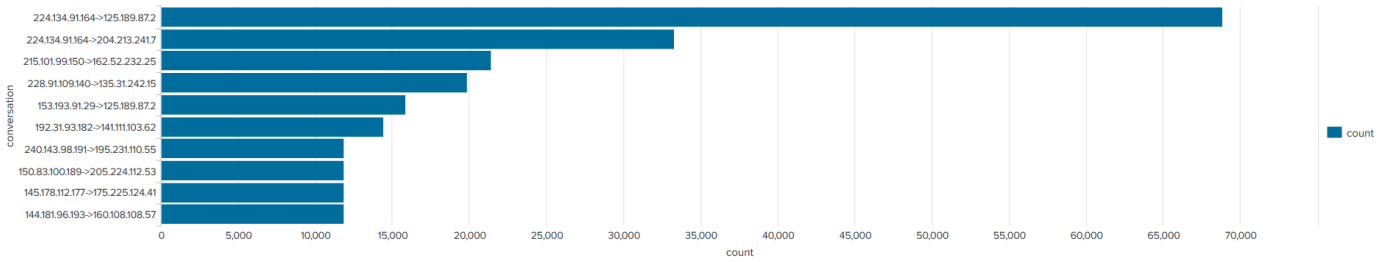

Figure 4 Top communication (Src_IP -> Dest_IP)

5.

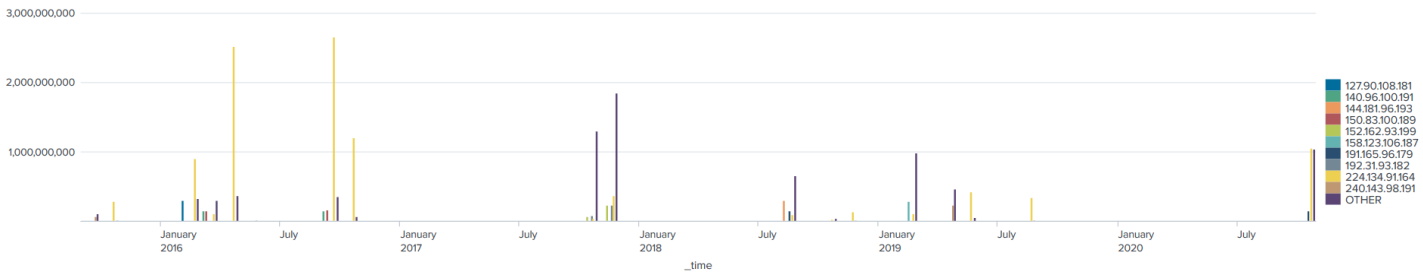

Figure 5 Timeline of Bytes transferred by Source IP address

6.

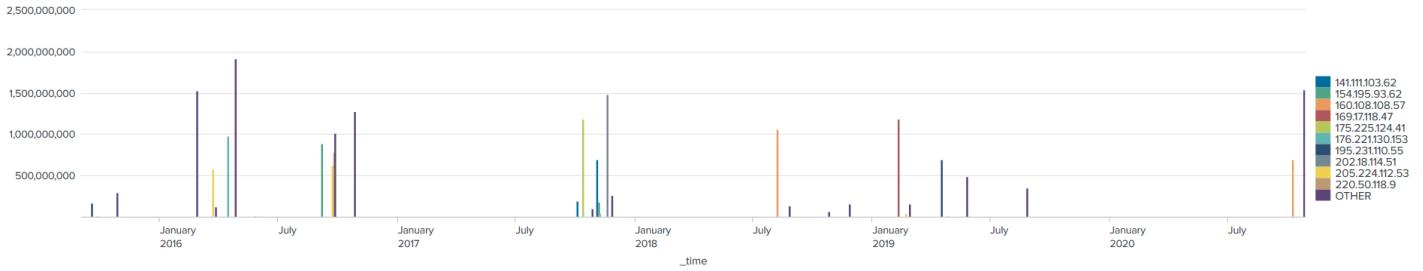

Figure 6 Timeline of Bytes transferred by Destination IP address

7.

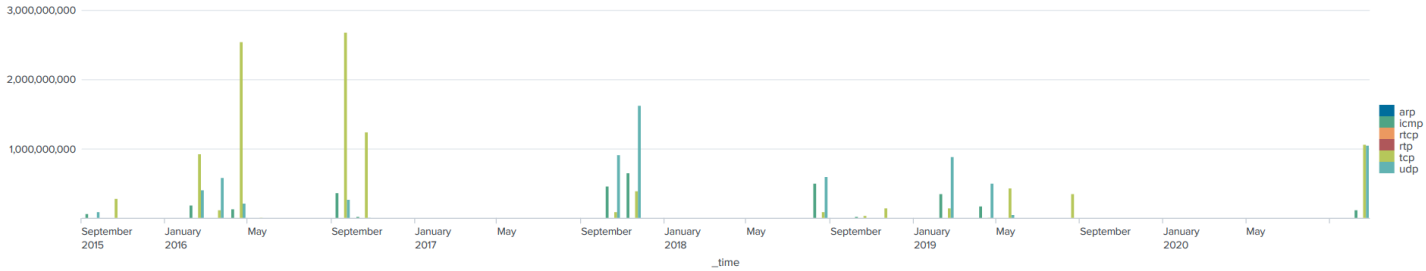

Figure 7 Timeline of Bytes by Protocol

8.

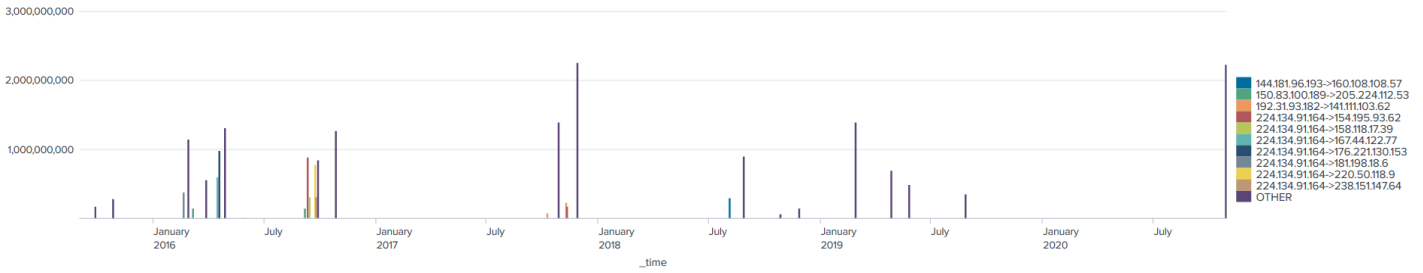

Figure 8 Timeline of Bytes by communication

Pre-Processing

On all the data sets, the following has been performed for pre-processing and making the data uniform.

1. Adding column names on to the data set, as given in the data description.

```
training_data = pd.read_csv("training_data.csv", header=None)
training_data.columns = ['Date_Flow_Start', 'Duration','Protocol','Src_IP','Src_Port','Direction','Dst_IP','Dst_Port','State','Source_Service','Dest_Service',
```

2. Changing the *timestamp* to type Integer
3. NetFlow data consists of categorical data that have to be converted into numerical or Boolean values, which can result in a matrix size that is too large and can produce memory concerns. To lower down the volume of data to be handled, we use a schematic to sample the NetFlow data using time window. We have chosen a time window

of 2 minutes with a stride of 1 minute. We then pre-process the bidirectional NetFlow dataset and extract categorical and numerical characteristics that describe the dataset within the given time window.

```
window_width = 120
window_stride = 60
```

4. Used RegEx matching for labelling the records as 1 for Botnet Traffic or 0 for Normal Traffic. This was done for the Validation Set in the task 1 and for all (Train, Valid and Test) sets in task 2.

Feature Engineering

This step is required to choose features from the ones extracted from the previous step. It entails the use of feature selection methods to decrease the dimensionality of the training data.

For the task 1, three different data sets were generated :

Set 1

The numerical data in the NetFlow data set were processed to generate features like :

a. Sum
b. Mean
c. Standard Deviation
d. Maximum
e. Minimum

Data considered was duration, total amount of bytes exchanged (bidirectional), and number of bytes from source to destination)

Set 2

In this set, we considered the categorical variables in the NetFlow dataset, i.e. the data on which mathematical operations cannot be applied. Data like Source ports, destination address, and destination ports.
The following features were generated on such data :

a. the number of unique occurrences in the subgroup
b. the normalized subgroup entropy defined as

$$E = -\sum_{x_i \in X} p(x_i) \log p(x_i) \quad \text{with} \quad p(x_i) = \frac{\#x_i}{\#X} \quad \text{and} \quad \text{X, the subgroup of the source address}$$

Set 3

In this set, we used Pearson correlation to generate features.
Correlation was calculated on data from set 1 and set 2, and features having high correlation value were selected and the others were dropped.

Methodology

Task 1

In this task we are using two unsupervised machine learning algorithms on each of the above 3 feature sets.
The unsupervised machine learning we have used are :

1. Isolation Forest Algorithm :

is an unsupervised learning algorithm commonly used for anomaly detection that works on the notion of "*isolating anomalies*", instead of the commonly used techniques such as profiling normal points.

The Isolation Forest algorithm is based on the notion that outliers are records that are uncommon and unique, which make them simpler to detect. Isolation Forest uses an ensemble of Isolation Trees for the given data points to isolate anomalies.

Implementation

We have used the PyOD python package to execute this algorithm. PyOD is a Python package for implementing outlier detection algorithms on multivariate datasets.

i. First we need to be set the contamination factor in our data, which gives what percentage of training data we expect is are outliers. Initially run the trained with default parameters and then are further tuned using the validation dataset.
ii. After training, we compute the of outlier scores of each record in the training data. As the outlier score is based on the *Euclidean Distance*, the greater score, the more anomalous the data. Then the scores are sorted in descending order to calculate the threshold score, i.e. a score above which we classify the record as an anomaly.
iii. Then we calculate the outlier scores for the validation data. If the predicted score is larger than the threshold score set in step 2, then the record is classified as an anomaly(in our case- botnet traffic). To evaluate the model, F1 metrics are used.
iv. The algorithm is then tuned fittingly taking the F1 scores on the validation data into consideration. Also, as we did not know the exact proportion of outliers in our training data, we also experiment with the contamination factor, but carefully so as to not over or underfit the model.
v. Outlier scores are then computed(using the best tuned model after step 4) on test data and outliers are predicted similarly as in step 3.

The above same implementation is done of all 3 generated feature sets.

2. CBLOF Algorithm :

computes the outlier score based on "*cluster-based local outlier factor*". An anomaly score is calculated using the distance of each record from its cluster center multiplied by the records belonging in the same cluster, given in the below equation.

$$CBLOF(p) = \begin{cases} |C_i| \cdot \min(d(p, C_j)) \text{ if } C_i \in SC \text{ where } p \in C_i \text{ and } C_j \in LC \\ |C_i| \cdot d(p, C_i) \text{ if } C_i \in LC \text{ where } p \in C_i \end{cases}$$

Implementation

Similar to Isolation Forest, PyOD is used to implement CBLOF.
We more or less follow the same approach as Isolation forest,

i. fit/train the model,
ii. calculate the outlier scores,
iii. use outlier scores to tune the hyper parameters on the validation set,
iv. And then apply the best tuned model on the train set.

The above same implementation is done of all 3 generated feature sets.

Task 2

In task 2 we are using gradient based methods(FGM) to generate adversarial samples against supervised machine learning model and how it can affect its robustness.

We need to first build a strong and robust classifier using supervised methods, hence deceiving a strong classifier would ensure robust efficiency and accuracy for the generator computing the adversarial samples.

In this task also, we are using NetFlow data, which has been preprocessed in a similar way as for task 1 and using feature engineering as explained in the "*Dataset*" section of this report.

Generation of Adversarial samples is quite different for computer vision-based domain, as convolutions are used for image processing which cannot be used for network dataset(mostly categorical). So we needed to generate features manually to give input to compute the adversarial samples, which also enabled us to use more flexible feature generation methods.

Implementation

i. We consumed the training data with classes(botnet or normal traffic) and prepared a supervised learning model by means of a Support Vector Classifier(Support Vector Machine – SVM).
ii. After training the model, we tuned the model parameters to get the optimum performance out of it. After obtaining the best possible parameter settings the model is fitted on these and will be used as the discriminator for the adversarial samples.
iii. In the next step we selected an IP address which was classified as an anomaly by the labeled training data and also predicted by the supervised model. The IP address selected was 147.49.95.181.
iv. Now we needed to add perturbations to the input data so that we can force the model to predict the Botnet Traffic IP address as a Normal Traffic(benign) IP. To insert perturbations to the data features, we implemented the Fast Gradient Method, which computes adversarial samples i.e. data with Normal data with the added Perturbations.
v. After computing the adversarial samples, we used the classifier again to predict the labels. On evaluating the predicted labels, we find that the classifier performed weakly in identifying anomalous data i.e. Botnet Traffic and also incorrectly classified the targeted IP address 147.49.95.181 as a Normal Traffic IP(Benign).

The above implementation proves that the adversarial samples generated using the Fast Gradient Method deceived the classifier in labeling 147.49.95.181.

Results

In this section we will discuss the results for both task 1 and task 2.
We have used the following metric for the evaluation of the models :

i. F1 score
ii. Accuracy
iii. Precision

All of these have been computed using the Scikit Learn Metrics Package in Python.

Task 1

We carried out parameter tuning by experimenting with the values of *Random State* and the *Contamination*
The results are for the tuning which fetched the best F1 scores with the validation set.
After carrying out parameter tuning for iForest, for set 1, best results were obtained with the *contamination* as 0.30, 0.25 for set 2 and 0.20 for set 3.
And for CBLOF, the best results were with *contamination* of 0.10 for set 1, 0.15 for set 2 and 0.30 for set 3.

The same can be seen in the F1 scores given below :

Model -	**Isolation Forest**	**Isolation Forest After parameter tuning**	**CBLOF**	**CBLOF After parameter tuning**
Set 1	42.31%	72.35%	55.42%	56.54%
Set 2	48.83%	73.82%	40.52%	74.02%
Set 3	48.75%	70.46%	31.81%	54.33%

Task 2

In task 2, before generating the Adversarial Samples, first we had to train and tune a supervised classifier.
After getting the best results on the test set (Support Vector Classifier), we used that model for testing with the Adversarial Samples generated using Fast Gradient Method

Data	**Parameters After tuning**	**Precision**	**Accuracy**	**F1 Score**
Validation with Labels (Train vs Validation)	No tuning	0.87	0.85	0.83
Test with Labels (Train vs Test)	No tuning	0.88	0.87	0.80
Validation with Labels (Train vs Validation)	*Gamma = 0.001*	0.89	0.91	0.88
Test with Labels (Train vs Test)	*Gamma = 0.001*	0.90	0.89	0.88
Test (Adversarial sample vs Test)	*Epsilon = 0.3*	0.17	0.25	0.21

Analysis

Task 1

We have used Isolation Forests and CBLOF algorithm to train and fit out data sets. Both the algorithms are unsupervised learning-based algorithms.
iForest utilizes no distance or density measures to detect anomalies. This eliminates main computational cost of distance computation in all distance and density-based methods. Also, it has a linear time complexity with low memory constraints.
iForest has the ability to scale up to manage large data volume and high dimensionality which is true cyber/network data.
One major advantage of using CBLOF is the ease of the understanding of its score and its capability to capture outliers that were earlier unseen by the global approaches. Also, CBLOF has linear time complexity.

After performing the steps explained in the Methodology section of the report, we came up with the following analysis :

- There is a significant improvement in the scores after tuning the parameters.
- On average iForest performs better than CBLOF.

- Some features sets have different results as compared to others. The Entropy based - Set 2 performs better for both algorithms.

Task 2

We have used Support Vector Classifier to first train our model before generating Adversarial Samples. SVC is a supervised learning algorithm (As we have been provided with Labels for all data sets).
We decided to use SVM based classifier because it is highly effective for high dimensional data and when the number of dimensions is greater than the number of labels.

After performing the step explained in the Methodology section of the report, we came up with the following analysis :

- The results in the table for task 2 in the *result* section emphasizes that the supervised classifier failed to classify the anomalies in the test data when the perturbations i.e. Adversarial samples have been added.
  This substantial decrease in the model's efficiency can be clearly seen in the low F1 score.
- Fast gradient method is implemented for computing the adversarial samples, which successfully fooled the classifier, as the classifier labeled the IP 147.49.95.181 as Normal Traffic.
- On examining the perturbations added to the features of the source IP address 147.49.95.181, we came to the inference that the Botnet Traffic IPs will be classified as a Normal Traffic if they decrease their period of transmission, bytes communicated to the destination IP, and total bytes transferred during transmission.

Conclusion

To conclude, our project targeted at training and comparing models that are able to detect botnets in a real network traffic represented by NetFlow datasets, and also check how by using gradient methods to generate Adversarial Samples of them can affect the detection.

Assumptions

- We have used Python for preprocessing, feature engineering, and training and testing the models. Splunk Native Features were only used for the initial data discovery and EDA. Also implementing ML on Splunk was more computationally demanding.
- After generating the features based on Mean, Median, Entropy, and correlation it is impossible to go back to the non-mathematised features of the data.

References

- M.-I. Nicolae et al., "Adversarial Robustness Toolbox v1.0.0," pp. 1–34, 2018.
- Y. Zhao, Z. Nasrullah, and Z. Li, "PyOD: A python toolbox for scalable outlier detection," J. Mach. Learn. Res., vol. 20, pp. 1–7, 2019.
- F. T. Liu, K. M. Ting and Z. Zhou, "Isolation Forest," *2008 Eighth IEEE International Conference on Data Mining*, Pisa, 2008, pp. 413-422, doi: 10.1109/ICDM.2008.17.
- Delplace, A., Hermoso, S., & Anandita, K. (2020). Cyber Attack Detection thanks to Machine Learning Algorithms. arXiv preprint arXiv:2001.06309.