Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Digital Forensics

Project 4:
NETWORK ANOMALY DETECTION

Authors
Alexander Bonora
Matteo Pomari

16/09/2022

1 Objectives

This project is intended as a further contribution to the article "Analysis of a 2D Representation for CPS Anomaly Detection in a Context-Based Security Framework." The purpose of this work is, starting from the representation of network traffic data provided, to build an autoencoder capable of distinguishing when the considered image represents an anomaly or not, which in our case correspond to an attack. This paper will first give a brief description of how the data was collected and how it was efficiently represented. We will describe how we build the dataset and how we handle it. Then we will talk about the autoencoder, about its loss function and about its threshold, which are key parameters for our objective. We finally can see the results and the conclusions.

2 Setup description

Let's begin with a brief description of the dataframe we are going to study. This dataset is composed of two parts: a calibration subset and a test subset. The former includes background traffic and can be used for training. While the second is a combination of background traffic and controlled attack traffic and can be used for testing. Data capture is performed for incoming and outgoing traffic. The captured data are organized in flows and for each flow the following features are provided: timestamp of the end of a flow, duration of the flow, source and destination IP addresses, source and destination port, protocol, flags, forwarding status, type of service, packets and bytes exchanged in the flow.

Concerning the attacks, the following classes have been simulated:

- Denial of Service (DoS);
  - DoS11;
  - DoS53;
- Port Scanning;
  - Scan11;
  - Scan44;

3 2D Traffic Representation

The dataset we are considering is a representation of a traffic where the presence of an attack is highlighted. It basically consists in arrays of pixels where the value of the elements corresponding to the attack became significantly different from all the others. In this work, exchanged bytes have been considered as traffic parameters and source and destination IPs have been used for indexing rows and columns. An example of images can be seen in Figure 1 but for more detailed information, please refer to the reference paper.

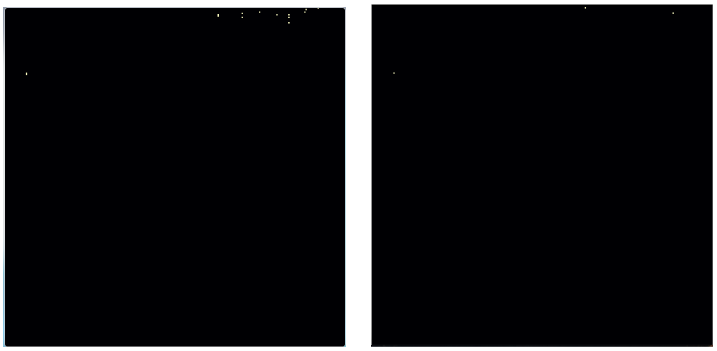

Figure 1: Examples of traffic representation

4 Dataset

The development environment that we decided to use is Google Colab, which allowed us to mount our Google drive and unzipping the files 'train.npz' and 'test.npz'. Respectively the files used for the training phase and the testing phase. An initial issue we faced was the size of the dataset. In fact, the file **'train.npz'** contains 435404 images while the **'test.npz'** contains 15055 images. For this reason we made use of the keras.tensor,optimal objects when you have to use matrix to represent data, and of keras optimized libraries. We also used the mmap strategy for free some ram and save variables to internal memory. Even with these strategies they are huge number to handle for our computers. For this reason we decided to randomly choose a percentage of these datasets and work with them. For our work we chose respectively:

PERC_TRAIN = 0.01%
PERC_TEST = 0.4%

At this point we created two array **'X_train'** and **'X_test**,' and resized them into matrices (256,256,1). In addition, in order to verify the correctness of our algorithm, we created another array **'Y_test'** where we saved the labels of the data we chose previously, taking care to keep the order intact to maintain the data-label relationship. An example of the distribution of the 'Y_test' is showed in
Figure 2.

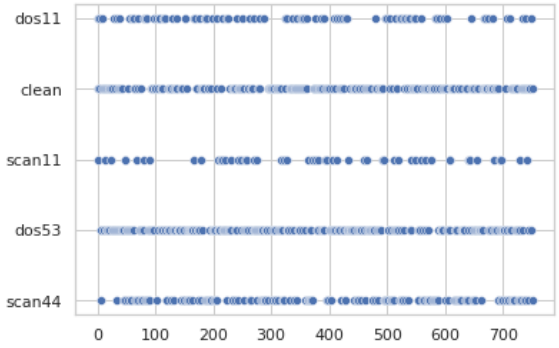

FIGURE 2: Distribution of Y_test

5 Autoencoder

Now to accomplish our goal we need to build a neural network capable of distinguishing, given an image, whether we are in a normal situation, i.e. clean, or whether we are in an anomaly situation, i.e. attack. To do this we decided to build an autoencoder.
An autoencoder is a neural network trained to attempt to copy input into output. Specifically, we designed a neural network architecture such that it imposes a bottleneck in the network that forces a compressed knowledge representation of the original input. The network can be seen as consisting of two parts: an encoding function and a decoding function that produces a reconstruction. Our autoencoder, following the logic just described, start from the original image size (256,256,1), manages to extract 65536 values, and then reconstruct images of the initial size in output.
Once this was done, we performed the autoencoder compile and the autoencoder fit, implementing the callbacks and using our defined **custom_loss_function()**.

```
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=4)
autoencoder.compile(optimizer, loss = custom_loss_function)
autoencoder.fit(X_train, X_train, epochs = 8, batch_size = 16, callbacks=[callback])

Epoch 1/8
273/273 [==============================] - 307s 1s/step - loss: 7071.4810
Epoch 2/8
273/273 [==============================] - 307s 1s/step - loss: 3610.7107
Epoch 3/8
273/273 [==============================] - 317s 1s/step - loss: 3094.7551
Epoch 4/8
273/273 [==============================] - 316s 1s/step - loss: 2729.2136
Epoch 5/8
273/273 [==============================] - 314s 1s/step - loss: 2358.2361
Epoch 6/8
273/273 [==============================] - 305s 1s/step - loss: 2072.5674
Epoch 7/8
273/273 [==============================] - 313s 1s/step - loss: 1889.3105
Epoch 8/8
273/273 [==============================] - 305s 1s/step - loss: 1696.5344
```

FIGURE 3: Fit and Compile

In Figure 3 we can appreciate how the loss decreases during the various epoches.

6 Loss Function

Let's now considerate the **loss_function** used during the compile phase of our autoencoder. The idea that we had for create a good loss_function is to prioritize the white pixels, since they are the ones that contain the discriminating information for a clean sample versus an anomaly one. To do this we applied a mask to the arrays contained in the **Y_true** in order to consider the losses of all non-black pixels. We defined two **loss_functions** in order to adapt the logic both for the train phase and for managing individual arrays.

7 Threshold

Let's now define the threshold value, which determines whether an image is to be considered clean or anomaly, by using the computed losses. The calculation of this value is a key point in order to have an acceptable accuracy of the neural network. A good threshold makes sure that our algorithm is able to determine the highest number of anomalies and the lowest number of false anomalies. The idea of the algorithm is to make a loop that gradually increases the threshold and at each step saves how many samples were categorized correctly. The starting value is the mean of mse_train calculated as percentage of the max mse_train value, which can reach the maximum value considered as 100% contained in mse_train. We evaluate the performance of the classification by increasing by 5% at each loop. This threshold is set to have a good performance for having low FALSE CLEAN. There is a commented part which sets a good threshold in order to have low FALSE ANOMALY. Priority was given in lowering the amount of anomalies falling through while still performing well for the clean not making it through.
In Figure 4 we can see the confusion matrix obtained through the value of the threshold obtained by the logic explained above.

FIGURE 4: Confusion matrix

8 Conclusions

A major drawback of this program is definitely the huge number of the dataset. We think would have had better results if we could have used a larger percentage of the data provided. However, thanks to the autoencoder, the loss function defined and thanks to the method we used to calculate the threshold we obtain satisfactory results in discriminating situations of attack ordinary situations.

References

"Analysis of a 2D Representation for CPS Anomaly Detection in a Context-Based Security Framework" Authors: Sara Baldoni, Marco Carli and Federica Battisti

https://www.deeplearningbook.org/contents/autoencoders.html

https://www.jeremyjordan.me/autoencoders/