Monday, October 24th, 2022

MALWARE DETECTION

Application for IoT Devices

By Pablo Ruiz Lopez

Springboard DSCT Bootcamp

"Your device has been infected": words we never want to read...

Summary

In this project, we intake network flow log files from 20 different malware attacks perpetuated in IoT devices in a project developed at the Stratosphere Lab, AIC Group, FEL, CTU University Czech Republic. The log files were written by a network flow analyzer and labeled by the laboratory according to the test made on each device and labeled at the lab according to the network flow and type of bot used for attacking the device.

First, we got all this data well processed since it was taken from text files wrote by separate machines, so we had to concatenate each malware attack log file, resulting in a raw data set of about **1.2M observations and 17 features.** After this step we did an extensive feature processing; exploratory, analytical, and categorical data analysis to determine which features were relevant for classifying a network flow as malicious (1) or benign (0) and the ones with proper distribution for a data science project. Afterwards, we did other (fundamental) data processing steps like dropping more non-relevant features, one-hot encoding, and dropping duplicate records. These steps allowed us to get a final data set to work with about **380K observations and 67 features.**

The features that were picked in the first data processing stage were:

ts: time stamp of capture in UNIX format
id.orig_p: attacker computer port of origin
id.resp_p: host port used for the response
proto: network communication protocol
service: application communication protocol
duration: time traded between attacker - host
orig_bytes: number of bytes sent to the host
orig_pkts: number of packets sent to the host
resp_bytes: number of response bytes
resp_pkts: number of response packets
conn_state: connection state during attack
orig_ip_bytes: bytes sent to host under IP
resp_ip_bytes: response bytes under IP.

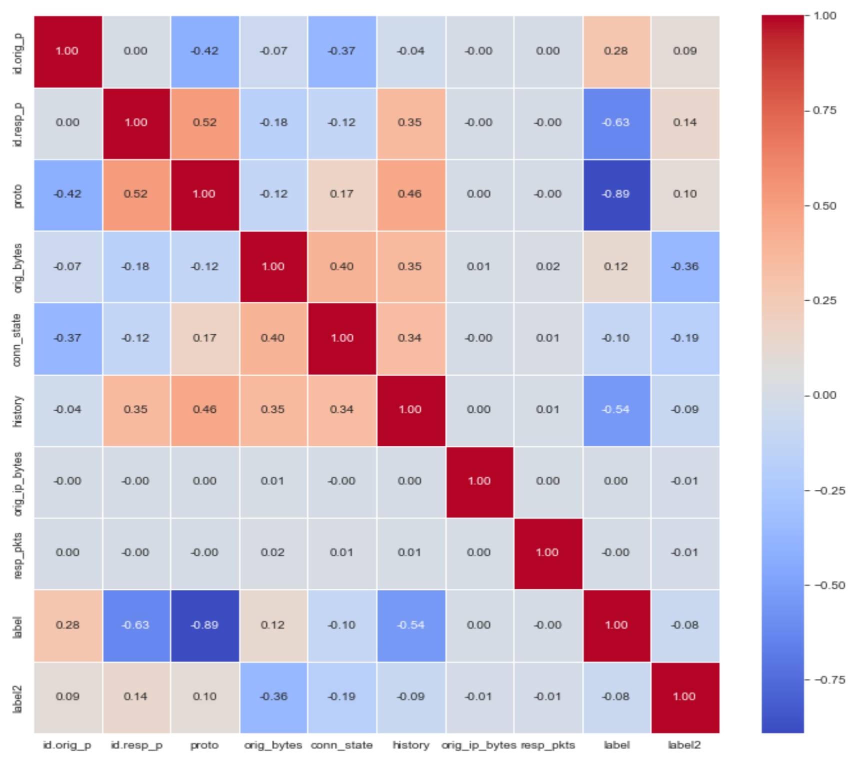

Image 1 – Correlation Matrix to check collinearity between the selected features

During the exploratory data analysis, we plotted the correlation matrix to see which feature was correlated the most with the label and to get a glance at the **collinearity in the features.**

In this step, we found that duration, orig_pkts, resp_bytes, service and orig_bytes were highly linearly correlated. The initial approach was to drop all of them and keep orig_bytes since is the one that correlated the most with the other four.

Another collinearity we found was between resp_ip_bytes and resp_ip_pkts, where we decided to keep the first one since the range of its distribution was less sparse. Then we plotted the matrix, as shown next (Image 1).

Further Feature Processing

Saving different datasets and Feature Engineering

As we can see in the image above, there is a fair level of collinearity found in the selected features, not too bad. Following this step, we decided to explore some dimensionality reduction techniques (PCA), encoding categorical features to pass it all to int/float so we prepare different datasets for inputting to the model. Among them, we included a raw dataset keeping the categorical variables as they were, a second one with the categorical variables encoded and keeping only 8 PCA component and a last one including the encoding and feature engineering steps taken without including the PCA components.

It is important to mention that we found a time stamp at the header of each log file that represents the starting time of the record and was used to decode ts feature from its UNIX format and we named it as:

starting_point: separate time stamp written on each log to indicate the starting point of each record and used for passing ts from UNIX to date-time.

Additionally, by reading the names of the log files, we could eventually retrieve the name of each bot type. This was a feature engineered and included in the dataset as

label2: consisted of 8 types of bot attacks Unfortunately, the class imbalance on this label was extremely high and this didn't allow us to use it for a multi-classification problem since we need far more data than that. However, our main label, balance, did allow us to work on a binary classification problem (malicious and benign network flows).

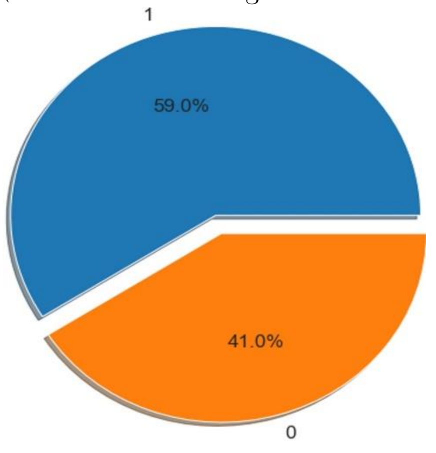

Image 2 – Label class balance:

Class 1: 59% (Malicious)

Class 0: 41% (Benign)

Initial Feature Selection

Criteria Used

The initial feature selection was based mainly on their distribution (most of features were found to be unique values or all null values). Other factors considered included relevancy for the use case, for instance

Initial Feature Engineering

Time Related Features

As an initial approach, there were some features engineered mostly related to time to see if there was any useful information on them that we could use for the prediction.

Label to Use in This Project

Binary Classification

This was selected to be a binary classification problem due to the properties of the data set and the class imbalance found for the multi-class project.

Key Insights for Business

Analysis on Features and Models

Most insightful findings based in features

Considering different models for this project

On this final stage of the project, we reviewed the distribution of some of the selected features that were extremely different between classes (see image 3) and decided to make some simple "initial single-feature models" based on simple assumptions like: "all malware network flows come from TCP protocol". Below are shown the initial model number, features used and their results:

#	Feature	Precision	Recall
1	proto	0.89	1.0
2	id.orig_p	0.88	0.90
3	id.resp_p	0.90	0.92
4	history	0.90	0.93
5	orig_ip_bytes	0.88	1.0

Table 1: Initial Single-Feature Models

As we can see, we got great results on these very simple models. This not only provides great insights into the features but also on the case study itself. Based on this exercise we found out non-linear correlations between the features and label that were not spotted.

Additionally, a very important step taken before running these models was to drop duplicated records shown in multiple log files that were eventually leaking information from train to test data splits and finally, dropped the features mentioned in table 1 plus resp_ip_bytes, resp_pkts which showed a very similar distribution as orig_ip_bytes. This was done primarily to simulate what would happen if we didn't have access to these highly informative features.

Lastly, we couldn't find a good reason to add any time dependent variables since the time of attacks is not relevant for the use case and due to the imbalance in types of botnet attacks, we decided to drop ts, starting_point and label2 as well.

ML Classification Models

Now with the final dataset (380K, 67), we jump into the adventure of testing different ML models.

The models tested are listed below in Table 2. For each model we developed a helper function that computed the classification report, best AUC score, accuracy, confusion matrix and ROC-AUC curve.

Given the nature of the classes, we decided to prioritize **Precision** and **Recall** on label 1 for model selection.

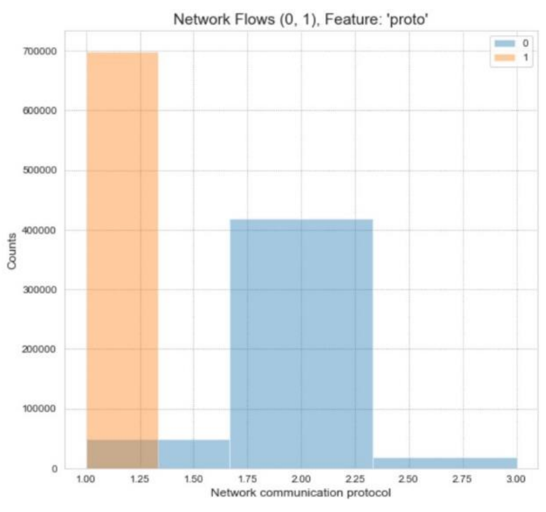

Image 3 – Example class historgram for proto

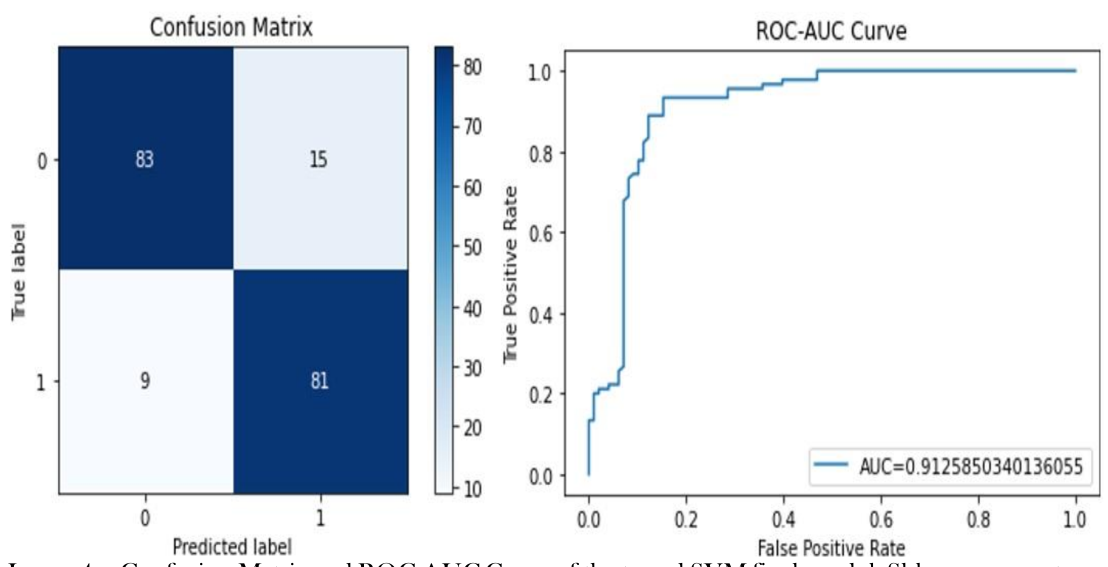

Image 4 – Confusion Matrix and ROC-AUC Curve of the tuned SVM final model. Sklearn paramaters selected and tuned for this model were: **scaler:** StandarScaler(), **clf:** SVC(), **C:** 1.0, **coef0:** 0.0, **decision_function_shape:** 'ovr', **degree:** 3, **gamma:** 'scale', **kernel:** 'rbf' and **probability:** False. All other used library parameters were set as default for this use case.

Model Selection

Model's Results

SVM: Best Performer

In the following table are shown the model results, the parameters used for each model were the default ones selected in sklearn libraries.

Model	Precision	Recall
Logistic Regression	0.84	0.90
KNN	0.76	0.87
Decision Tree	0.78	0.80
Random Forest	0.81	0.86
SVM	0.82	0.92
XGBoost	0.81	0.87

Table 2: ML Models and Results

So, we have two best performers: Logistic Regression and SVM models. To choose which one to choose further, we decided to take into consideration also the computational time for training, which in this case was less for SVM Model.

In this final stage of the project, we decided to compute a Grid Search Cross-Validation step on SVM model and explore which parameters were best for it. This allowed us to increase precision by 0.02 while decreasing recall by also 0.02 (tradeoff found).

Summary

Key Insights and Future Work

These were the results obtained by using the SVM tuned model:

```
| Precision:   | 0.84 |
| Recall   | 0.90 |
| F1 Score    | 0.87 |
| Support    | 0.90 |
| Best AUC Score    | 0.87 |
| Accuracy    | 0.87 |
```

Model sklearn parameters are listed in Image 4 description.

Future Work

In this project, we could confirm the importance of EDA.