Malware Detection using Static and Dynamic Analysis

Sarthak Rout Dawood Bin Mansoor Anubhav Kalyani

1 August 2020

1 Introduction

In this problem, we have used Google Colaboratory as a platform to develop our model, taking advantage of higher processing power (3 GHz) and memory (13 GB) offered by it and envisioning GPU usage in future. Google Colab natively offers a Jupyter Notebook run by Python 3.7 . To store our data-sets, we used Google Drive which allowed us quick computations.
After downloading the files, we observed that the Static Analysis Data consists of two text files for each 'file' – 'Strings.txt' and 'Structure_Info.txt'. We inferred that the 'Strings.txt' was output of command 'strings' to the file which returns strings of contiguous printable characters of length at least 4.

2 Analysing Strings.txt

Looking into some samples of 'Strings.txt' , we observed the name of some functions such as 'GetProcAddress', 'RegDeleteValueA', 'WSOCK32.dll' etc. So, we decided to somehow get all such strings and use them as features, as functions inherently decide the working of a malware; they could be used in discriminating between benign and malware classes. As a heuristic for names of functions, we decided that we will take those strings, which are alphanumeric (Ex: we3d, 54sd5) and which are not numeric (Ex: 2323, 4444444, 232) and a minimum length of 5. We stored these strings in a dictionary with the name of the folder – hash as a key.

3 Analysing Structure_Info.txt

Looking into Structure_Info.txt, we first found it tough to understand what various terms and hex values refer to. Googling about 'IMAGE_DOS_HEADER', brought us to the topic of 'Portable Executable File Format'. We searched through a lot of blogs and thesis papers [1][2][3]. Using this information, we decided that we would take all the features in the structure info including the flags and 'DllCharacteristics' , ignoring sha256 and md5 hashes.

4 Analysing API Trace Reports

For the Dynamic Analysis Part, Seeing the term 'cuckoo' in place of username, we were able to learn about Cuckoo Sandbox which generates API trace reports. We referred to these papers [4][5] where use of Cuckoo Sandbox in Malware Detection was referred. We learnt about n-api-call-grams, which referred to tracking n api calls together, Ex: Let A1, A2, A3, A4, A5 be 5 calls and we need 3-grams, then we will count A1A2A3, A2A3A4, A3A4A5 each as a separate object to store in dictionary. This type of analysis is done because, not only presence of functions but their relative position and order of call also affects the overall execution of the binary.

5 Using Pandas

We stored all these dictionaries in Pandas Data-Frames and concatenated them later. For the strings part, we only kept those strings which occurred in above a certain threshold number of files as there are too many strings and assuming low prediction power of such strings, they were dropped. The threshold was decided by hit-and-trial to be about 300 for Benign data and 100 for each class of Malware. Such a threshold was also made for api-call-grams which greatly helped in reduction of features not appearing above in a certain number of files.

6 Using SciKit-Learn

For feature reduction, RFE (Recursive Feature Elimination) and PCA (Principal Component Analysis) techniques were time consuming and were not feasible for training purposes. From about 8000 features, we used SelectKBest using 'chi2' function to get 3000 independent features, important to our model. We divided feature data into to parts with testing size = 0.25. Before training, we used MinMaxScaler to scale all our data, so, that there is no issue with convergence of 'lbfgs'. Then, we used a Logistic Regression classifier with multi-class as 'ovr' for binary classification and default solver as 'lbfgs'. We also used Support Vector Machines and Decision Trees algorithms, but Logistic Regression gave a better score.

7 Results

The model gives a minimum of score of 2500 files:

- Accuracy: 0.9984
- Precision: 0.9969
- Recall: 1.000
- F1 : 0.9984

which corresponds to 4 false positives along with 1274 true positives and 1222 true negatives.

8 Important Libraries Used

- Numpy
- Pandas
- SciKit - Learn
- Glob
- Pickle
- Joblib

9 Conclusion

The problem statement was very interesting. We learnt a lot about malware detection and how machine learning tools can catch malware with zero-day vulnerabilities which evade signature-based tools.

References

[1] https://blog.kowalczyk.info/articles/pefileformat.html

[2] https://resources.infosecinstitute.com/2-malware-researchers-handbook-demystifying-pe-file/

[3] https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1624&context=etd_projects

[4] https://www.sciencedirect.com/science/article/pii/S1319157817300149/

[5] https://www.researchgate.net/publication/262169869_Behavioural_detection_with_API_call-grams_to_identify_malicious_PE_files