Malware Detection Using ML Libraries
By Hackermen69

Introduction

The problem statement required us to implement a neural network that could classify files as benign or malicious based on the provided static and dynamic analysis data. The neural network has been built using Tensorflow Machine Learning Library.

File Information

Here is a description of the file contents of our submission-

- ➔ **MalwareDetection.py** - This is the main deliverable. Contains all the code right from feature selection, extraction, and the neural networks.
- ➔ **README.md** - This is the readme file for the code.
- ➔ **Hackathon_Documentation.pdf** - Current file. This contains the documentation for our method used and other details.
- ➔ **wts_stat.h5** - Weights for Static analysis.
- ➔ **wts_dy.h5** - Weights for Dynamic analysis.
- ➔ **output.csv** - Contains sample outputs as expected.

Steps followed

- **Data Collection**
- **Feature extractions & selection**
- **Classification**

Observations

An overview of each step is provided as follows :-

Data Collection

Collected and Examined Static and Dynamic Analysis Data for Malware and Benign samples provided in the database.

Feature Extraction and Selection

Extracted the different features from the database using python script and selected the most important ones. Different approaches were adopted for static and dynamic analysis data and are discussed below.

Static Analysis

The features were divided into various parts using the sections in the file structure_info.txt.

The main features used, based on sections are as follows:

GENERAL_INFO

- Virtual size -> sum of all virtual sizes
- Imported functions ->Number of [IMAGE_IMPORT_DESCRIPTOR]
- Presence of debug section ->Bool
- Resources ->Bool
- Relocations ->Bool->IMAGE FILE RELOCS STRIPPED
- Number of Symbols -> File Header-> NumberOfSymbols

This is basic information obtained from the PE header.

HEADER_INFO

- Timestamp ->File_header->TimeDateStamp
- List of Image Characteristics (list of strings) ->File_header FLAGS

From the **optional header** we obtain the following:

- Magic Number ->Magic

• Major Image Version ->MajorImageVersion

• Minor Image Version ->MinorImageVersion

• Major Linker Version ->MajorLinkerVersion

• Minor Linker Version ->MinorLinkerVersion

• Major System Versions ->MajorOperatingSystemVersion

• Minor System Version ->MinorOperatingSystemVersion

• Code Size ->SizeOfCode

• Header Size ->SizeOfHeaders

• Commit Size ->SizeOfHeapCommit

Features involving strings are first converted to their byte representation and then parsed through the FeatureHasher to get 10 bins of summarized data. This ensures that the data is vectorized and is of a fixed size

IMPORTED_Funcs

This data is summarized using the FeatureHasher. 256 bins are used for summarizing all unique libraries and 1024 bins are used to represent every library:FunctionName pair.

SECTION INFORMATION

• Section Name ->IMAGE_SECTION_HEADER Name

• Virtual Size ->Misc_VirtualSize

• Size ->SizeOfRawData

• Entropy ->Entropy

• Section Characteristics (list of strings) ->FLAGS

The hashing trick is used on the Name:Value pairs. The name being the section name, and the values being section size, section entropy and virtual size, each paired individually with a name. These pairs are summarized using the FeatureHasher with each value set allotted 50 bins. Section characteristics are captured separately using the same hashing trick as mentioned previously.

Dynamic Analysis

The main features used were mainly based on the Signature Array list in the dynamic analysis file which included the complete data of the file tests. This list was mainly used to classify between malicious and benign files as both these files showed significant amounts of variance with respect to this parameter. FeatureHasher trick similar to Static Analysis was used in this as well, with the bin size as 50.

Classification

Used machine learning classifiers to train the classifiers using extracted features.

Static Analysis

The neural network was trained on 75% of the dataset and the rest was used for testing. Train accuracy was 80% on 50 epochs. We got an accuracy of 83% in the test dataset. We used Binary Cross-Entropy loss, with the Adam optimizer, with a batch size of 128.

Dynamic Analysis

The neural network was trained on 75% of the total dataset and the rest 25% was used for testing. The neural network gave an accuracy of 99.50% on 12 epochs of Neural Network. Loss function used was Binary cross-entropy loss with Adam as optimizer, with a batch size of 128.

Team Members

- Yatharth Goswami, IIT Kanpur, 191178
- Som Tambe, IIT Kanpur, 190847
- Atharv Singh Patlan, IIT Kanpur, 190200