CS6570 Secure Systems Engineering
Course Project

Akshith CS19B005, Shashank CS19B043

May 20, 2022

In this project, we implemented the research paper **Analyzing Hardware Based Malware Detectors**.

1 Brief Summary of the Research Paper

Hardware based detectors rely on Machine Learning classifiers to detect malware-like execution pattern based on Hardware Performance Counters information at run-time. The paper attempts to thoroughly analyze various robust machine learning methods to classify benign and malware applications, using hardware based detectors. In the software implementation, various ML classifiers, including `BayesNet, NaiveBayes, Logistic, MultilayerPerceptron, Stochastic Gradient Descent (SGD), SimpleLogistic, SMO, JRIP, OneR, PART, J48` were used to identify malware at run-time.

The paper also presents the results obtained from the hardware implementation of these algorithms, using Xilinx High-Level Synthesis (HLS) compiler (which translates high level code into HDL code).

2 Setting up the environment

The code files were written in a Linux system (Ubuntu). The commands shown below are also for the same. As part of this project, we implemented the ML classification algorithms in two different methods:

1. Using the python library `sklearn` (MalwareAnalysis.py). Another library `mlxtend` is also needed. These libraries can be installed using the commands:
   `$ pip install sklearn`
   `$ pip install mlxtend`

2. Using the APIs available in `python-weka-wrapper3` (MalwareAnalysis-weka.py).
   Libraries and Dependencies are installed using the commands provided for Ubuntu systems in the Installation Guide. This video might be helpful while downloading `Open-JDK` and setting the `JAVA_HOME` environment variable.
   Instead of creating a virtual environment (venv), as shown in the installation guide, we installed `python-javabridge` and `python-weka-wrapper3` using the commands
   `$ pip install python-javabridge`
   `$ pip install python-weka-wrapper3`

   Make sure to be consistent with the python (3+) and the `python-weka-wrapper3` versions, for the libraries to be compatible. Don't install `python-weka-wrapper`.

Other common python libraries like `numpy, matplotlib, pandas, seaborn` are also needed.

3 Executing the codes

The dataset file provided was renamed as "dataset.csv". This file should be present in the current working directory (same as the code files). Once the environments are set up (as explained in the previous section), the codes can be executed using the python3 interpreter.

1. To execute the `sklearn` version of the implementation, run the file "MalwareAnalysis.py"
   `$ python3 MalwareAnalysis.py`

2. Run the python script "convert.py" to extract the required columns from the dataset. This also creates a new csv file "data.csv", that is used by the file "MalwareAnalysis-weka.py" to run the classification algorithms.
   `$ python3 convert.py`
   `$ python3 MalwareAnalysis-weka.py`

We also included the ipynb file "MalwareAnalysis.ipynb" in the submission zip, which is easier to go through and understand, as relevant sections of code are placed in the same cell. Also, all the results of execution are already visible in the ipynb file, making it easy for verification. MultiLayerPerceptron can't be run in normal system, as it uses excess memory due to which the process gets killed (can be checked by reading the system logs using the command `$ grep Killed /var/log/syslog`. Hence, we commented MLP in the submitted file.

4 Inference and Analysis on the Results

1. **Implementation using classifiers from sklearn**:
   A graphical plot of accuracy (with each classifier, using different number of features) is obtained. This plot only shows the accuracies with `NaiveBayes, Logistic, MultilayerPerceptron, SGD, Decision Tree (J48)` classifiers, as they are only ones available directly as library functions.
   Similar to what is done in the research paper, we extracted the top 32 contributing features (the ones with the highest correlation value), and re-arranged the data with these features first.
   Each classifier is tested with different number of features (top 1, top 2, top 4, top 8, top 32). As expected, there is not much variation in the accuracies with different number of features for a classifer.

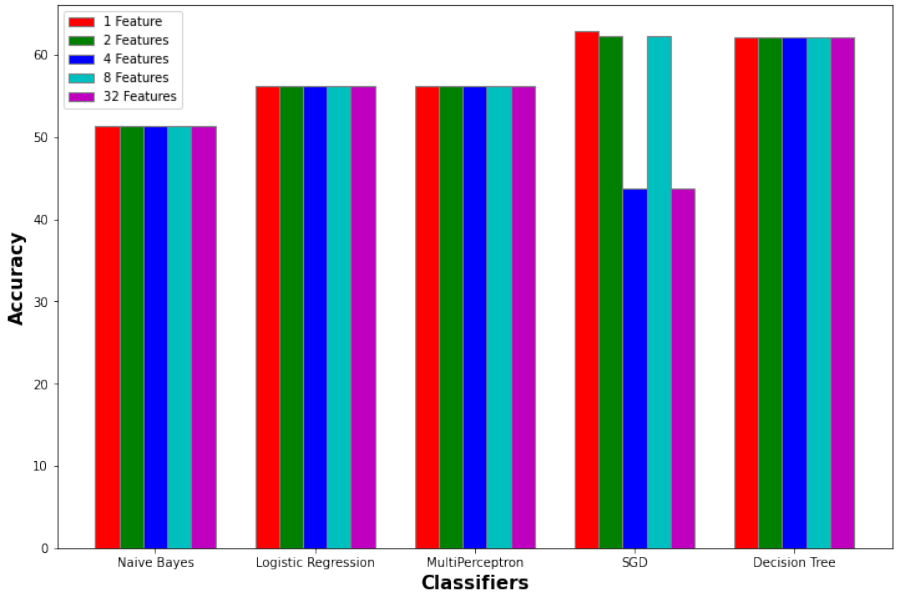

Figure 1: Plot of accuracies for different classifiers (sklearn) & features

   Among all the classifiers, SGD gave the highest accuracy, close to 63%.

2. **Implementation using classifiers from weka**:
   In this version, all the 11 classifiers are implemented. This program takes several hours to run. As a reason, we didn't run the classifiers for different number of features (1, 2, 4, 8, 32) as done in the previous version. Our program can be easily extended to run for different no. of features by placing the evaluation code in another function, and calling the function with different instances of "data" variable. Kindly look at the code for better clarity.
   The accuracies obtained using the python-weka-wrapper are significantly higher compared to those obtained in the sklearn implementation. For instance, the accuracy obtained with the **J48 classifier is 96.4%**.

   They also take much longer time to train & evaluate compared to the previous implementation.
   The reason for the long time might be due to the fact that the wrapper uses `JVM` in the background, and not to mention the *large size* of the dataset.

The long time taken to run the classifiers calls for hardware accelerated implementations, to boost performance & decrease the execution time.

5 References

1. For the sklearn implementation, references were taken from the official documentation available for the library.
2. References to implement the python-weka-wrapper3 APIs were taken from the official documentation and the official github repository.