HCL Hack 2020

Bug Slayers

Static Analysis

Workflow

Feature extraction is the most crucial step in any classification problem, and these were given to us in the form of PE File. We figured that an effective method would be to select our features from the header data given in these PE Files.

Hence we extracted the features from the header information given and parsed the data into a CSV file for training the Machine Learning model. The values of all features were given in the hexadecimal system, so we converted and encoded them all into integers of the decimal system.

Results and Observations

- We tried various classification Classification algorithms on our training set each giving drastically different Accuracies and F2 Score ranging from 56% (in case of Naive Bayes Classifier) to 97.9%(Random Forest)
- We used the Scikit learn library to apply Principal Component Analysis for feature reduction to mere 82 features.
- As Random Forest seems most precise on testing on the test dataset , we further fine tuned the model and set n_estimators to 10 after an iterative trial and error process for optimal results.
- The confusion matrix for our model turned to be as follows:

  961 13

  31 1129

- **Accuracy - 0.979**

- **F2 Score- 0.976**
- **False Positive = 0.61%**
- **False Negative = 1.45%**

- As seen from the above stats, we just got 13 False positives predictions from over 2000 testing data points which proved that our model performed exceedingly well.

Dynamic Analysis

Workflow

Usually, dynamic analysis involves running the particular file of interest, and then analyzing its behaviour during runtime. Here, we are provided with a dataset that consists of a report generated by running the file in the Cuckoo Sandbox.

In our implementation, we are using “API Call Frequencies” as features that our Machine Learning algorithms train on.

In order to do this, we do the following:

1. Extract the relevant calls to api from each of the Cuckoo reports.
2. Encode each api call name into an integer from a string
3. List data in a CSV format containing all of the information regarding the frequency of each call.
4. Run the classification problem through machine learning algorithms with the dataset obtained from the CSV files.

It is important to note that we used a list of API calls that was extracted from the dataset itself. A separate script to do this was written. Furthermore, we utilized some of the calls from the windows api documentation.

Results and Observations

- We tried various classification Classification algorithms on our training set, each giving drastically different Accuracies and F2 Score ranging from 42% (in case of Logistic Regression Classifier) to 95.6%(Random Forest). 3 layered ANN performed well too(91.2%).
- We used the Scikit learn library to apply Principal Component Analysis for feature reduction to mere 45 features.
- As Random Forest seems most precise on testing on the test dataset , we further fine tuned the model and set n_estimators to 12 after an iterative trial and error process for optimal results.
- The confusion matrix of our model turned to be as follows:

  602 09
  43 400

- **Accuracy - 0.956**
- **F2 Score - 0.952**
- **False Positive = 0.86%**
- **False Negative = 4.11%**

- As seen from the above stats, we just got 9 False positives predictions from over 1500 testing data points which proved that our model performed exceedingly well.

File structure for Test Directory

```
Testing_data/
├──── Dynamic_Analysis_Data
│    ├──── benign
│    │    └──── 3606c445416d6baf10e7416ee0c1eea825cedbad490b4f2c203731e8b077fea1.json
│    └──── malware
│         ├──── backdoor
│         │    └──── 3606c445416d6baf10e7416ee0c1eea825cedbad490b4f2c203731e8b077faa.json
│         ├──── trojan
│         │    └──── 3606c445416d6baf10e7416ee0c1eea825cedbad490b4f2c203731e8b077febb.json
│         ├──── trojandownloader
│         │    └──── 3606c445416d6baf10e7416ee0c1eea825cedbad490b4f2c203731e8b077fecc.json
│         ├──── trojandropper
│         │    └──── 3606c445416d6baf10e7416ee0c1eea825cedbad490b4f2c203731e8b077fedd.json
│         ├──── virus
│         │    └──── 3606c445416d6baf10e7416ee0c1eea825cedbad490b4f2c203731e8b077feee.json
│         └──── worm
│              └──── 3606c445416d6baf10e7416ee0c1eea825cedbad490b4f2c203731e8b077feff.json
└──── Static_Analysis_Data
     ├──── benign
     │    └──── 0a21ef18ba03622736a8edd5390afbab6088dcacc3d5877eb0b28206285f569d
     │         ├──── String.txt
     │         └──── Structure_Info.txt
     └──── malware
          ├──── backdoor
          │    └──── 0a21ef18ba03622736a8edd5390afbab6088dcacc3d5877eb0b28206285f569a
          │         ├──── String.txt
          │         └──── Structure_Info.txt
          ├──── trojan
          │    └──── 0a21ef18ba03622736a8edd5390afbab6088dcacc3d5877eb0b28206285f569b
          │         ├──── String.txt
          │         └──── Structure_Info.txt
          ├──── trojandownloader
          │    └──── 0a21ef18ba03622736a8edd5390afbab6088dcacc3d5877eb0b28206285f569c
          │         ├──── String.txt
          │         └──── Structure_Info.txt
          ├──── trojandropper
          │    └──── 0a21ef18ba03622736a8edd5390afbab6088dcacc3d5877eb0b28206285f569f
          │         ├──── String.txt
          │         └──── Structure_Info.txt
          ├──── virus
          │    └──── 0a21ef18ba03622736a8edd5390afbab6088dcacc3d5877eb0b28206285f569g
          │         ├──── String.txt
          │         └──── Structure_Info.txt
          └──── worm
```

```
└──── 0a21ef18ba03622736a8edd5390afbab6088dcacc3d5877eb0b28206285f569h
    ├──── String.txt
    └──── Structure_Info.txt
```