Hierarchically Learned Malware Identification & Classification Model

Furkan Özgültekin, Mail: fozgultekin19@ku.edu.tr, ID: 72049

Ayten Dilara Yavuz, Mail: ayavuz19@ku.edu.tr, ID: 72281

Ahmet Başar Ertürk, Mail: aerturk19@ku.edu.tr, ID: 71742

Introduction

In this current digital age, cybersecurity is getting more and more important due to adversary attacks becoming more advanced each day. One of the most common cybersecurity issues is malware. Malware stands for “malicious software” and it has various types such as viruses, worms, trojans, spywares, ransomwares, backdoors, and scarewares. These malware have specific features that distinguish them from each other. When a device gets infected with malware, the user's personal information, sensitive information and financial security is compromised. In order to prevent the harm from malware, they need to be classified before taking preventative actions. Malware classification has 2 phases. First is identifying whether the software is “malware” or “benign”. Second phase is classifying the malware’s family. Security methods like firewalls and antivirus softwares are a very common approach to classify malware but they are not as effective anymore because malwares are getting more advanced and more difficult to be “found”. Nowadays, Machine Learning (ML) is used for solving cybersecurity problems, including malware detection.

State-of-the-art malware classification models are advanced ML algorithms that are designed to detect and classify malware. These models use various techniques such as deep learning, neural networks, and feature extraction to analyze the behavior and characteristics of malware samples. These current ML models utilize both classification and identification of malware in a single model or only take into consideration either malware detection or classification [1].

Deep learning technique is a subfield of machine learning that uses neural networks to process and analyze large amounts of complex data. These neural networks are made up of layers of interconnected nodes, called artificial neurons, that are trained to perform specific tasks. In the context of malware classification, deep learning can be used to analyze the behavior and characteristics of different types of malware and accurately classify them into different categories. One of the main advantages of using deep learning in malware classification is the ability to automatically learn and extract features from the raw data, without the need for manual feature engineering. This can make the process of identifying new and unknown malware much

more efficient and accurate. It's worth noting that deep learning approaches are computationally expensive and require large amounts of labeled data to train, which can be a limiting factor, but with the increasing computational power and the amount of labeled data, this approach has become more and more popular in the field of malware classification.

Similarly to deep learning, hierarchical learning is also a technique of ML that is being used for malware detection. It involves the use of multiple levels or layers of models to perform a task. The advantage of using a hierarchical approach in malware classification is that it allows the detection of malware at different levels of abstraction, which make it possible to detect both known and unknown malware, and also to improve the performance of the classifier. Overall, the hierarchical learning approach to malware classification can be more accurate and robust than using a single model or single analysis technique.

Problem Statement

ML models usually specify only one of the following about a program: Whether they are a malware or not, and given they are malware, which family the malware belongs to. Given a program, it is sometimes not enough to know if it's malicious or benign. For example, if you have already run the program or you just discovered you had malware running on your computer for some time, you may want to learn what damage the malware caused or its behavior. Trying to find which malware family a program belongs to without first being sure it is malicious can also be problematic.

The proposal of this project is to use a pipeline of two machine learning models, rather than just one, to mitigate potential issues with retraining and constant drift caused by new malware sub-families. This approach is expected to result in a more robust model. The goal of using this type of approach is to improve training efficiency and accuracy by using a more easily understandable model. This aims to highlight the significance of both model understandability and scalability in this problem, as the volume of data is expected to constantly increase in an open-world scenario.

Proposed Approach

The pipeline of this project's proposed models would be as such (from top to bottom):

Model M: {File} -> {"Benign", "Malware , Family"}

1. Identification of a file M1:
    a. M1: {File} -> {"Benign","Malware"}
        i. If "Benign": returns "Benign"
        ii. If "Malware": pass through M2
2. Classification of a Malware M2:
    a. M2: {File (known to be malware)} -> {"Family-1",..., "Family-k", "OOD"}
    b. M2 might have two inner pipelines that can help mitigate smaller sized samples with two/three different models (one catered towards smaller samples and one for bigger sampled data) that support ensemble learning.

3. Output:
    a. returns Benign or M2

Wherein an Asymmetric Hierarchical Model is prevalent for malware classification, we hypothesize that this model would be advantageous to a single model for both malware identification and classification due to new out of scope malwares possibly having similarities to other malware but having a result out of distribution. This model could result in a reduction of false positives coming from actual benign software possibly having out of distribution classification when compared to previous models.

Furthermore this model can be optimized more wherein such that if the labels of the dataset contain specific types of "worms" "trojans" etc. it can first hierarchically classify that it is a "worm" then assess if its a P2P-worm or an IRC-worm.

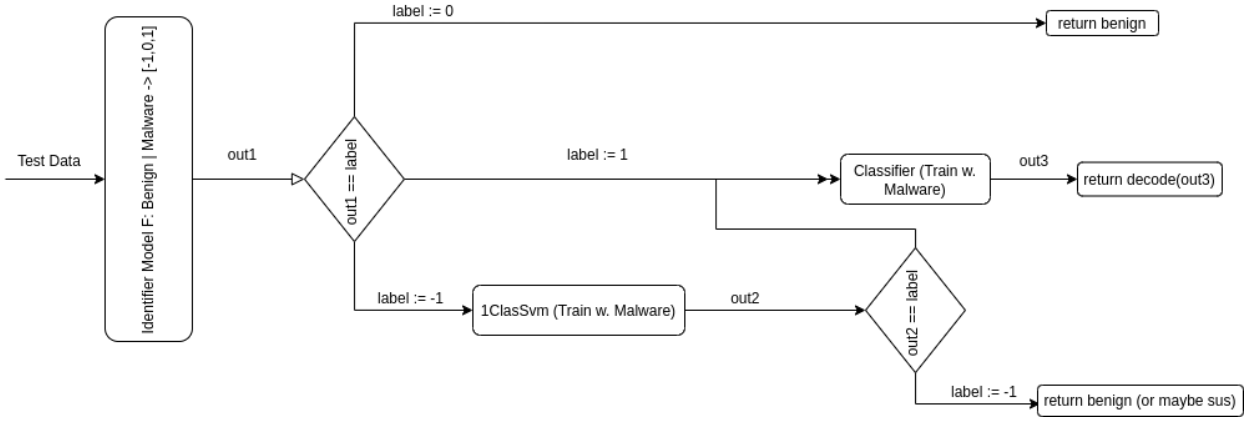

Fig. 1: Pipeline of the Proposed Model

Methodologies, Tools, and Dataset

Training Methodology: Due to there being two distinct models, the training is disjointed where

1. Model M1: Trained with All of the training data.
2. Model M2: Intentionally trained with some of the training data that is labeled as Malware for the purpose of having a part of the test data out of the distribution of the training labels.
3. An existing state of the art model: Also intentionally trained with some of the data that is labeled as Malware for the purpose of having a part of the test data out of the distribution of the training labels.

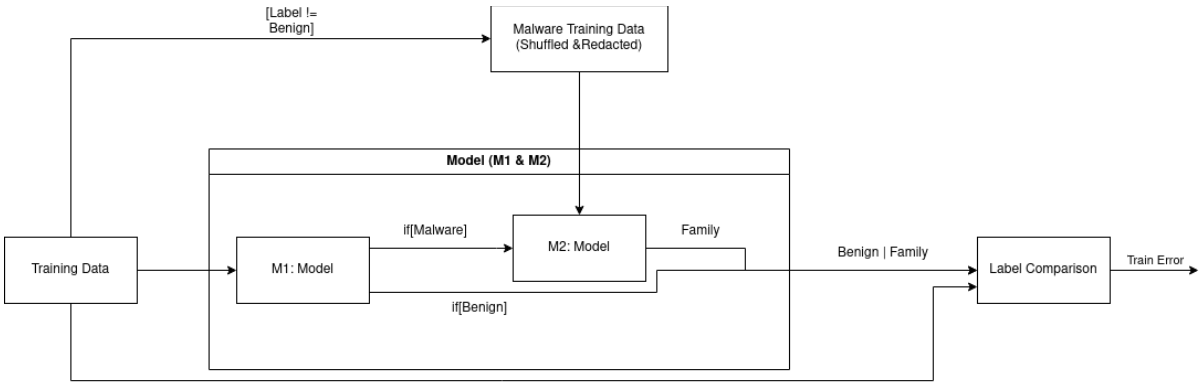

Fig. 2: The Flow Chart of Training for the Proposed Model

In this project, we trained our models with a deep learning approach, running for 5000 epochs.

After training and testing the model, another training model is constructed to improve results where the classification model is trained with only a few selected malware family data. This way, the accuracy of the classification model improved since there were a smaller number of families to choose from in this version.

Testing Methodology: The Test Data contains all the labels of malware and also benign labeled data such that the data is independent of the training data. It is also assumed that training has been completed and M2 contains partial information about the malware families that exist in the Test Data, whilst calculating the error, Out of Distribution (OOD) predictions are accepted as true if the test data label is not in M2.

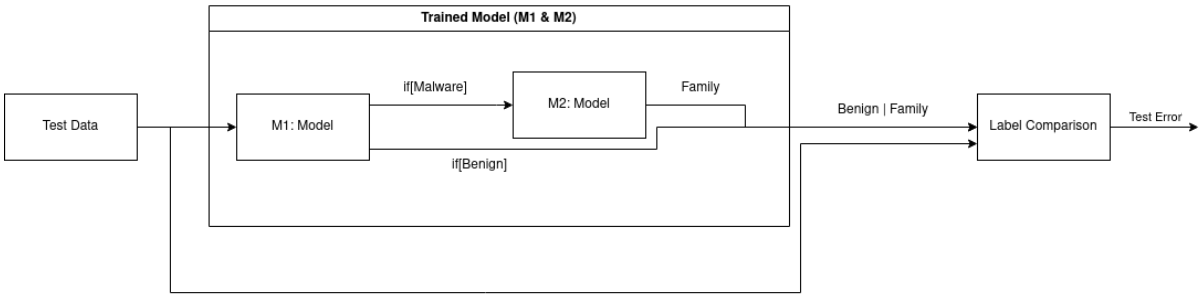

Fig. 3: The Flow Chart of Testing for the Proposed Model

Dataset: The dataset is a combination of benign and malware labeled data with malware data being subdivided into several families of data. The data is divided into training data and test data with approximately 80% of the Dataset being set to Training and 20% of the data being divided into Test data. In this project, the dataset used to train and test the models is the EMBER dataset which is used in static analysis. EMBER dataset is a dataset consisting of labeled samples of Windows portable executable files, to be used as a benchmark for training machine learning models to detect malware through static analysis. The EMBER dataset has to be downloaded and unzipped in the Datasets folder. To download EMBER, the following link should be used: https://ember.elastic.co/ember_dataset_2018_2.tar.bz2. In order to train the model, the data is first vectorized. The labels of the data in EMBER is as follows:

900K train: 300K malware 300K benign 300K unlabeled
200K test: 100K malware 100K benign

Benchmarking: The existing model is compared to the proposed model, the performances of both are assessed in terms of training time, classification accuracy and false positives/negatives. The possibility of threading might be explored for our proposed model and be compared that way as well. Initially, this was our aim for benchmarking. However, after observing the testing accuracy, we developed a different approach. See the Discussion & Evaluation part for more.

Tools: In this project, machine learning Python package PyTorch is used.

Hypothesis: We hypothesize that with introduction of new labels (malware–in the open world assumption) training can be faster in our Model if only M2 is trained for new labels and the drift with new malware introductions will marginally decrease with new types of malware being introduced. We also believe that this kind of a model can support asynchronous training for M1 and M2 at the same time, also decreasing the training time.

Running the Malware Classification Model: First, run model_install.sh (or run the commands inside of it starting from top in the terminal). Make sure you have downloaded and unzipped the dataset in a folder named "Datasets": (https://ember.elastic.co/ember_dataset_2018_2.tar.bz2 specifically). Then, run model_run.sh (or run the commands inside of it starting from top in the terminal). Finally, run the pipeline to test.

Results

After performing the training, the following loss vs epoch values (shown in Figure 4 and 5) were obtained for the malware identifier model and malware classifier model respectively:

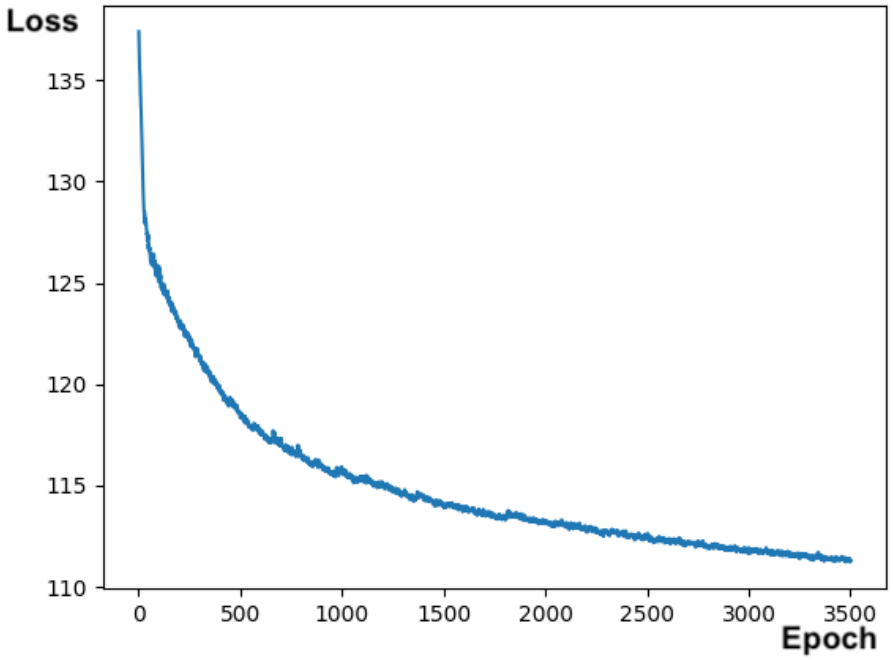

Fig. 4: Loss vs Number of Epochs Graph for Malware Identifier Model

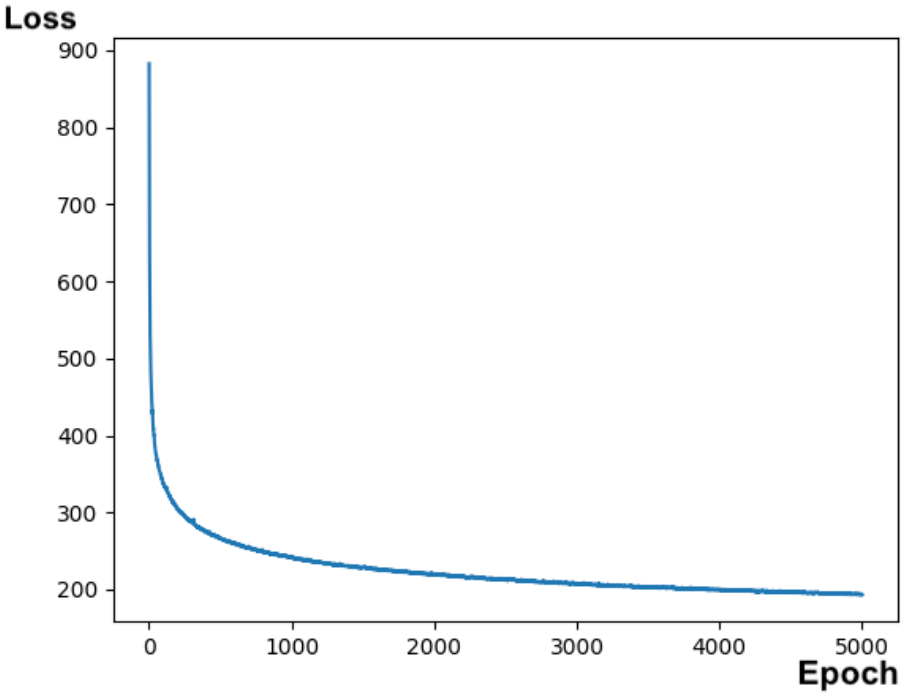

Fig. 5: Loss vs Number of Epochs Graph for Malware Classifier Model

Results of testing the models areas follows:

Test accuracy for Identifier: 0.354415

Test accuracy for Classifier with all families data: $1x10^{-5}$
Test accuracy for Classifier with selected families data: 0.01162

Test accuracy for Isolation Forest (Replacement for SVM): 0.995115

Test accuracy for the overall pipeline with all families data: 0.00299
Test accuracy for the overall pipeline with selected families data: 0.00051

They can be seen in the appendix as well.

Discussion & Evaluation

Even though the accuracy values are low, the shortcoming of this method is that all of the models in the pipeline needs to label the input correctly in order to obtain the final correct output. Probably in many cases, either one of the models were successful at labeling the input but the other/next model was not; or similarly, because the first model was not successful, the following model had no chance at being successful. All of these examples are counted as misclassification in our model. Therefore, we decided to benchmark the accuracy of our pipeline against the accuracy of the classifier model alone, since the pipeline is our proposal for the improvement of malware classification.

When we do that, it can be seen that accuracy increased from $1x10^{-5}$ in classification alone to 0.00299 in the proposed pipeline model, which means nearly 300x better accuracy. However, the accuracy of the overall pipeline unexpectedly decreased when the classifier model was trained on limited malware families data, even though the accuracy of the classifier model increased. The possible reason for this is that since most of the malware data were filtered off in this part, at test time there was an overload of benign malware inputs that caused the overall accuracy of the model to heavily depend on the first model in the pipeline. Moreover, because the accuracy of the first model is not very high as well, any misclassification on that model inevitably ended up as a misclassification.

Conclusion

In this project, a hierarchically learned malware identification and classification model is proposed as opposed to state of the art single ML model malware classifiers. Our hypothesis was that by only training the M2 model on new labels (malware–in the open world assumption), the model will be able to learn faster and the impact of new malware introductions on the model will be minimized. Additionally, this approach allows for concurrent, asynchronous training of both the M1 and M2 models, further reducing training time. However, the shortcoming of the proposed model is that all models in the pipeline must label the input correctly to obtain the correct output. As a result, this lowers the accuracy of the overall pipeline, so the accuracy of the pipeline model is benchmarked against the accuracy of the classification

model alone. After analyzing the results, it can be said that the proposed pipeline method improves the accuracy of malware classification by 300 times compared to a single classifier model.

APPENDIX

Appendix A: Accuracy result with data including all of the malware families:

```
**************************************************
Testing Identifier
/home/furqn/.local/lib/python3.10/site-packages/torch/utils/data/_utils/collate.py:172: UserWarning: The given NumPy
array is not writable, and PyTorch does not support non-writable tensors. This means writing to this tensor will
result in undefined behavior. You may want to copy the array to protect its data or make it writable before
converting it to a tensor. This type of warning will be suppressed for the rest of this program. (Triggered
internally at ../torch/csrc/utils/tensor_numpy.cpp:199.)
  return collate([torch.as_tensor(b) for b in batch], collate_fn_map=collate_fn_map)
Test accuracy:  0.354415
WARNING: EMBER feature version 2 were computed using lief version 0.9.0-
WARNING:   lief version 0.12.3-39115d10 found instead. There may be slight inconsistencies
WARNING:   in the feature calculations.
DISCLAIMER: Extracting malware only data, which might take a minute...
**************************************************
Testing Classifier
Test accuracy:  1e-05
**************************************************
Testing Isolation Forest
Test accuracy:  0.995115
**************************************************
Testing Pipeline
Test accuracy:  0.00299
```

Appendix B: Accuracy result with data that consists of only selected malware families

```
**************************************************
Testing Identifier
/home/furqn/.local/lib/python3.10/site-packages/torch/utils/data/_utils/collate.py:172: UserWarning: The given NumPy
array is not writable, and PyTorch does not support non-writable tensors. This means writing to this tensor will
result in undefined behavior. You may want to copy the array to protect its data or make it writable before
converting it to a tensor. This type of warning will be suppressed for the rest of this program. (Triggered
internally at ../torch/csrc/utils/tensor_numpy.cpp:199.)
  return collate([torch.as_tensor(b) for b in batch], collate_fn_map=collate_fn_map)
Test accuracy:  0.354415
WARNING: EMBER feature version 2 were computed using lief version 0.9.0-
WARNING:   lief version 0.12.3-39115d10 found instead. There may be slight inconsistencies
WARNING:   in the feature calculations.
DISCLAIMER: Extracting malware only data, which might take a minute...
**************************************************
Testing Classifier
Test accuracy:  0.011620040590552747
**************************************************
Testing Isolation Forest
Test accuracy:  0.995115
**************************************************
Testing Pipeline
WARNING: EMBER feature version 2 were computed using lief version 0.9.0-
WARNING:   lief version 0.12.3-39115d10 found instead. There may be slight inconsistencies
WARNING:   in the feature calculations.
Test accuracy:  0.00051
```

References

[1] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, “Drebin: Effective and Explainable Detection of Android Malware in Your Pocket.” [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2017/09/11_3_1.pdf