AI and CyberSecurity Midterm-2021 Fall

Bibek Upadhayay

Problem Statement

In this midterm project, are going to train a deep neural network based on the MalConv architecture to classify PE files as malware or benign. We are going to use the Ember dataset to train the model. After the model is trained we will be deploying the model to the AWS cloud using AWS Sagemaker and create an endpoint such that we can access it via the Client PE program to check the PE files locally.

Introduction

PE Files

PE (Portable Executable) format is one COFF (Common Object File Format (COFF) format available for executable, object code, DLLs, FON font files, and core dumps in 32-bit and 64-bit versions of Windows operating systems. The PE file block diagram is given in figure 1. PE format is simply a data structure that tells Windows OS loader what information is required to manage the wrapped executable code. The PE data structures include DOS Header, DOS Stub, PE File Header, Image Optional Header, Section Table, Data Dictionaries, and Sections.

But, why do we care about PE format when it comes to malware classification?

PE file codes can be changed easily which makes them vulnerable to malicious code. The PE files can be easily injected with malicious code, some of the examples are Trojans, backdoors, ransomware worms, and APT which are achieved by infecting PE files. Once the PE files are infected, the malware takes action without approval from the user. Hence, by inspecting the PE files one can identify whether it is malicious or benign. But can we automate the process? Yes, use machine learning

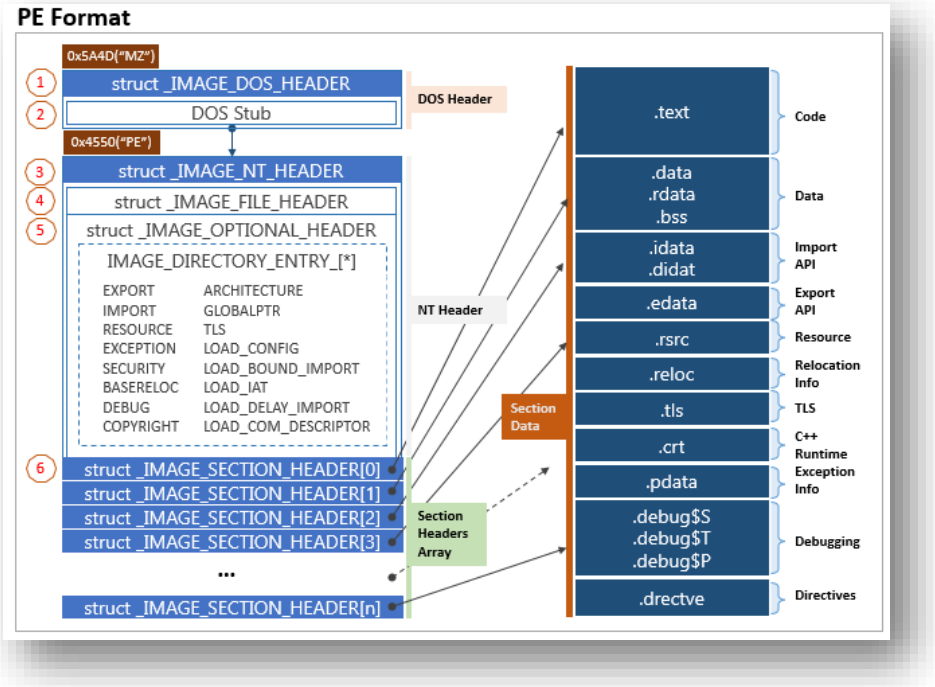

Fig: Portable Executable files and blocks (Source)

MalConv

One way to automate the classification of PE files is to use a machine learning approach. The deep neural network methods can be implemented for the detection approach, however, Edward et. al. proposed the MalConv architecture that tackled the problem of presenting raw bytes sequentially with linear complexity dependence which means that the model is capable of scaling with sequence length. In addition to that using the convolution layer also captures both the local and global context of examining the overall binary file. The model uses the convolutional activation with a global max-pooling that allows the model to produce its activation regardless of the location of the detected features. This resolves the problem of positional variance in executable files where the PE binary can be rearranged in any arbitrary orderings. The model architecture for the MalConv is given in figure 2

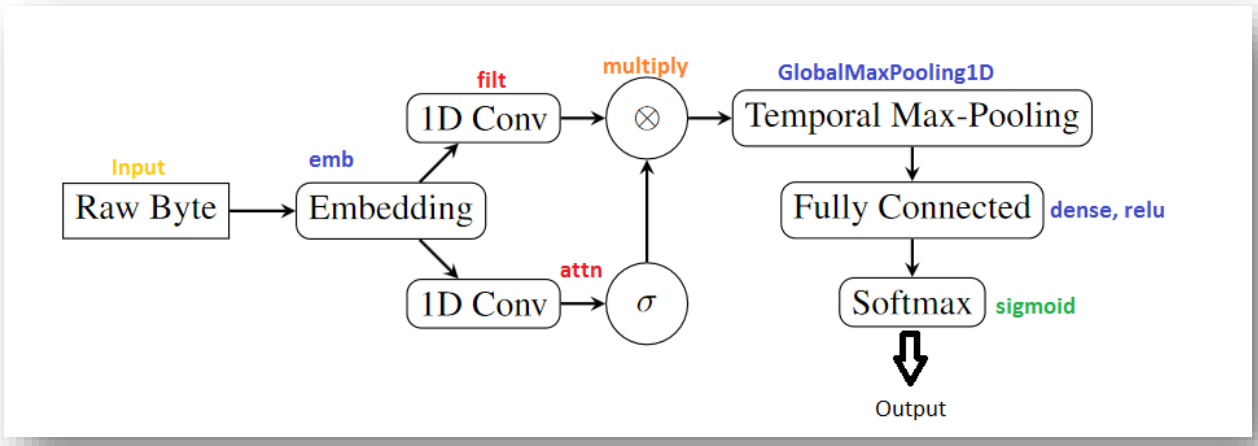

Fig 2: MalConv Model Architecture with TensorFlow layers (Source)

Dataset

Ember stands for Elastic Malware Benchmark for Empowering Researchers which is a collection of features from PE files that serve as a benchmark for researchers. The EMBER2017 dataset contains features from 1.1 million PE files. The dataset can be found on GitHub. The dataset also comes with the functionality to extract the features from PE files that are based on the LIEF project.

Methodology

In this project, we started with training the MalConv model on the Ember2018 dataset and deploying it in AWS Sagemaker to test the model via the client files. There are numerous steps in this process which are described as below:

Step 1: Data Processing

The first step in data processing is to use the LIEF feature extractor. The features extracted are then normalized using MinMax Scaler. There is a 20K dataset that is not labeled, hence those data were discarded.

Step 2: Training the MalConv Model

The MalConv model is set up using the TensorFlow sequential layers. The model summary is given in figure 3. The small difference is that in this model we make use of Sigmoid instead of softmax. The model is trained on a total of 1,059,945 parameters. In our model, we use the kernel and strides both of 500. To calculate the loss we use binary cross-entropy loss and a learning rate of 0.001

```
Model: "model_5"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_9 (InputLayer)         [(None, 2381)]            0
_________________________________________________________________________________________________
embedding_6 (Embedding)      (None, 2381, 8)           19048       input_9[0][0]
_________________________________________________________________________________________________
conv1d_12 (Conv1D)           (None, 4, 128)            512128      embedding_6[0][0]
_________________________________________________________________________________________________
conv1d_13 (Conv1D)           (None, 4, 128)            512128      embedding_6[0][0]
_________________________________________________________________________________________________
multiply_6 (Multiply)        (None, 4, 128)            0           conv1d_12[0][0]
                                                                   conv1d_13[0][0]
_________________________________________________________________________________________________
global_max_pooling1d_6 (GlobalM (None, 128)            0           multiply_6[0][0]
_________________________________________________________________________________________________
dense_14 (Dense)             (None, 128)               16512       global_max_pooling1d_6[0][0]
_________________________________________________________________________________________________
dense_15 (Dense)             (None, 1)                 129         dense_14[0][0]
=================================================================================================
Total params: 1,059,945
Trainable params: 1,059,945
Non-trainable params: 0
_________________________________________________________________________________________________
```

Fig 3: Model summary of MalConv model.

Step 3: Saving the Model and Creating client PE file

We save the model to JSON object along with its weight. We also created a client function and a file that can be used to test whether the PE file is malicious or benign. The client function can be used to test the PE file inside the notebook whereas the ClientPE python file is used to connect to the AWS Sagemaker endpoint to test the deployed model.

In below figure 4, we can see all the files in the respective folder.

```
build/              malConvTrained.data-00000-of-00001   resources/
checkpoint          malConvTrained.index                 scripts/
data/              'malconv- try 1.ipynb'                setup.py
dist/               malModel.json                        x_test_df
ember/              model_1.h5                           x_test_df.h5
ember.egg-info/     model2.h5                            x_train_df
example.csv         model_weight_1.h5                    x_train_df.h5
jcpicker.exe        newSaved/                            y_test_df
licenses/           putty.exe                            y_test_df.h5
LICENSE.txt         README.md                            y_train_df
malconv/            requirements_conda.txt               y_train_df.h5
malconv2.ipynb      requirements_notebook.txt
malconv.h5          requirements.txt
```

Figure 4: Displaying all the files

When we run the ClientPE file on ‘putty.exe’, the model output it as a benign file.

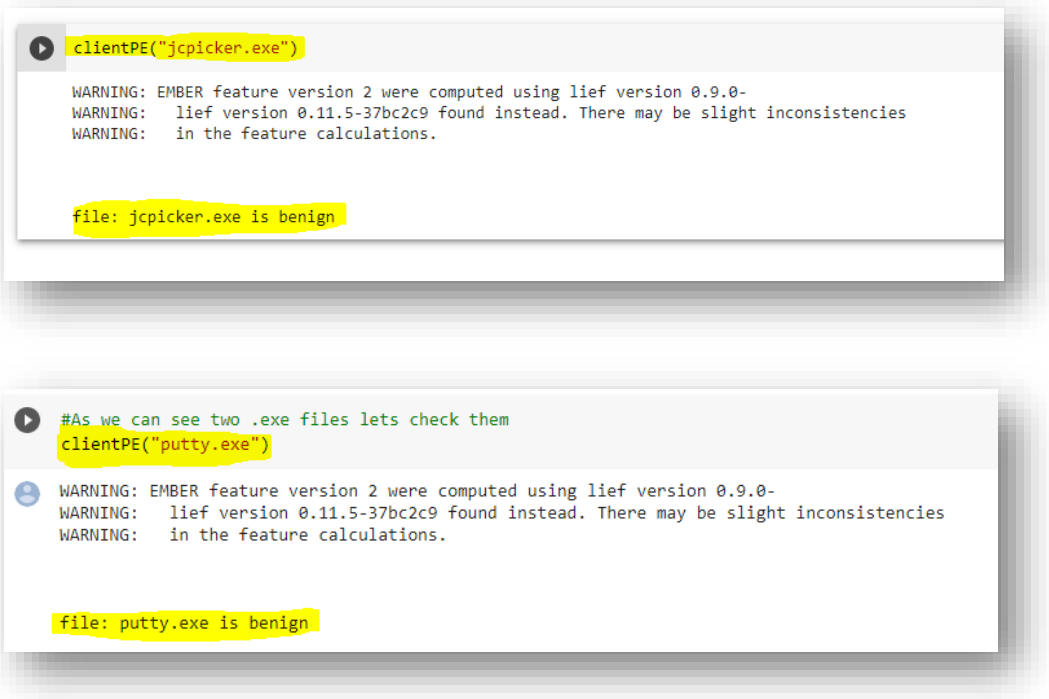

Figure 5: Displaying the result of the model on PE file

Step 4: Deploying the model in AWS SageMaker

After our model has been saved, we deploy our saved model in the AWS Sagemaker. We created an endpoint 'sagemaker-tensorflow-serving-2021-10-13-23-15-18-514' and call this endpoint in our ClientPE.py file using AWS access keys.

Results and Discussion

The confusionMatrix is given below:

	Pred Benign	Pred Malicious
Is Benign	64693	35307
Is Malicious	15178	84822

Figure 6: Confusion Matrix

Our trained model achieved an accuracy of 75% in a testing dataset with an F1 score of 77.065. The confusion matrix of the model can be observed in figure 6. The False-positive number is around 35K which tells us that the model is predicting even the benign files as malware. This might be the similar nature of the PE files in both malware and benign files. Perhaps, using the different hyperparameters can be used to increase the accuracy and make the results better. We also faced an issue calling a Sagemaker's endpoint, which might be the result of AWS academy's shorter session period.

Conclusion

In this project, we trained the MalConv architecture model using the Ember 2018 dataset and deployed the model in the AWS Sagemaker. As we can observe the better model performance using the MalConv model deploying the model in the cloud make it available for the end-users to find the malware files in their's PE file. The code is implemented in GitHub.

References:

1. https://github.com/elastic/ember
2. https://lief-project.github.io/
3. https://arxiv.org/abs/1710.09435
4. https://blog.reasonsecurity.com/2020/08/18/pe-classes-are-safe-its-your-business-thats-under-attack/
5. https://arxiv.org/abs/1804.04637