University of New Haven

Tagliatela college of Engineering

Masters in Data Science

Cloud-based PE Malware Detection API

Artificial Intelligence and Cyber Security

Date: 10/14/2021

Raja Muppalla

Under the guidance of,

Dr. Vahid Behzadan

VBehzadan@newhaven.edu

Contents

1. Purpose of the Project :

The purpose of this project research is to use machine learning models to detect and classify malware.
There are three parts in this project.
1. The first step is to use the Ember opensource dataset, EMBER-2017 v2, to train a deep neural network to identify PE files as malware or benign.
2. The second step is to upload the model to the cloud and create an endpoint (API) for it.
3. The final step is to construct a client that is nothing more than a python script that imports a PE file and determines if it is malware or benign.

2. Project Requirements :

In the project, we will first Access to Google CoLab and Amazon Sagemaker are required for this project. For this project, we believe that working in these services is a good idea.

3. Project Implementation :

This project is carried out in a task-oriented manner. As a result, the report's specifics are based on the tasks completed.

3.1 Training :

Data is extracted, vectorized, and preprocessed as the first step in this assignment. Following that, Keras is used to create a deep neural network architecture. On the extracted data, the model is learned, validated, and tested. To get improved model performance, hyperparameters are changed. The model is preserved, along with its weights. Finally, a method is developed to take a PE file and determine its type, such as benign or malicious.

The first step is the Data Extraction and Preprocessing the data which was created in the Jupyter notebook in Google Collaboratory.

Task_1_Training

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount(
e).
```

In [3]:
```
!wget https://dsci6015aisecf21.s3.us-east-2.amazonaws.com/X_train.dat
!wget https://dsci6015aisecf21.s3.us-east-2.amazonaws.com/X_test.dat
!wget https://dsci6015aisecf21.s3.us-east-2.amazonaws.com/y_train.dat
!wget https://dsci6015aisecf21.s3.us-east-2.amazonaws.com/y_test.dat
!wget https://dsci6015aisecf21.s3.us-east-2.amazonaws.com/metadata.csv
```

Copying downloaded files to Drive :

In [5]:
```
!cp /content/X_train.dat /content/drive/MyDrive/vMalConv/X_train.dat
!cp /content/X_test.dat /content/drive/MyDrive/vMalConv/X_test.dat
!cp /content/y_train.dat /content/drive/MyDrive/vMalConv/y_train.dat
!cp /content/y_test.dat /content/drive/MyDrive/vMalConv/y_test.dat
!cp /content/metadata.csv /content/drive/MyDrive/vMalConv/metadata.csv
```

Installing Ember dataset :

In [6]:
```
!wget https://github.com/endgameinc/ember/archive/master.zip
!unzip master.zip
!rm master.zip
!cp -r ember-master/* .
!rm -r ember-master
!pip install -r requirements.txt
!python setup.py install
```

```
--2021-10-14 23:00:26--  https://github.com/endgameinc/ember/archive/master.zip
```

vectorizing the features :

In [7]:
```
import ember
X_train, y_train, X_test, y_test = ember.read_vectorized_features("drive/MyDrive/vMalConv/")
metadata_dataframe = ember.read_metadata("drive/MyDrive/vMalConv/")
```

```
WARNING: EMBER feature version 2 were computed using lief version 0.9.0-
WARNING:   lief version 0.11.5-37bc2c9 found instead. There may be slight inconsistencies
WARNING:   in the feature calculations.
```

Get ridding of rows with no labels :

In [8]:
```
labelrows = (y_train != -1)
X_train = X_train[labelrows]
y_train = y_train[labelrows]
```

In [9]:
```
import h5py
h5f = h5py.File('X_train.h5', 'w')
h5f.create_dataset('X_train', data=X_train)
h5f.close()
h5f = h5py.File('y_train.h5', 'w')
h5f.create_dataset('y_train', data=y_train)
h5f.close()
```

In [10]:
```
!cp /content/X_train.h5 /content/drive/MyDrive/vMalConv/X_train.h5
!cp /content/y_train.h5 /content/drive/MyDrive/vMalConv/y_train.h5
```

Creating the architecture of MalConv in Keras :

In [11]:
```
def make_model():
  import tensorflow as tf
  from tensorflow import keras
  from tensorflow.keras import layers
  feature_size=2381
  tf.compat.v1.disable_eager_execution()

  keras.backend.clear_session()

  # Model architecture
  from tensorflow.keras import layers

  model = tf.keras.Sequential()
  model.add(layers.InputLayer(input_shape=(1,feature_size)))
  model.add(layers.Dropout(0.2))
  model.add(layers.Dense(1500, activation='relu'))
  model.add(layers.Dropout(0.5))
  model.add(layers.Dense(1, activation='sigmoid'))

  model.compile(tf.keras.optimizers.Adam(learning_rate=0.001),
            loss='binary_crossentropy',
            metrics=['accuracy',tf.keras.metrics.AUC(),tf.keras.metrics.Precision()])
  print(model.summary())
  return model
```

In [12]:
```
model = make_model()
model.save('./content/drive/MyDrive/vMalConv/model.h5')
```

Standard Scaler partial fitting to avoid overloading :

In [13]:
```
from sklearn.preprocessing import StandardScaler
mms = StandardScaler()
for x in range(0,600000,100000):
  mms.partial_fit(X_train[x:x+100000])
```

In [14]:
```
X_train = mms.transform(X_train)
```

In [15]:
```
## Reshape to create 3 channels ##
import numpy as np
X_train = np.reshape(X_train,(-1,1,2381))
y_train = np.reshape(y_train,(-1,1,1))
```

Model training :

In [16]:
```
%tensorflow_version 2.x
import tensorflow as tf
tf.compat.v1.disable_eager_execution()

save_dir = "drive/MyDrive/vMalConv/"

callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)

model.compile(tf.keras.optimizers.Adam(learning_rate=0.01),
            loss='binary_crossentropy',
            metrics=['accuracy',tf.keras.metrics.AUC(),tf.keras.metrics.Precision()])

history = model.fit(X_train, y_train,
                  batch_size=128,
                  epochs=30,
                    validation_split=.2,
                    callbacks=[callback]
                    )
# Save the weights #
model.save_weights (save_dir+'weights.h5')

# Save the model architecture #
model_json = model.to_json()
with open(save_dir+"model.json", "w") as json_file:
    json_file.write(model_json)

print("model saved.")
```

```
Train on 480000 samples, validate on 120000 samples
Epoch 1/30
```

Model Testing :

The model is put to the test using the test data, and the model's performance is determined. For later usage, the training mode is saved and uploaded to Google Drive. Create a function that accepts the PE files as an argument, runs them through the trained model, and returns the type of the PE files, Malware or Benign, as part of the model testing.

In [18]:
```
1
2 test_pefile("putty.exe")
```

```
WARNING: EMBER feature version 2 were computed using lief version 0.9.0-
WARNING:   lief version 0.10.0-845f675 found instead. There may be slight inconsistencies
WARNING:   in the feature calculations.
```

```
/usr/local/lib/python3.7/dist-packages/keras/engine/training.py:2470: UserWarning: `Model.sta
uture version. This property should not be used in TensorFlow 2.0, as `updates` are applied a
  warnings.warn('`Model.state_updates` will be removed in a future version. '
```

Out[18]:
```
array([[[0.5]]], dtype=float32)
```

Benign Sample

3.2 Deploy Model on the cloud :

WS Sagemaker is the cloud service used in this work. To deploy the model to Sagemaker, the saved model is uploaded and loaded. This resulted in the creation of an endpoint. Create a notebook instance in AWS Sagemaker and a notebook where all the executions will be done to deploy the model to the cloud (AWS). Then, for the construction of the model's endpoint, import the necessary libraries. To the notebook instance, upload the stored model and model weights.

In [1]:
```
1 #setting up
2 import boto3, re
3 from sagemaker import get_execution_role
4
5 role = get_execution_role()
```

In [2]:
```
1 import keras
2 from keras.models import model_from_json
3 import tensorflow as tf
4
5 tf.compat.v1.disable_eager_execution()
```

```
Using TensorFlow backend.
```

In [4]:
```
1 #loading the model
2 json_file = open('keras_model/model.json', 'r')
3 loaded_model_json = json_file.read()
4 json_file.close()
5 loaded_model = model_from_json(loaded_model_json,custom_objects={"GlorotUniform": tf.keras.initializers.glorot_uniform})
```

In [6]:
```
1 #loading the model weights
2 loaded_model.load_weights('keras_model/weights.h5')
3 print("Loaded model from disk")
```

Endpoint Creation :

```
In [23]:
1 %%time
2 predictor = sagemaker_model.deploy(initial_instance_count=1,
3                                    instance_type='ml.t2.medium')
```

```
update_endpoint is a no-op in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.

-------!CPU times: user 1.04 s, sys: 123 ms, total: 1.16 s
Wall time: 3min 34s
```

Endpoint Name :

```
In [24]:
1 predictor.endpoint
```

```
The endpoint attribute has been renamed in sagemaker>=2.
See: https://sagemaker.readthedocs.io/en/stable/v2.html for details.
```

```
Out[24]: 'sagemaker-tensorflow-serving-2021-10-14-00-07-15-516'
```

```
In [25]:
1 endpoint_name = 'sagemaker-tensorflow-serving-2021-10-14-00-07-15-516'
```

The enpoint is created successfully.

3.3 Creating a client :

In this task, we will write a Python function that accepts a PF file as an input and returns the file's nature. All of the internal activities needed to get to the desired outcome are completed using the cloud API developed in the previous job.
The boto3 library is used to connect to the AWS Sagemaker API, and the requisite keys and token ids of the AWS CLI are specified.

```
In [6]:
1 !wget https://repo.anaconda.com/archive/Anaconda3-2020.02-Windows-x86_64.exe
```

```
--2021-10-15 00:25:34--  https://repo.anaconda.com/archive/Anaconda3-2020.02-Windows-x86_64.exe
Resolving repo.anaconda.com (repo.anaconda.com)... 104.16.131.3, 104.16.130.3, 2606:4700::6810:8303, ...
Connecting to repo.anaconda.com (repo.anaconda.com)|104.16.131.3|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 488908696 (466M) [application/octet-stream]
Saving to: 'Anaconda3-2020.02-Windows-x86_64.exe'

Anaconda3-2020.02-W 100%[===================>] 466.26M   177MB/s    in 2.6s

2021-10-15 00:25:36 (177 MB/s) - 'Anaconda3-2020.02-Windows-x86_64.exe' saved [488908696/488908696]
```

Client Execution :

```
In [10]:  1 !python clientPE.py Anaconda3-2020.02-Windows-x86_64.exe

WARNING: EMBER feature version 2 were computed using lief version 0.9.0-
WARNING:   lief version 0.10.0-845f675 found instead. There may be slight inconsistencies
WARNING:   in the feature calculations.
tcmalloc: large alloc 4400185344 bytes == 0x55c6c2034000 @  0x7fcbc2cad1e7 0x55c6a1252130 0x55c6a122f413 0x55c6a12600a
1220c52 0x55c6a1293c25 0x55c6a128eced 0x55c6a1221bda 0x55c6a128fc0d 0x55c6a1221afa 0x55c6a128fc0d 0x55c6a1221afa 0x55c
0x55c6a128e9ee 0x55c6a128e6f3 0x55c6a13584c2 0x55c6a135883d 0x55c6a13586e6 0x55c6a1330163 0x55c6a132fe0c 0x7fcbc1a97bf
132fcea
tcmalloc: large alloc 3911270400 bytes == 0x55c6c2034000 @  0x7fcbc2cad1e7 0x7fcbbe20946e 0x7fcbbe259c7b 0x7fcbbe259d9
e259fe9 0x7fcbbe25cd7d 0x7fcbbe24d6ba 0x55c6a1222749 0x7fcbbe246ef7 0x55c6a1220437 0x55c6a1220240 0x55c6a1293973 0x55c
0x55c6a1221bda 0x55c6a1290737 0x55c6a1221afa 0x55c6a128fc0d 0x55c6a128eced 0x55c6a1221bda 0x55c6a128f915 0x55c6a128ece
1221bda 0x55c6a128fc0d 0x55c6a1221afa 0x55c6a128fc0d 0x55c6a1221afa 0x55c6a128f915 0x55c6a128e9ee 0x55c6a128e6f3 0x55c
0x55c6a135883d
{'predictions': [[0.5]]}
```

4 Conclusion :

This project's construction is totally dependent on the versions of the libraries and packages used, as well as their installation. I learned a lot about a lot of different services and packages while working on this project. The overall experience of constructing this project has been educational and enlightening.