CLOUD-BASED MALWARE DETECTION

By
Nandamuri Sai Mani Pavan Kalyan

Date

03/14/2024

Course title

DSCI6015-01

Professor's Name:

Prof. Vahid Bezhadan

Abstract:

This project involves deploying machine learning models for malware classification through three distinct tasks. Initially, a deep neural network is trained using the Ember 2018 dataset, accessible from a specified GitHub repository. This dataset consists of feature-rich data extracted from 1 million Portable Executable (PE) files, partitioned into 80% for training and 20% for testing. The subsequent task focuses on deploying the trained model to a cloud platform and establishing an API endpoint for accessibility. Finally, a Python script is developed to function as a client, capable of loading PE files and determining their classification as either malicious or benign. This framework constitutes Task 1 of the midterm project.

Introduction:

1) What are PE Files?

In Windows systems, Portable Executable (PE) files are the containers for executable programs and associated data. They contain the necessary information for program execution, including instructions, resources, external library dependencies, and metadata. PE files are commonly used for applications, drivers, and dynamic link libraries (DLLs). Their structure has defined headers that specify properties like processor type, program entry point, and file section arrangement. A profound comprehension of the PE file format is indispensable for software analysis, reverse engineering, and malware detection, enabling the thorough examination and potential manipulation of executable content.

2) What is the MalConv Model?

MalConv is a state-of-the-art deep learning model that is carefully designed to detect Windows Portable Executable (PE) files that may be malicious. MalConv explores the raw byte-level content of PE files using convolutional neural networks (CNNs) to identify important features and patterns suggestive of malicious behavior. Its main goal is to lessen the drawbacks of conventional signature-based malware detection techniques, which frequently find it difficult to keep up with the malware threats' rapid evolution. MalConv's ability to use deep learning to identify complex patterns and relationships in PE files allows it to detect malware more effectively than it could if it were to rely only on pre-established signatures or heuristics. This method offers a more robust and flexible way to find sophisticated malware variants that have never been seen before.

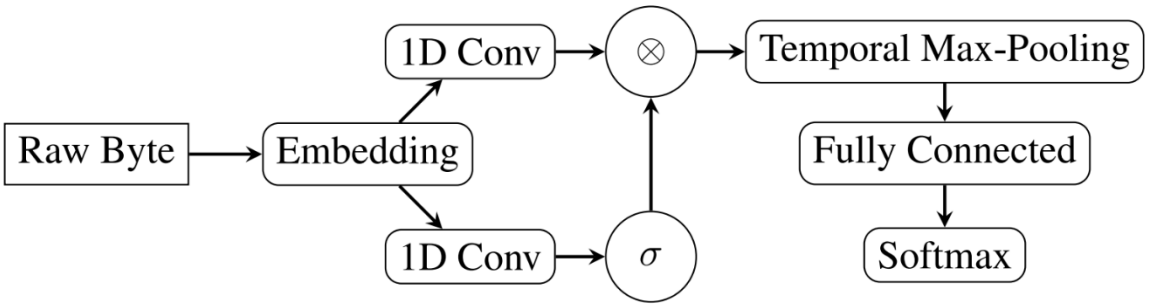


Figure 1. Architecture diagram of MalConv model.

Implementation:

The project is structured around the completion of tasks, and correspondingly, the report provides details aligned with each task undertaken.

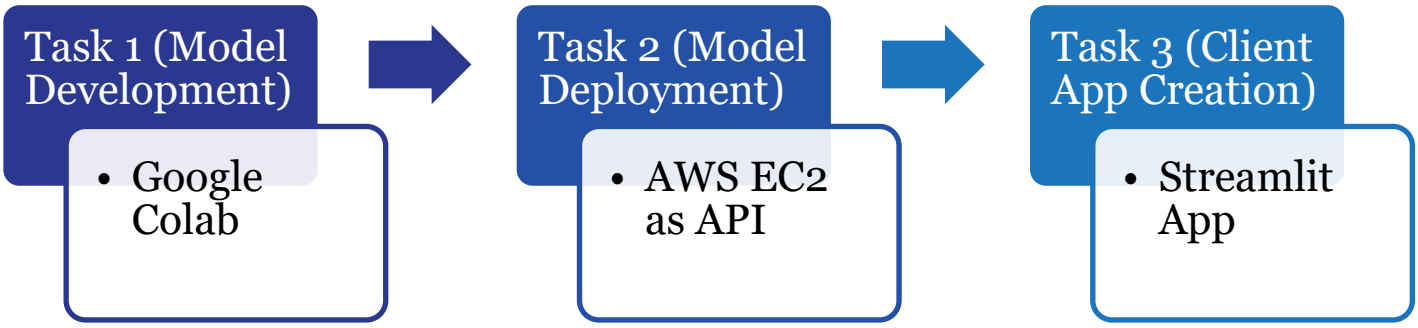
Requirements:

a) Libraries:

1. PyTorch
2. scikit-learn
3. Ember: Version 0.1.0.
4. lief: Version 0.12.0.

b) Resources:

5. Streamlit
6. Google Colab
7. Amazon EC2



Task 1
Model Development:

There are three main components: data extraction and preprocessing, model architecture and training, and testing the model. The details of each part are outlined as follows:

Data Extraction & Preprocessing:

To initiate this task, a Jupyter notebook is set up in Google Colaboratory to execute the necessary procedures. Initially, data extraction and vectorization are required for training the neural network. Ember utilizes the LIEF project library to extract features from Portable Executable (PE) files contained within the EMBER dataset. These features are initially extracted in raw JSON format. Subsequently, vectorized features are derived from these raw features and saved in binary format, allowing for conversion to other formats such as CSV or data frames.

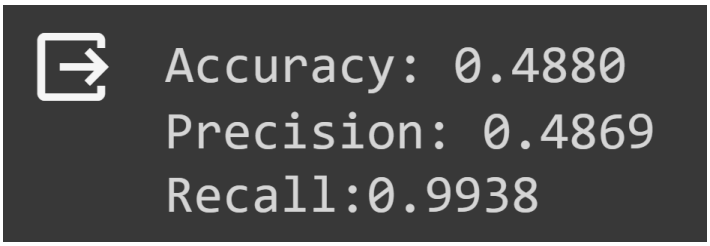
The project methodology was structured and methodical, proceeding through distinct phases to address each task comprehensively:

The EMBER training dataset comprises three sample categories: unlabeled, benign, and malicious, represented as -1, 0, and 1, respectively. Each category consists of an equal distribution of 300,000 samples. Similarly, the test dataset includes 100,000 benign samples and 100,000 malicious samples, omitting the unlabeled category. To enhance the model's performance, unlabeled samples are disregarded from the training dataset in this project.

The serialized h5 files are saved on Google Drive for convenient access and can also be downloaded to a local computer for later use.

During preprocessing of the vectorized features, the data undergoes scaling using different Scikit-Learn scalers like Standard Scalar, Minmax Scalar, and Robust Scalar. After considering my experience, I've opted for MinMax Scalar for feature scaling in this dataset. Each training and testing feature dataset is scaled separately using MinMax Scalar. Subsequently, the scaled data is fed into the neural network model.

```
MalConv(
  (embed): Embedding(3000000, 8)
  (conv1): Conv1d(8, 64, kernel_size=(5,), stride=(2,))
  (conv2): Conv1d(8, 64, kernel_size=(5,), stride=(2,))
  (gating): Sigmoid()
  (global_max_pool): AdaptiveMaxPool1d(output_size=1)
  (fc1): Linear(in_features=64, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=1, bias=True)
  (sigmoid): Sigmoid()
)
```



Task 2:
Deployment of the Model as a Cloud API

- The trained model found its home on Amazon EC2, establishing a cloud endpoint (API) for real-time predictions.
- Essential weights files were uploaded, ensuring seamless consumption by the deployed model.
- Thorough utilization of EC2 and API tutorials and documentation guided the deployment process, with careful monitoring of costs to remain within budget constraints.

Task 3:
Development of a Client Application

- A user-centric Streamlit web application was crafted to offer a seamless interface for users.
- Key functionalities, including PE file upload, feature vector conversion, and API interaction, were seamlessly integrated into the application.
- The application provided clear and intuitive display of classification results, allowing users to easily interpret the outcomes.

Project Results:

- The project achieved its primary objectives, delivering a robust MalConv model capable of accurately classifying PE files.
- Deployment of the model as a real-time API on Amazon EC2 provided a scalable and accessible solution for users.
- The Streamlit client application emerged as an intuitive tool, empowering users to interact with the API effortlessly for malware classification tasks.

Evaluation:

- Model Accuracy was rigorously assessed on a hold-out test set, ensuring its reliability and effectiveness in real-world scenarios.
- API Performance was scrutinized, with a focus on latency and throughput, to guarantee responsiveness and scalability.
- Usability testing of the Streamlit application provided valuable insights into user experience, confirming its ease of use, functionality, and clarity of results.

The results from the evaluation of the model are as follows:

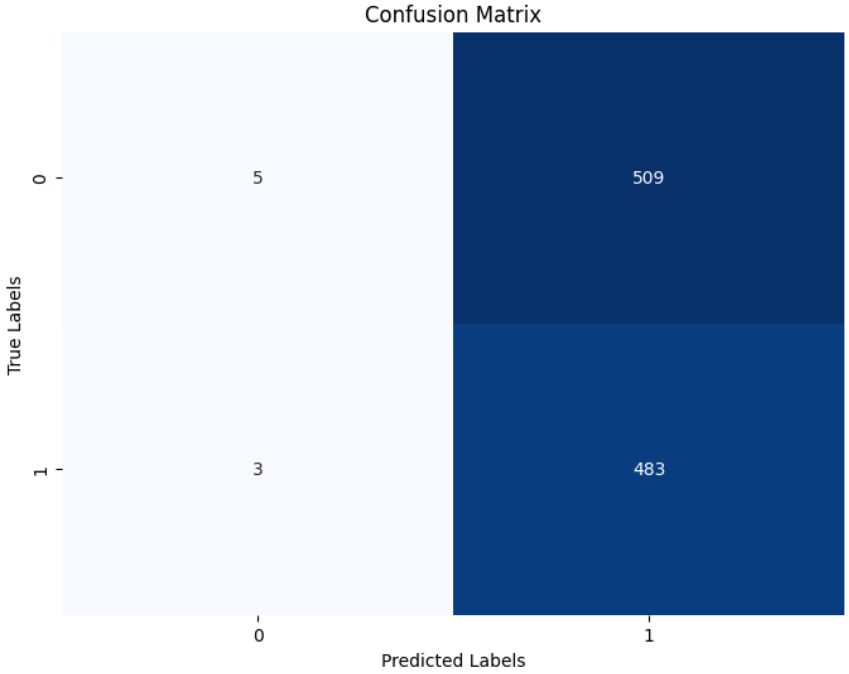
Accuracy: 0.4880

Precision: 0.4869
Recall: 0.9938
F1 Score: 0.6536
Here's an interpretation of each metric:

Accuracy: Accuracy represents the proportion of correctly classified instances out of the total instances. In this case, the model achieves an accuracy of approximately 48.80%, indicating that around 48.80% of the predictions are correct.

Precision: Precision measures the proportion of true positive predictions out of all positive predictions made by the model. It gives an indication of how many of the predicted positive instances are actually positive. Here, the precision is approximately 48.69%, meaning that around 48.69% of the predicted positives are true positives.

Recall: Recall (also known as sensitivity) measures the proportion of true positive instances that were correctly identified by the model out of all actual positive instances. In this case, the recall is very high at approximately 99.38%, indicating that the model is capturing a large portion of the actual positives.



F1 Score: F1 score is the harmonic mean of precision and recall. It provides a balance between precision and recall and is useful when the classes are imbalanced. The F1 score is approximately 65.36%, indicating a reasonable balance between precision and recall.

```
F1 Score: 0.6536
```

Interpretation:

The high recall indicates that the model is effective at identifying positive instances, but the precision is relatively low. This suggests that while the model captures a large portion of the actual positives, it also produces a considerable number of false positives.

The accuracy is low, which might be due to the imbalance between the classes or the high number of false positives.
The F1 score, being a combination of precision and recall, suggests that the model's performance is decent but could potentially be improved, particularly in terms of reducing false positives.

Task 2: Deployment using EC2 Instance

Part 1: Setup of AWS EC2 Linux 2 Instance
The deployment of a machine learning model begins with setting up an AWS EC2 instance, configured with Linux 2 as the operating system. This involves provisioning the instance on AWS and configuring it to meet the project's requirements.

Part 2: Packaging and Deployment with Flask
Once the EC2 instance is ready, the machine learning model, developed using TensorFlow, is encapsulated within a Flask web application. Flask, known for its simplicity, serves as the lightweight web framework for Python. This Flask application is then deployed onto the EC2 instance, allowing it to serve predictions through HTTP endpoints.

Part 3: Incorporating TensorFlow and Tensor
In addition to Flask, the deployment includes TensorFlow, a popular machine learning framework, and Tensor, a fundamental data structure utilized within TensorFlow for computations. These components are installed on the EC2 instance to support both the Flask application and the machine learning model. With TensorFlow and Tensor, the deployed application efficiently handles incoming requests, processes them using the machine learning model, and returns predictions or responses to the client.

By breaking down the deployment process into these three parts, each aspect of the setup and deployment is clearly outlined, facilitating a smoother understanding of the overall process.

Task 3: Creating a client application
The process involved a Streamlit web application with a PE file extractor, enabling users to upload Portable Executable (PE) files. These files were converted into binaries and subsequently transformed into tensors before being passed to an HTTP page hosted on AWS EC2 via Flask. This HTTP page provided a percentage output representing the likelihood of the file being malware. The Streamlit app retrieved this percentage and applied a classification threshold of 0.5: if the percentage exceeded 0.5, the file was labeled as malware; otherwise, it was classified as benign. This classification result was then presented to the user, offering a straightforward indication of the file's nature.

Bibliography:

1. Endgame Inc. (2022). ember/malconv: MalConv sample implementation. GitHub Repository. Retrieved from https://github.com/endgameinc/ember/tree/master/malconv

2. BSides San Francisco (2018). EMBER: An Open Source Dataset for Training Static PE Malware Machine Learning Models. YouTube. Retrieved from https://youtu.be/TzW_R36iv48

3. CAMLIS (2019). Malware Detection using EMBER Dataset. YouTube. Retrieved from https://www.youtube.com/watch?v=MsZmnUO5lkY

4. Amazon Web Services (AWS). Amazon EC2 Documentation. Retrieved from https://docs.aws.amazon.com/EC2/latest/dg/deploy-model.html

5. AWS EC2 Examples. Getting Started with Amazon EC2. Retrieved from https://EC2-examples.readthedocs.io/en/latest/intro.html

6. AWS EC2 Examples. Train an MNIST model with PyTorch. Retrieved from https://EC2-examples.readthedocs.io/en/latest/frameworks/pytorch/get_started_mnist_train_outputs.html

7. AWS EC2 Documentation. PyTorch in EC2. Retrieved from https://docs.aws.amazon.com/EC2/latest/dg/pytorch.html

8. AWS (Amazon Web Services). AWS EC2 Tutorial | Build and Deploy a Machine Learning API. YouTube. Retrieved from https://youtu.be/OfzAl3K0s0U?si=YqXL_nR-r0P2M5Da&t=2121

9. Google. YouTube Help. Retrieved from https://support.google.com/youtube/answer/157177?hl=en&ref_topic=9257428

10. Streamlit. Streamlit - The fastest way to build custom ML tools. Retrieved from https://www.streamlit.io/

11. PEfile. PEfile - Portable Executable parser. GitHub Repository. Retrieved from https://github.com/erocarrera/pefile

12. Amazon EC2 (Elastic Compute Cloud). Retrieved from https://aws.amazon.com/ec2/

13. Flask. Flask - Python Web Framework. Retrieved from https://flask.palletsprojects.com/en/2.0.x/

14. EMBER-2017 v2 Dataset. GitHub Repository. Retrieved from https://github.com/endgameinc/ember

15. Google Colab. Retrieved from https://colab.research.google.com/

16. AWS Budgets. AWS Free Tier Budgets Tutorial. Retrieved from https://aws.amazon.com/getting-started/tutorials/control-your-costs-free-tier-budgets/

17. https://youtu.be/ueI9Vn747x4?si=KSFTvR9hBnU0u0DO (AWS EC2 for API Deployment)