CLOUD-BASED MALWARE DETECTION

By: Venkata Chanakya Samsani

Date: 03/15/2024

Course title: DSCI6015-01

Professor’s Name: Prof. Vahid Bezhadan

**Abstract:**

This project focuses on deploying machine learning models for malware classification through three key tasks. Initially, a deep neural network is trained using the Ember 2018 dataset, obtained from a specified GitHub repository. The dataset comprises feature-rich data extracted from 1 million Portable Executable (PE) files, divided into 80% for training and 20% for testing. Subsequently, the trained model is deployed to a cloud platform, 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**:

What are PE Files?

Portable Executable (PE) files are containers for executable programs and associated data in Windows systems. They contain vital information for program execution, including instructions, resources, external library dependencies, and metadata. Understanding the PE file format is crucial for software analysis, reverse engineering, and malware detection, allowing for thorough examination and potential manipulation of executable content.

What is the MalConv Model?

MalConv is a cutting-edge deep learning model designed to detect potentially malicious Windows Portable Executable (PE) files. Utilizing convolutional neural networks (CNNs), MalConv analyzes the raw byte-level content of PE files to identify significant features and patterns indicative of malicious behavior. Unlike conventional signature-based malware detection methods, MalConv leverages deep learning to identify complex patterns and relationships in PE files, enabling it to detect malware more effectively. This approach provides a robust and flexible solution for detecting sophisticated malware variants that may not have been previously encountered.

A screenshot of a computer

Description automatically generated

**Implementation:**

The project is structured around completing tasks, with the report providing details aligned with each undertaken task.

Requirements:

Libraries:

PyTorch

scikit-learn

Ember: Version 0.1.0

lief: Version 0.12.0

Resources:

Streamlit

Google Colab

Amazon EC2

Task 1 - Model Development:

Data Extraction & Preprocessing:

The project initiates with a Jupyter notebook set up in Google Colaboratory to execute necessary procedures. Data extraction and vectorization are performed for training the neural network using the Ember dataset, employing the LIEF project library to extract features from PE files. Serialized h5 files are saved for easy access, and the vectorized features undergo scaling using MinMax Scalar for improved model performance.

Task 2: Deployment of the Model as a Cloud API

Part 1: Setup of AWS EC2 Linux 2 Instance

The deployment process begins with provisioning an AWS EC2 instance configured with Linux 2 as the operating system. This involves setting up 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 trained 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. The Flask application is deployed onto the EC2 instance, enabling 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: Development of a Client Application

The project entails the creation of a user-centric Streamlit web application to provide a seamless interface for users. Key functionalities, such as PE file upload, feature vector conversion, and API interaction, are integrated into the application. This allows for clear and intuitive display of classification results, facilitating easy interpretation for users.

**Project Results:**

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

**Evaluation:**

**Test Accuracy**: The test accuracy of the model is approximately 50.18%. This metric represents the proportion of correctly classified instances out of the total instances in the test dataset. The relatively low accuracy suggests that around half of the predictions made by the model are correct.

**Test Precision**: The test precision is approximately 55.08%. Precision measures the proportion of true positive predictions out of all positive predictions made by the model. A precision of 55.08% indicates that around 55.08% of the predicted positive instances are actually true positives.

**Test Recall:** The test recall, also known as sensitivity, is approximately 1.30%. Recall measures the proportion of true positive instances that were correctly identified by the model out of all actual positive instances. A recall of 1.30% suggests that the model is capturing a very small portion of the actual positives.

A diagram of a confused matrix

Description automatically generated

**Interpretation:**

The model demonstrates a relatively high precision, indicating that when it predicts a file as malicious, it is correct around 55.08% of the time.

However, the recall rate is notably low, suggesting that the model is only capturing a very small portion (approximately 1.30%) of the actual malicious files present in the dataset.

The low recall rate coupled with the moderate precision implies that while the model is relatively good at identifying true positives, it is missing a significant number of positive instances, leading to a high number of false negatives.

The accuracy, though higher than random guessing, is still relatively low, indicating that the model's performance is limited.

Overall, these results suggest that the model has limitations in effectively capturing the characteristics of malicious files. Further optimization and refinement of the model architecture and training process may be necessary to improve its performance, particularly in terms of recall model's decent performance.

**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)