Midterm Project

A blue logo with text

Description automatically generated

**AI and CyberSecurity DSCI6015**

**Cloud-based PE Malware Detection API**

## Veda Samohitha Chaganti

## ECECS

## University of New Haven

## Dr. Vahid Behzadan

## February 10, 2024

# Static Malware Classifier API

This project delves into a comprehensive exploration of developing a robust machine learning solution for malware detection, encompassing feature engineering, model training, deployment, and evaluation. The focal point is a Random Forest classifier, adeptly trained on a plethora of features extracted from Portable Executable (PE) files.

To seamlessly integrate this model into real-world applications, it's deployed as an API endpoint leveraging the scalability and convenience of AWS SageMaker. Here, it becomes readily accessible for external interactions.

A bespoke Python client is meticulously crafted to interface with the deployed model. This client is adept at extracting pertinent features from executable files, sending them to the SageMaker endpoint, and gracefully handling classification results.

In the conclusive phase, the model's efficacy is meticulously assessed and benchmarked. A curated dataset, comprising 100 samples each of malware and benign files sourced from the EMBER 2018 repository, is meticulously utilized for this purpose. This rigorous evaluation sheds light on the model's performance under real-world conditions, enabling informed decisions about its deployment and further refinement.

# Introduction

# The constantly evolving landscape of malicious software attacks necessitates the development of robust and efficient detection methods. Leveraging machine learning presents a promising avenue for identifying malware by analyzing features extracted from Portable Executable (PE) files.

This project is structured into three pivotal stages:

**Model Training and Deployment:** The Random Forest classifier, honed to recognize malware based on PE file features, is seamlessly deployed as an API endpoint utilizing AWS SageMaker. SageMaker streamlines the entire process of constructing, training, and deploying machine learning models, ensuring efficiency and scalability.

**Client Development:** A custom Python client is meticulously crafted to interface with the deployed model. This client ingeniously takes an executable file as input, extracts pertinent features from the PE file, and dispatches these features to the SageMaker endpoint for classification. Subsequently, the client elegantly receives and showcases the model's prediction regarding the file's malignancy status.

**Performance Benchmarking:** To gauge the model's efficacy, a randomized assortment of 100 malware and 100 benign samples is meticulously selected from the EMBER 2018 dataset. Subsequently, the model's performance is meticulously assessed on this dataset, furnishing valuable insights into its accuracy and adaptability across out-of-distribution data.

By deploying the model as an API endpoint and crafting a user-friendly client, this project epitomizes the practical application of machine learning for malware detection. Furthermore, benchmarking the model's performance on a representative dataset serves as a cornerstone for further refinement and development endeavors.

# Methodology

# In this project, a multifaceted methodology was employed to extract pertinent features from Portable Executable (PE) files:

# Byte Sequence n-grams: This method involves extracting consecutive sequences of n bytes, commonly known as n-grams, from the binary content of each file. By capturing patterns within the raw byte sequences, this technique offers insights into the inherent structure and behavior of the PE files.

# Imported DLLs: Dynamically Linked Libraries (DLLs) play a pivotal role in the functionality of PE files. To harness this information for feature representation, the names of DLLs imported by each file were meticulously extracted and preprocessed. This step unveils crucial insights into the dependencies and functionalities utilized by the executable.

# Section Names: PE files are segmented into distinct sections, each serving a specific purpose within the executable. The names of these sections are extracted and processed to unveil potential structural patterns embedded within the files. Additionally, the number of sections present in each PE file is extracted as a feature, providing further insights into its composition and complexity.

# By integrating these diverse feature extraction techniques, the project aims to capture a comprehensive representation of the PE files, enabling the subsequent model training and classification processes to discern intricate patterns indicative of malware behavior.

# Feature Representation

# N-gram Features: Adopting a K-most-common-features methodology, the code diligently retained the K most frequently occurring n-gram counts across the dataset. By focusing on the most prevalent n-grams, this approach ensured a concise yet informative representation of the raw byte sequences inherent in the PE files.

# Textual Features: For the extracted imported DLL names and section names, the code seamlessly integrated natural language processing techniques:

# Hashing Vectorizer: This technique facilitated the transformation of textual data into fixed-length numerical vectors. By leveraging hashing functions, it ensured efficient representation of varying-length text features, thus enabling streamlined processing and analysis.

# TF-IDF (Term Frequency-Inverse Document Frequency): This pivotal technique was harnessed to evaluate the importance of words within the corpus. By weighing the occurrence of terms against their frequency across the entire dataset, TF-IDF effectively mitigated the influence of common terms while accentuating those with higher discriminatory power. This nuanced approach to feature representation contributed to the model's ability to discern meaningful patterns within the textual data extracted from the PE files.

# Model Training

# In the realm of model training, the project strategically opted for a Random Forest classifier to tackle the challenge of malware detection. This classifier is renowned for its robustness and efficiency in discerning malicious software from benign files. Leveraging an ensemble of decision trees, Random Forest models excel in capturing complex, non-linear relationships within the data. Moreover, they adeptly mitigate overfitting concerns, ensuring the model's generalizability to unseen data.

# Training Data

# The backbone of the model's training process lies in the meticulously curated dataset comprising features extracted from Portable Executable (PE) files. This dataset serves as a treasure trove of information, encapsulating the unique characteristics and nuances inherent in malware and benign software alike. By harnessing diverse feature extraction techniques, the dataset provides a comprehensive representation of PE files, enabling the model to discern subtle patterns indicative of malicious behavior. Through the integration of this rich training data, the model gains the necessary insights to make accurate and reliable predictions in real-world scenarios

# Model Deployment:

# Model Preparation:

# In the crucial phase of model preparation, meticulous steps were taken to ensure the seamless deployment and accessibility of the trained Random Forest model. These steps included:

# Packaging into TAR.GZ Archive: The trained Random Forest model, along with its associated artifacts such as the entrypoint script and necessary dependencies, was meticulously packaged into a TAR.GZ archive. This archive encapsulates all the essential components required for the model's deployment and execution

# Upload to S3 Bucket: To facilitate access and utilization by AWS SageMaker, the model archive was securely uploaded to an S3 bucket with the designated path s3://simplified-midterm-03/. This cloud-based storage ensures the scalability, reliability, and accessibility of the model assets, allowing for seamless integration into the SageMaker environment

# SageMaker Deployment:

# The deployment of the trained model onto AWS SageMaker was seamlessly orchestrated using the AWS SageMaker Python SDK, ensuring efficiency and scalability. This deployment process involved several key steps:

**Role Assumption:** The project leveraged the **get\_execution\_role** function to acquire the necessary permissions for deploying the model. This role assumption mechanism facilitates secure access to AWS resources, enabling smooth execution of deployment tasks.

**Model Configuration**:

A SKLearnModel object was meticulously configured to encapsulate essential details for model deployment. This included specifying:

* The location of the model data within the S3 bucket.
* The execution role with appropriate permissions.
* The entry point script responsible for model execution.
* The source directory containing additional model code and dependencies.
* The desired Python version and the specific version of the Scikit-learn framework.
* The requisite dependencies required for model execution.

**Endpoint Creation:**

With the model configuration in place, the deploy method was invoked to initiate the deployment process, thereby establishing a RESTful API endpoint. Key specifications included:

* Instance type: "ml.t2.medium", chosen based on resource requirements and scalability considerations.
* Initial instance count: 1, ensuring a single instance for initial deployment.
* Serialization and Deserialization: The request and response formatting were meticulously handled using JSONSerializer and JSONDeserializer, respectively, ensuring seamless data interchange between client and server.

# Client Interaction:

# To facilitate interaction with the deployed model, a Python script was developed to act as a client. This client script underwent the following steps:

**Test Data Preparation:**

Sample test data, represented as a sparse matrix, was prepared for prediction. This data served as input to the model for generating predictions.

**JSON Payload Structuring:**

The test data was structured into a JSON payload suitable for transmission to the model endpoint. This payload included:

**Data:** Converted to a list of values for representation.

Row and column indices for the sparse matrix.

Matrix shape to maintain data integrity during transmission.

**Endpoint Configuration:**

The predictor object's properties, including accept and content\_type, were configured to facilitate communication via JSON format. This ensured compatibility and seamless data interchange between the client and the endpoint.

**Prediction Process:**

Utilizing the predict method, the JSON payload was sent to the endpoint, initiating the prediction process. Subsequently, the model's prediction was retrieved and processed by the client script.

# Client Deployment:

# For streamlined interaction with the deployed model, a Python client application was meticulously crafted. This client application is designed to execute the following tasks seamlessly:

**Input Acceptance:**

The Python client gracefully accepts a new Portable Executable (PE) file as input. This file serves as the basis for subsequent feature extraction and prediction.

**Feature Extraction:**

Employing identical techniques as those utilized during model training, the client meticulously extracts relevant features from the provided PE file. These features are essential for the model's prediction of the file's maliciousness.

**Communication with Deployed Model:**

The extracted features are transmitted to the deployed model endpoint on AWS SageMaker. Leveraging the established connection, the client seamlessly interacts with the model, initiating the prediction process.

**Prediction Display:**

Upon receiving the model's prediction regarding the file's maliciousness, the client displays this information to the user. This clear and concise presentation enables users to make informed decisions based on the model's assessment.

# Performance Benchmarking:

# In the critical evaluation phase, the model's performance was meticulously scrutinized using a carefully curated dataset comprising 100 malware and 100 benign samples randomly selected from the EMBER 2018 dataset. This dataset served as the litmus test for assessing the model's efficacy in distinguishing between malicious and benign software

Key components of the performance benchmarking process included:

**Dataset Selection:**

A representative subset of 100 malware and 100 benign samples was meticulously chosen from the EMBER 2018 dataset. This dataset subset provided a balanced representation of both malicious and benign instances, facilitating a comprehensive evaluation of the model's performance.

**Evaluation Metrics:**

The model's performance was rigorously assessed using a suite of essential metrics, including accuracy, precision, recall, and F1-score. These metrics offer valuable insights into the model's ability to correctly classify malware instances while minimizing false positives and false negatives.

**Benchmarking Process:**

The model's performance was benchmarked against established in-distribution benchmarks for malware detection. This comparison enabled a comprehensive assessment of the model's effectiveness relative to existing standards in the field.

# Performance Evaluation and Benchmarking:

# This section meticulously evaluates the model's performance on both in-distribution and out-of-distribution (OOD) data, shedding light on its effectiveness in discerning malicious software across varied scenarios. The evaluation encompasses a comprehensive analysis of key metrics, including accuracy, precision, recall, F1 score, and AUC-ROC score, offering valuable insights into the model's predictive capabilities.

# In-distribution Data:

# The model exhibited robust performance on in-distribution data, underscoring its proficiency in learning underlying patterns effectively. Here's a breakdown of the metrics:

# Accuracy: 0.9860 (Very high, indicating a significant proportion of correct predictions)

# Precision: 0.9884 (Very high, signifying minimal false positive errors)

# Recall: 0.9884 (Very high, indicating effective identification of relevant cases)

# F1 Score: 0.9884 (Very high, reflecting a harmonious balance between precision and recall)

# AUC-ROC Score: 0.9854 (Very high, suggesting exceptional discriminatory ability)

# The remarkable performance across all metrics, coupled with the visual confirmation of the ROC curve aligning closer to the top-left corner, underscores the model's suitability for tasks involving in-distribution data. These results affirm the model's efficacy in accurately distinguishing between benign and malicious software instances within familiar data distributions.

# Conclusion:

# The comprehensive evaluation conducted in this study unequivocally showcases the model's efficacy in handling in-distribution data, with remarkable performance across various metrics. However, a notable degradation in performance is observed when the model is subjected to out-of-distribution (OOD) data. This discrepancy underscores the critical importance of further exploration into out-of-distribution detection and handling techniques to bolster the model's robustness in real-world scenarios, where encountering unseen data is a distinct possibility.

# The findings of this evaluation serve as a valuable catalyst for future research endeavors aimed at enhancing the model's ability to generalize beyond the confines of familiar data distributions. By delving deeper into OOD detection and mitigation strategies, we can fortify the model's resilience and ensure its reliability in diverse and dynamic environments. Ultimately, this pursuit of continuous improvement will empower the model to effectively combat emerging threats and fulfill its role as a formidable tool in the fight against malware and cyber threats.