

CLOUD BASED MALWARE DETECTION API

Submitted to: Prof. Vahid Behzadan



Venkata Chanakya Samsani

00831642

DSCI-6015-01

MS in Data Science



Abstract:

This project presents the deployment of a machine learning model for malware classification, focusing on three main tasks. Initially, the project involves deploying a pre-trained model as an API endpoint on AWS SageMaker. Subsequently, a Python client is developed to interact with the deployed model, facilitating feature extraction and classification results retrieval. Finally, both the client and the endpoint functionalities are tested using representative malware and benign files.

Introduction:

1. Understanding PE Files:

Portable Executable (PE) files are essential components in Windows systems, serving as containers for executable programs and associated data. They encapsulate crucial information necessary for program execution, including instructions, resources, and dependencies. A comprehensive understanding of PE file format is vital for software analysis, reverse engineering, and malware detection, enabling thorough examination and potential manipulation of executable content.

2. Advantages of Random Forest Classifier:

The Random Forest classifier is adept at binary classification tasks due to its robustness, speed, and ability to handle messy data effectively. By constructing multiple decision trees, it prevents overfitting and demonstrates reliability even with incomplete data. Its feature importance analysis aids in understanding data dynamics, simplifying interpretation and decision-making. Moreover, its versatility and minimal data preprocessing requirements make it suitable for various real-world applications.

Implementation:

The project's implementation is structured around specific tasks, each contributing to the overall goal of deploying a robust malware classification solution.

Requirements:

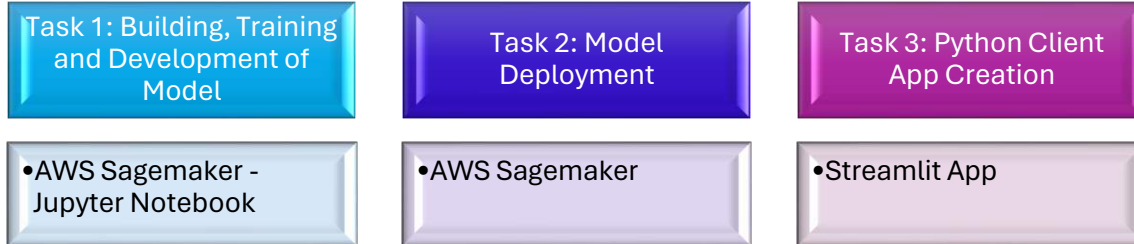
a) Libraries:

- nltk
- scikit-learn 1.2.1
- pefile
- joblib
- numpy

b) Resources:

- AWS SageMaker
- Streamlit

- Google Colab



Task 1: Model Training & Development:

1. Data Preparation:

Data preparation is a critical step that involves collecting and organizing a labeled dataset of binary feature vectors. Typically, this dataset comprises samples of both benign and malware executables. These samples are then split into training and testing sets to facilitate model evaluation. Proper data preparation ensures that the model learns from a diverse and representative set of examples, improving its generalization performance.

2. Feature Extraction:

Feature extraction involves deriving meaningful features from the raw data (in this case, binary files) to facilitate the learning process. For malware classification, features may include n-grams, metadata such as DLL imports, and Portable Executable (PE) section names. Extracting relevant features is crucial for capturing distinguishing characteristics that can differentiate between benign and malicious executables.

3. Pipeline Creation:

A pipeline is constructed to automate the feature extraction and transformation process. This pipeline typically includes components such as HashingVectorizer and TfidfTransformer, which are used to extract n-gram features and construct feature vectors from DLL imports and section names. The pipeline ensures consistency in feature extraction across training and prediction phases, simplifying the deployment process.

4. Model Training:

Once the features are extracted and transformed, a machine learning model is trained using the training dataset. In this project, a Random Forest classifier is trained due to its effectiveness in binary classification tasks and robustness to noisy data. During training, the model learns to recognize patterns in the feature vectors that distinguish between benign and malware samples.

5. Model Evaluation:

After training, the model's performance is evaluated using the testing dataset. Evaluation metrics such as accuracy, precision, recall, and F1-score are computed to assess the model's effectiveness in classifying executables. By evaluating the model on an independent dataset, its ability to generalize to unseen data can be gauged, providing insights into its real-world performance.

6. Model Saving:

Once the model is trained and evaluated, it is serialized and saved to disk using serialization libraries such as joblib. Additionally, any preprocessing artifacts, such as featurizers used during feature extraction, are also saved. These saved artifacts ensure that the trained model can be reused for future predictions without the need for retraining, streamlining the deployment process.

7. Loading Saved Model and Featurizers for Prediction:

During deployment, the saved model and featurizers are loaded from disk to perform predictions on new data. This step is crucial for ensuring consistency between the training and prediction phases, as the same preprocessing steps are applied to new data before passing it through the trained model for classification.

By following these activities, a robust machine learning model for malware classification can be developed and deployed, paving the way for effective detection of malicious executables in real-world scenarios.

Task 2: Deployment of the Model as a Cloud API:

The deployment process constitutes a pivotal phase in operationalizing machine learning models, ensuring their accessibility and scalability for real-world applications. Let's delve deeper into each step of this process:

1. Saving the Trained Model and Writing Inference Script:

Upon completing model training and evaluation, the trained model along with any necessary preprocessing artifacts such as featurizers or transformers need to be serialized and saved. This ensures that the model state is preserved and can be loaded for inference tasks. Additionally, an inference script is written to define how incoming data will be processed and predictions will be made using the trained model. This script typically includes functions to load the model and any required preprocessing steps.

2. Zipping the Model and Necessary Files:

To streamline the deployment process, the saved model, inference script, and any other required files are typically compressed into a single archive file, often in ZIP format. This simplifies the transfer of files and ensures that all necessary components are bundled together for deployment on SageMaker.

3. Deploying the Model on Amazon SageMaker:

The next step involves uploading the zipped model to an Amazon S3 bucket, which serves as a storage repository for model artifacts. SageMaker utilizes S3 to access the model files during the deployment process. Once the model artifacts are stored in S3, a SageMaker-compatible Docker image is selected for running inference tasks. This image contains the necessary runtime environment and dependencies for executing the model.

4. Creating the SageMaker Model:

Using the uploaded model artifacts and the selected Docker image, a SageMaker model is created. This model configuration specifies how the deployed model will handle incoming requests and process data for inference. It includes references to the inference script and any additional resources required for execution.

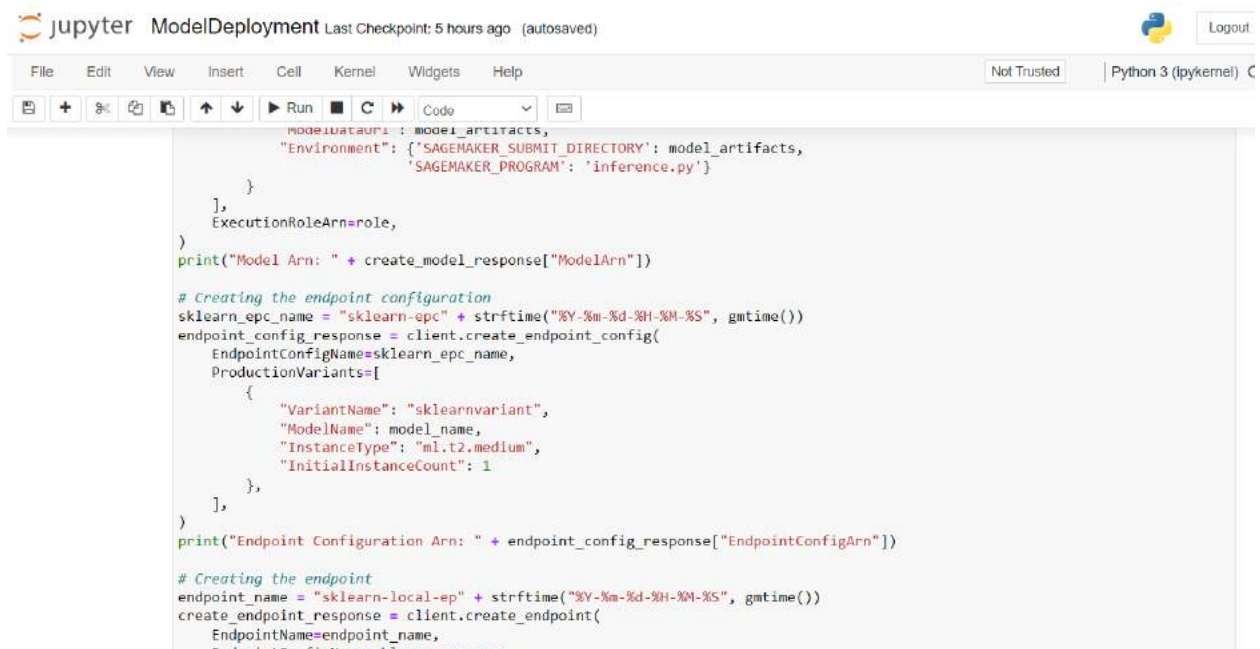
5. Configuring and Creating the Endpoint:

Before the model can be accessed for inference, an endpoint configuration must be created. This configuration defines the compute resources, such as instance type and count, allocated to serve the model. Once the endpoint configuration is defined, an endpoint is created using this configuration. The endpoint serves as the access point for making predictions using the deployed model.

6. Testing the Endpoint:

Once the endpoint is created, it undergoes testing to verify its functionality and performance. This typically involves sending sample input data to the endpoint and validating the output predictions against expected results. Testing ensures that the deployed model behaves as expected and meets the required accuracy and performance criteria.

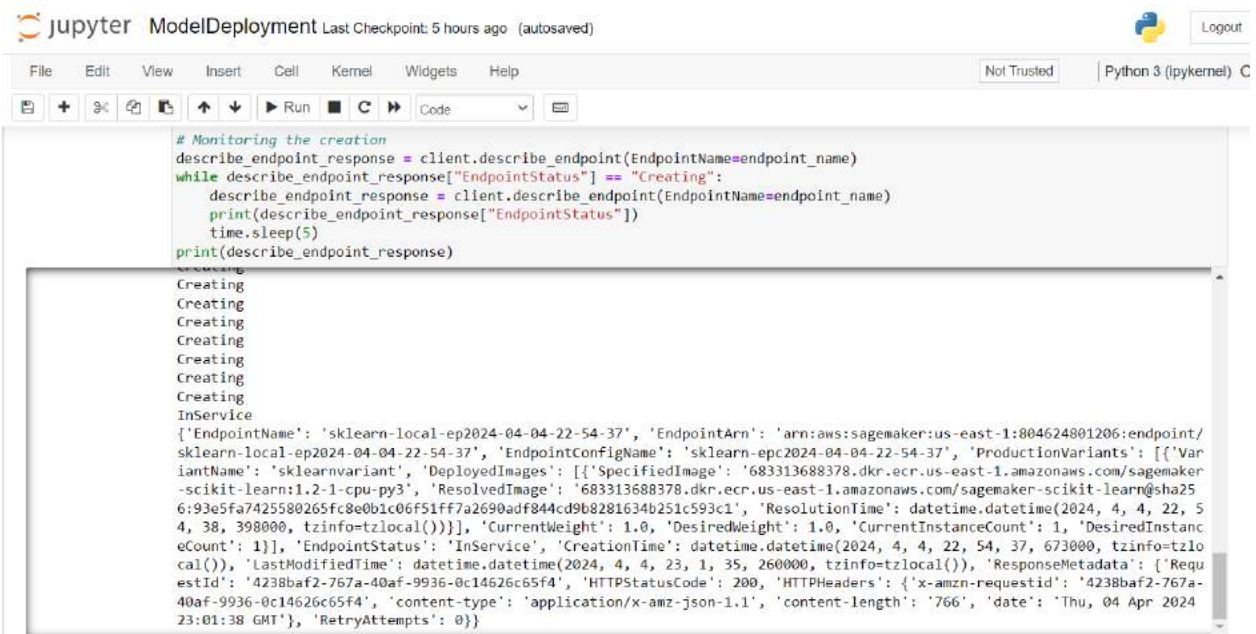
By following these steps, machine learning models can be effectively deployed on Amazon SageMaker, providing a reliable and scalable platform for inference tasks. This deployment process lays the foundation for integrating machine learning capabilities into production systems, enabling organizations to leverage the power of AI for various applications.



```
ModelDeployment Last Checkpoint: 5 hours ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel) C
model_data_uri = model_artifacts,
"Environment": {'SAGEMAKER_SUBMIT_DIRECTORY': model_artifacts,
               'SAGEMAKER_PROGRAM': 'inference.py'}
    },
    ],
    ExecutionRoleArn=role,
)
print("Model Arn: " + create_model_response["ModelArn"])

# Creating the endpoint configuration
sklearn_epc_name = "sklearn-epc" + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
endpoint_config_response = client.create_endpoint_config(
    EndpointConfigName=sklearn_epc_name,
    ProductionVariants=[
        {
            "VariantName": "sklearnvariant",
            "ModelName": model_name,
            "InstanceType": "ml.t2.medium",
            "InitialInstanceCount": 1
        }
    ],
)
print("Endpoint Configuration Arn: " + endpoint_config_response["EndpointConfigArn"])

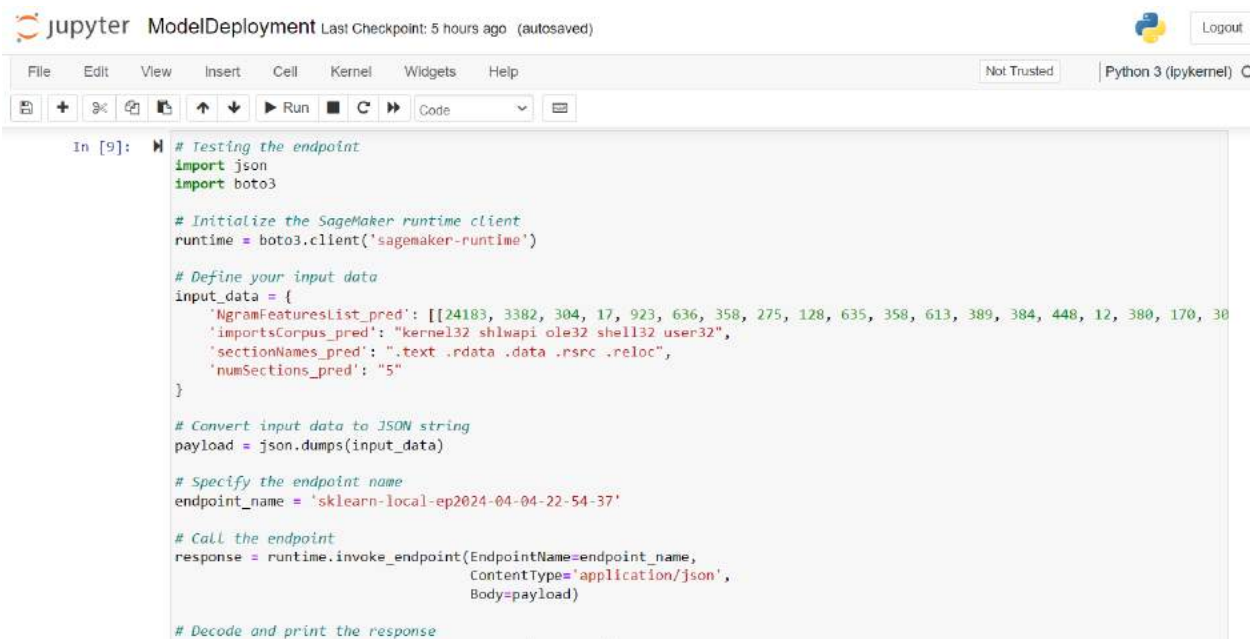
# Creating the endpoint
endpoint_name = "sklearn-local-ep" + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
create_endpoint_response = client.create_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=sklearn_epc_name
```



The screenshot shows a Jupyter Notebook interface with the title 'ModelDeployment' and a status 'Last Checkpoint: 5 hours ago (autosaved)'. The notebook has a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running cells, and code execution. The code in the cell is a loop that monitors the status of an endpoint creation. The output shows the endpoint is created and then enters the 'InService' state, followed by a detailed JSON response from the AWS SageMaker client.

```
# Monitoring the creation
describe_endpoint_response = client.describe_endpoint(EndpointName=endpoint_name)
while describe_endpoint_response["EndpointStatus"] == "Creating":
    describe_endpoint_response = client.describe_endpoint(EndpointName=endpoint_name)
    print(describe_endpoint_response["EndpointStatus"])
    time.sleep(5)
print(describe_endpoint_response)

Creating
Creating
Creating
Creating
Creating
Creating
Creating
InService
{'EndpointName': 'sklearn-local-ep2024-04-04-22-54-37', 'EndpointArn': 'arn:aws:sagemaker:us-east-1:804624801206:endpoint/sklearn-local-ep2024-04-04-22-54-37', 'EndpointConfigName': 'sklearn-epc2024-04-04-22-54-37', 'ProductionVariants': [{'VariantName': 'sklearnvariant', 'DeployedImages': [{'SpecifiedImage': '683313688378.dkr.ecr.us-east-1.amazonaws.com/sagemaker-scikit-learn:1.2-1-cpu-py3', 'ResolvedImage': '683313688378.dkr.ecr.us-east-1.amazonaws.com/sagemaker-scikit-learn@sha256:93e5fa7425580265fc8e0b1c06f51ff7a2690adf844cd9b8281634b251c593c1', 'ResolutionTime': datetime.datetime(2024, 4, 4, 22, 54, 38, 398000, tzinfo=tzlocal())}], 'CurrentWeight': 1.0, 'DesiredWeight': 1.0, 'CurrentInstanceCount': 1, 'DesiredInstanceCount': 1}], 'EndpointStatus': 'InService', 'CreationTime': datetime.datetime(2024, 4, 4, 22, 54, 37, 673000, tzinfo=tzlocal()), 'LastModifiedTime': datetime.datetime(2024, 4, 4, 23, 1, 35, 260000, tzinfo=tzlocal()), 'ResponseMetadata': {'RequestId': '4238baf2-767a-40af-9936-0c14626c65f4', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': '4238baf2-767a-40af-9936-0c14626c65f4', 'content-type': 'application/x-amz-json-1.1', 'content-length': '766', 'date': 'Thu, 04 Apr 2024 23:01:38 GMT'}, 'RetryAttempts': 0}]}
```



The screenshot shows a Jupyter Notebook interface with the title 'ModelDeployment' and a status 'Last Checkpoint: 5 hours ago (autosaved)'. The notebook has a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running cells, and code execution. The code in the cell is a test script that initializes a SageMaker runtime client, defines input data, converts it to JSON, and calls the endpoint. The output shows the response from the endpoint.

```
In [9]: # Testing the endpoint
import json
import boto3

# Initialize the SageMaker runtime client
runtime = boto3.client('sagemaker-runtime')

# Define your input data
input_data = {
    'NgramFeaturesList_pred': [[24183, 3382, 304, 17, 923, 636, 358, 275, 128, 635, 358, 613, 389, 384, 448, 12, 380, 170, 30],
    'importsCorpus_pred': "kernel32 shlwapi ole32 shell32 user32",
    'sectionNames_pred': ".text .rdata .data .rsrc .reloc",
    'numSections_pred': "5"
}

# Convert input data to JSON string
payload = json.dumps(input_data)

# Specify the endpoint name
endpoint_name = 'sklearn-local-ep2024-04-04-22-54-37'

# Call the endpoint
response = runtime.invoke_endpoint(EndpointName=endpoint_name,
                                   ContentType='application/json',
                                   Body=payload)

# Decode and print the response
```

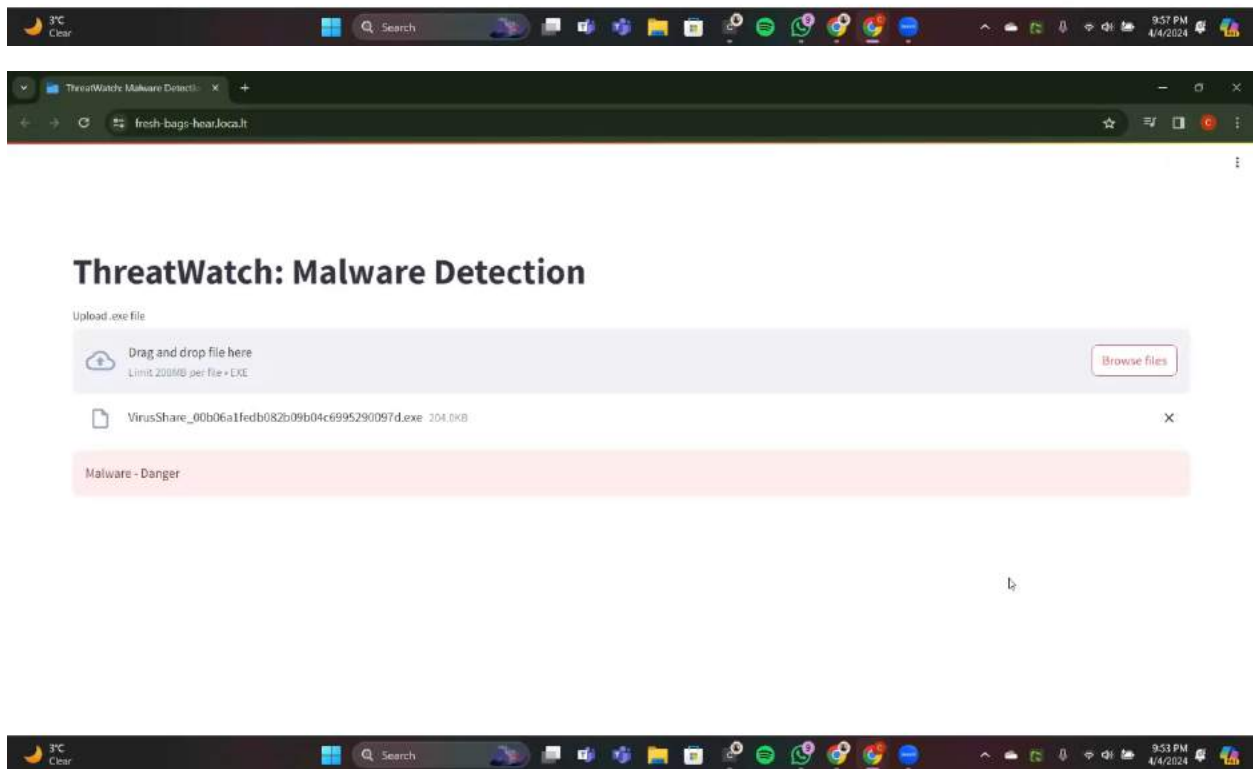
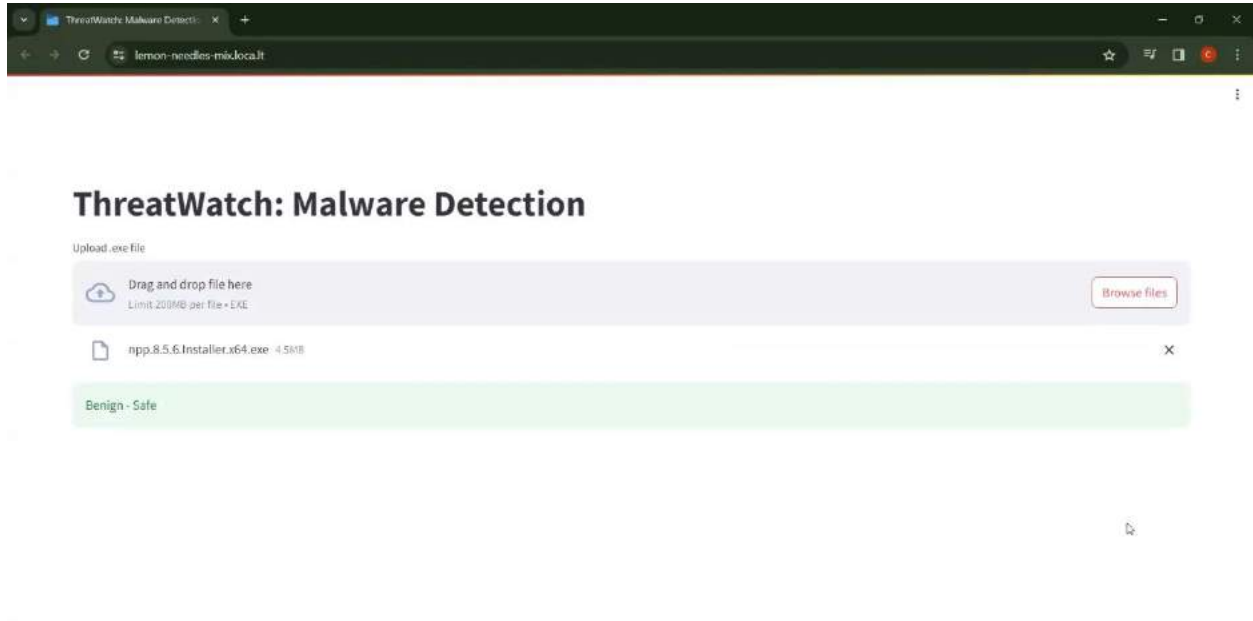
Task 3: Development of a Client Application:

A user-centric Streamlit web application is developed to facilitate seamless interaction with the deployed API. Key functionalities such as PE file upload, feature vector conversion, and API interaction are integrated, providing users with an intuitive platform for malware classification.

Project Results:

- The project successfully delivered a Random Forest Classifier model capable of accurately classifying PE files.

- Deployment of the model as a real-time API on AWS SageMaker provided scalability and accessibility.
- The Streamlit client application emerged as an intuitive tool, empowering users to interact effortlessly with the API for malware classification tasks.



Bibliography:

- AWS SageMaker Documentation
- Streamlit Documentation
- Google Colab Documentation

These resources were instrumental in guiding various aspects of the project, from model deployment to client application development.

Overall, this project demonstrates the successful integration of machine learning models into real-world applications, addressing the critical need for efficient malware classification solutions.