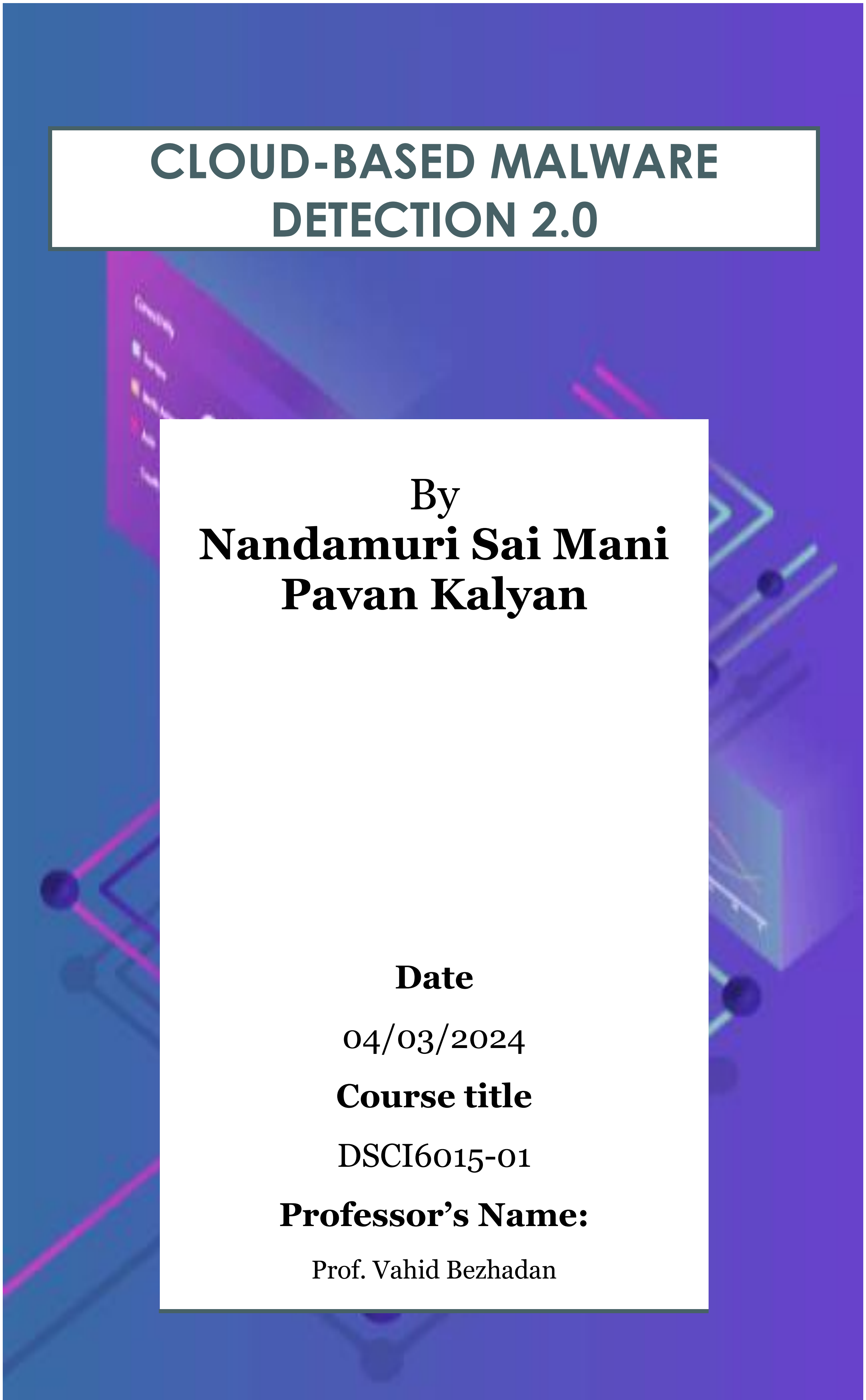


CLOUD-BASED MALWARE DETECTION 2.0

By
Nandamuri Sai Mani Pavan Kalyan

Date

04/03/2024

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.

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) Advantages of Random Forest Classifier

The Random Forest tool is great at sorting things into two groups because it's strong, right-on point, and fast. It puts many choice paths together which stops it from being too narrow and works well even with messy info, making sure the choices it makes can be trusted. This method's way of showing which bits of info matter most helps make picking the right bits simpler and helps us really get what's going on in the data. Plus, it's good at dealing with blanks and doesn't need you to tweak the data much before you start, which makes the first steps easier. This tool works well in many areas, and the way it shows which bits of info are key just makes it even more useful in real life.

Implementation:

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

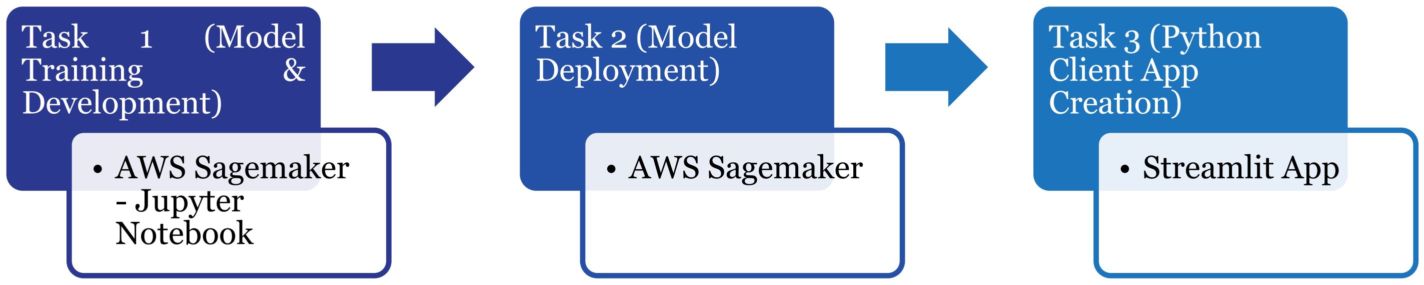
Requirements:

a) Libraries:

1. nltk
2. sci-kit learn 1.2.1
3. pefile
4. joblib
5. numpy

b) Resources:

1. AWS Sagemaker
2. Streamlit
3. Google Colab


Task 1 (Model Training & Development)
• AWS Sagemaker - Jupyter Notebook
Task 2 (Model Deployment)
• AWS Sagemaker
Task 3 (Python Client App Creation)
• Streamlit App

Task 1
Model Training & Development:

This Task involves this series of activities:

1. **Data Preparation:** It collects a labeled dataset of binary feature vectors from two directories, one containing benign samples and the other containing malware samples. It then splits this dataset into training and testing sets.

2. **Feature Extraction:** It extracts features from the binary files, including n-grams and metadata such as DLL imports and PE (Portable Executable) section names. These features are processed to create feature vectors.

3. **Pipeline Creation:** It constructs a pipeline using HashingVectorizer and TfidfTransformer to extract n-gram features and construct feature vectors from DLL imports and section names.

4. **Model Training:** It trains a Random Forest classifier using the feature vectors generated from the training dataset.

5. **Model Evaluation**: It evaluates the trained model's performance on the testing dataset.

6.**Model Saving:** It saves the trained model (model.joblib) and the featurizers (imports_featurizer.pkl and section_names_featurizer.pkl) using joblib.dump.

7. **Model Loading and Prediction:** It loads the saved model and featurizers and performs prediction on a new binary file (MSTeamsSetup_c_l_.exe).

Purpose of these dumps:

- **model.joblib:** This file stores the trained Random Forest classifier model, allowing it to be reused for future predictions without needing to retrain the model.

- **imports_featurizer.pkl** and **section_names_featurizer.pkl:** These files store the featurizers used to transform DLL imports and section names into feature vectors. They are essential for preprocessing new data for prediction, ensuring consistency between the training and prediction processes.

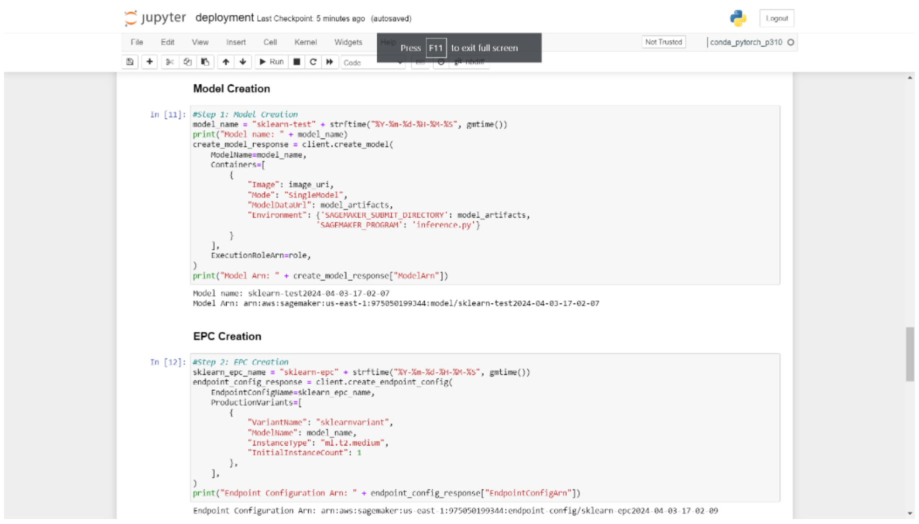
Task 2:
Deployment of the Model as a Cloud API

This code snippet is an example of deploying a machine learning model trained with scikit-learn on Amazon SageMaker using a SageMaker-compatible Docker image for inference. Below is a summary of what each part of the code accomplishes:

1. Saving Models and Writing Inference Script: This section saves trained models (`model.joblib`, `imports_featurizer.pkl`, and `section_names_featurizer.pkl`) and writes an inference script (`inference.py`) that handles input data on the SageMaker platform, processes it, and provides predictions as output.

2. Zipping the Model and Necessary Files: Zips the saved model and inference script along with necessary files.

3. Deployment:
   - Connecting to Model S3 Bucket: Connects to the S3 bucket to store the model artifacts.
   - Retrieving a SKLearn Image URI: Retrieves the SageMaker-compatible Docker image URI for scikit-learn.
   - Importing the Zipped Model from S3 Bucket: Uploads the zipped model to the specified S3 bucket.
   - Model Creation: Creates a SageMaker model using the specified image URI and uploaded model artifacts.


jupyter deployment Last Checkpoint: 5 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Press F11 to exit full screen
Not Trusted
conda_pytorch_p310
Run
Code

Model Creation

```
In [11]: #Step 1: Model Creation
model_name = "sklearn-test" + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print("Model name: " + model_name)
create_model_response = client.create_model(
    ModelName=model_name,
    Containers=[
        {
            "Image": image_uri,
            "Mode": "SingleModel",
            "ModelDataUrl": model_artifacts,
            "Environment": {'SAGEMAKER_SUBMIT_DIRECTORY': model_artifacts,
                            'SAGEMAKER_PROGRAM': 'inference.py'}
        }
    ],
    ExecutionRoleArn=role,
)
print("Model Arn: " + create_model_response["ModelArn"])
```

```
Model name: sklearn-test2024-04-03-17-02-07
Model Arn: arn:aws:sagemaker:us-east-1:975050199344:model/sklearn-test2024-04-03-17-02-07
```

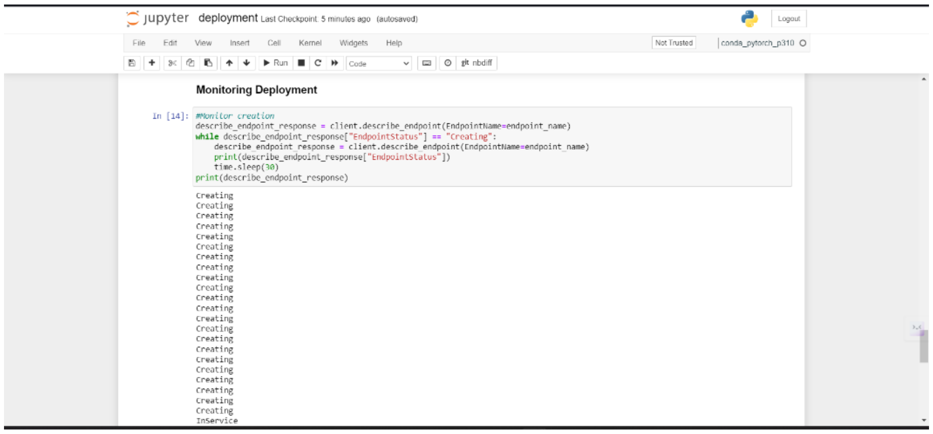
EPC Creation

```
In [12]: #Step 2: EPC Creation
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"])
```

```
Endpoint Configuration Arn: arn:aws:sagemaker:us-east-1:975050199344:endpoint-config/sklearn-epc2024-04-03-17-02-09
```

   - EPC (Endpoint Configuration) Creation: Creates an endpoint configuration for the model.

- Endpoint Creation on SageMaker: Creates an endpoint on SageMaker using the created endpoint configuration.



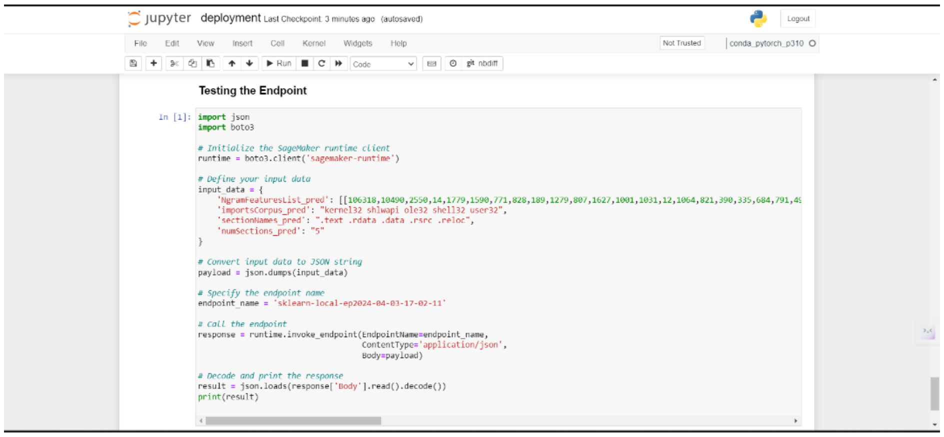
Monitoring Deployment

```
In [14]: #Monitor 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(30)
print(describe_endpoint_response)
```

```
Creating
Creating
Creating
Creating
Creating
Creating
Creating
Creating
Creating
Creating
Creating
Creating
Creating
Creating
Creating
Creating
Creating
Creating
Creating
Creating
Creating
Creating
InService
```

- Monitoring Deployment: Monitors the deployment status until the endpoint is created.

4. Testing the Endpoint: Tests the deployed endpoint by sending sample input data (`input_data`) to the endpoint and printing the prediction result.



Testing the Endpoint

```
In [1]: import json
import boto3

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

# Define your input data
input_data = {
    'NgramFeaturesList_pred': [[106318,10490,2550,14,1779,1590,771,828,189,1279,807,1627,1001,1031,12,1064,821,390,335,684,791,4
    '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-ep2024-04-03-17-02-11'

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

# Decode and print the response
result = json.loads(response['Body'].read().decode())
print(result)
```

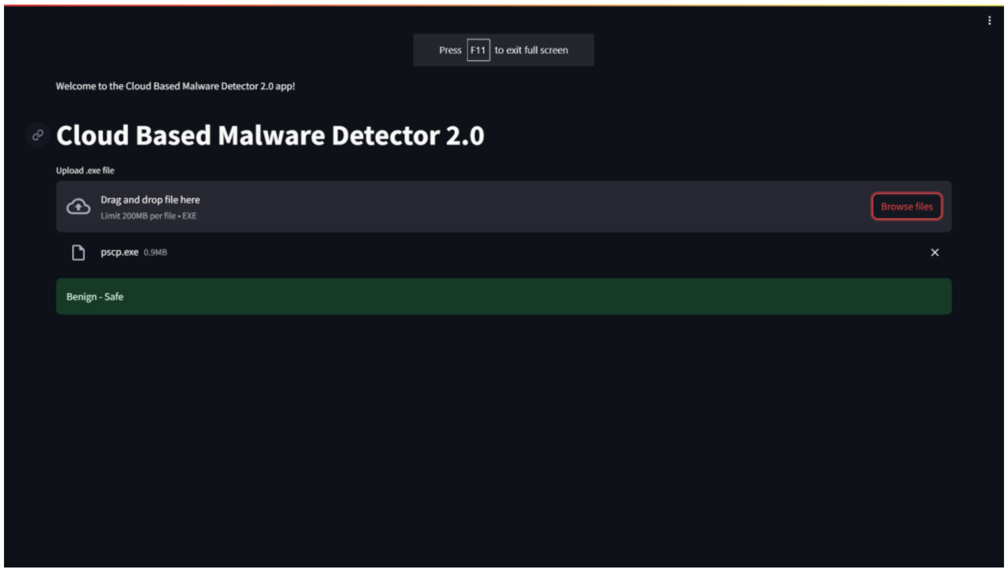
Overall, this code automates the process of deploying a scikit-learn model on Amazon SageMaker, including model preparation, deployment configuration, and endpoint testing.

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 Random Forest Classifer model capable of accurately classifying PE files.

- Deployment of the model as a real-time API on AWS Sagemaker 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.



Press F11 to exit full screen

Welcome to the Cloud Based Malware Detector 2.0 app!

Cloud Based Malware Detector 2.0

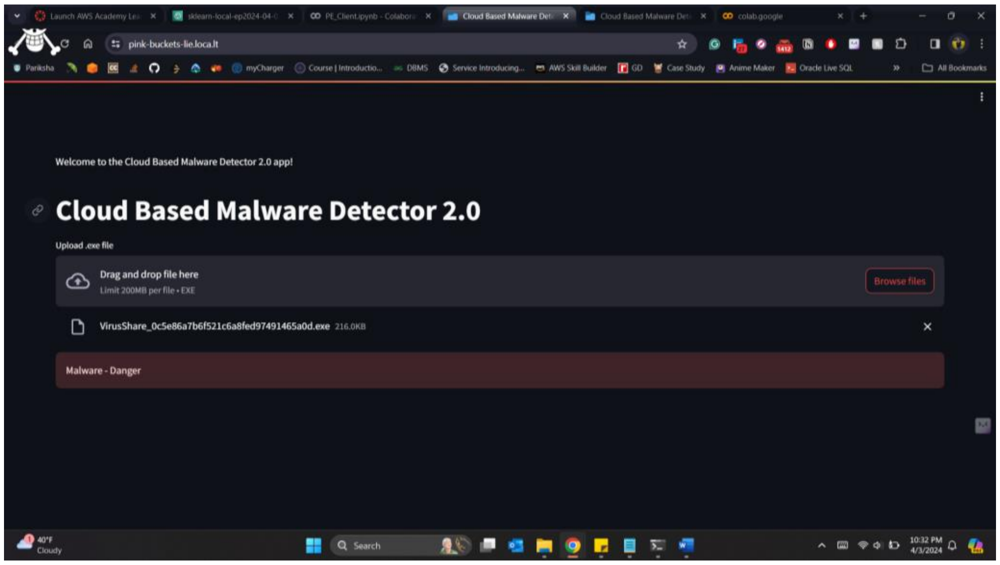
Upload .exe file

Drag and drop file here
Limit 200MB per file • EXE

Browse files

pscp.exe 0.9MB

Benign - Safe



pink-buckets-lie.loca.lt

Welcome to the Cloud Based Malware Detector 2.0 app!

Cloud Based Malware Detector 2.0

Upload .exe file

Drag and drop file here
Limit 200MB per file • EXE

Browse files

VirusShare_0c5e86a7b6f521c6a8fed97491465a0d.exe 216.0KB

Malware - Danger

Bibliography:

AWS SageMaker Endpoint Deployment:

1. Amazon SageMaker Documentation:
   https://docs.aws.amazon.com/sagemaker/index.html

2. Deploying a Model to Amazon SageMaker:
   https://docs.aws.amazon.com/sagemaker/latest/dg/deploy-model.html

3. Deploy a Machine Learning Model with Amazon SageMaker:
   https://aws.amazon.com/getting-started/hands-on/deploy-machine-learning-model-sagemaker/

Streamlit App Creation:

1. Streamlit Documentation:
   https://docs.streamlit.io/

2. Streamlit Gallery:
   https://streamlit.io/gallery

3. Building a Simple Machine Learning Web App with Streamlit:
   https://www.analyticsvidhya.com/blog/2021/06/building-a-simple-machine-learning-web-app-with-streamlit/

Google Colab:

1. Google Colab Documentation:
   https://colab.research.google.com/notebooks/intro.ipynb

2. Google Colab Tutorial for Beginners:
   https://www.geeksforgeeks.org/how-to-use-google-colab/

3. Using Google Colab with AWS Sagemaker:
   https://aws.amazon.com/blogs/machine-learning/using-google-colab-with-aws-sagemaker/