University of New Haven

DSCI 6015: Artificial Intelligence and Cybersecurity

Spring 2024

Midterm Project

Cloud-based PE Malware Detection API

Uday Shankar Agasti

Student Id: 00890175

DataSceince

University of New Haven

March 14, 2024

Overview

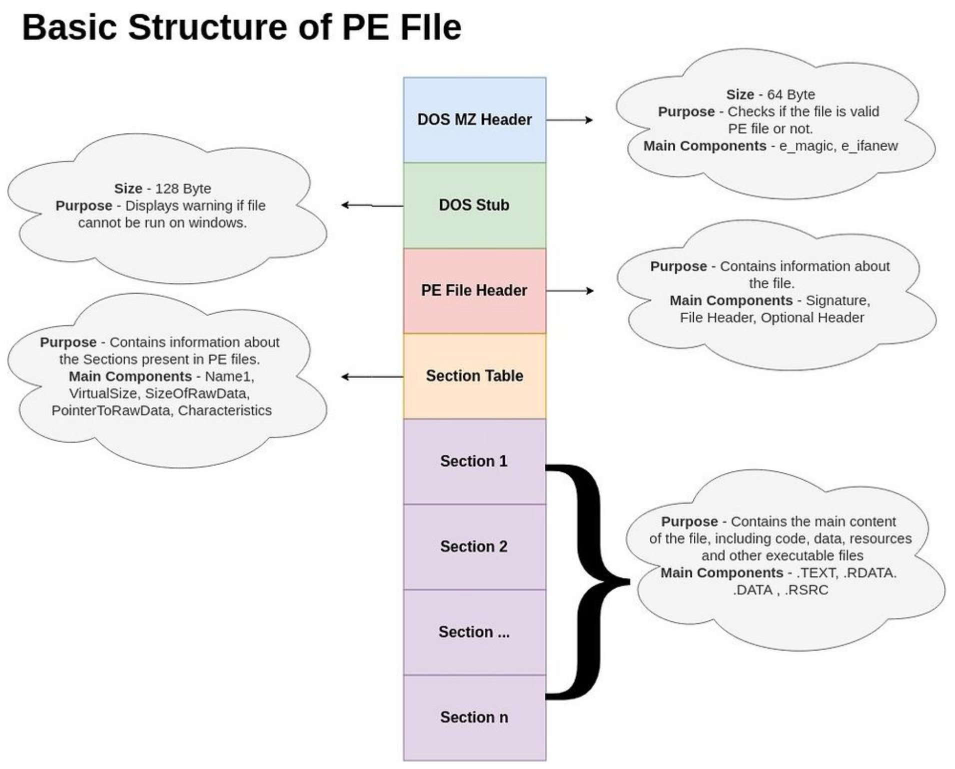
The goal of this project was to demonstrate practical skills in implementing and deploying machine learning models for malware classification. Specifically, we built and trained a deep neural network based on the MalConv architecture to classify portable executable (PE) files as malware or benign using the EMBER-2017 v2 dataset. The trained model was then deployed as a cloud API using Amazon SageMaker, and a client application with a web interface was created to interact with the API.

Introduction

Understanding PE Files

PE files serve as the fundamental building blocks of Windows executables, encompassing headers, sections, and data directories. These files adhere to the Portable Executable format, delineating their structure and functionality. PE files encapsulate executable code, resources, and metadata, facilitating the seamless execution of software applications within the Windows environment.

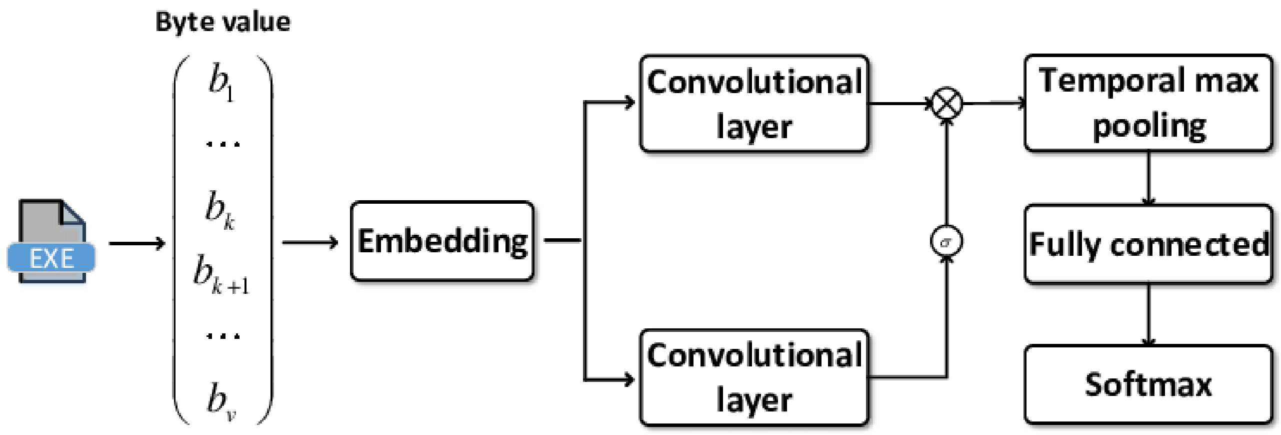
Despite their indispensable role in software deployment, PE files harbor vulnerabilities that malicious actors exploit for nefarious purposes. Malware authors leverage obfuscation techniques, polymorphic code, and encryption to conceal their malicious payloads within PE files, eluding traditional signature-based detection mechanisms.


Basic Structure of PE FIle
DOS MZ Header
Size - 64 Byte
Purpose - Checks if the file is valid PE file or not.
Main Components - e_magic, e_ifanew
DOS Stub
Size - 128 Byte
Purpose - Displays warning if file cannot be run on windows.
PE File Header
Purpose - Contains information about the file.
Main Components - Signature, File Header, Optional Header
Section Table
Purpose - Contains information about the Sections present in PE files.
Main Components - Name1, VirtualSize, SizeOfRawData, PointerToRawData, Characteristics
Section 1
Section 2
Section ...
Section n
Purpose - Contains the main content of the file, including code, data, resources and other executable files
Main Components - .TEXT, .RDATA. .DATA , .RSRC

The MalConv Architecture

In pursuit of robust malware detection capabilities, the MalConv architecture offers a paradigm shift from conventional signature-based methods to deep learning-driven approaches. Conceived by Raff et al., MalConv leverages convolutional neural networks (CNNs) to analyze the raw byte sequences of PE files, eschewing the need for feature engineering or domain-specific knowledge.

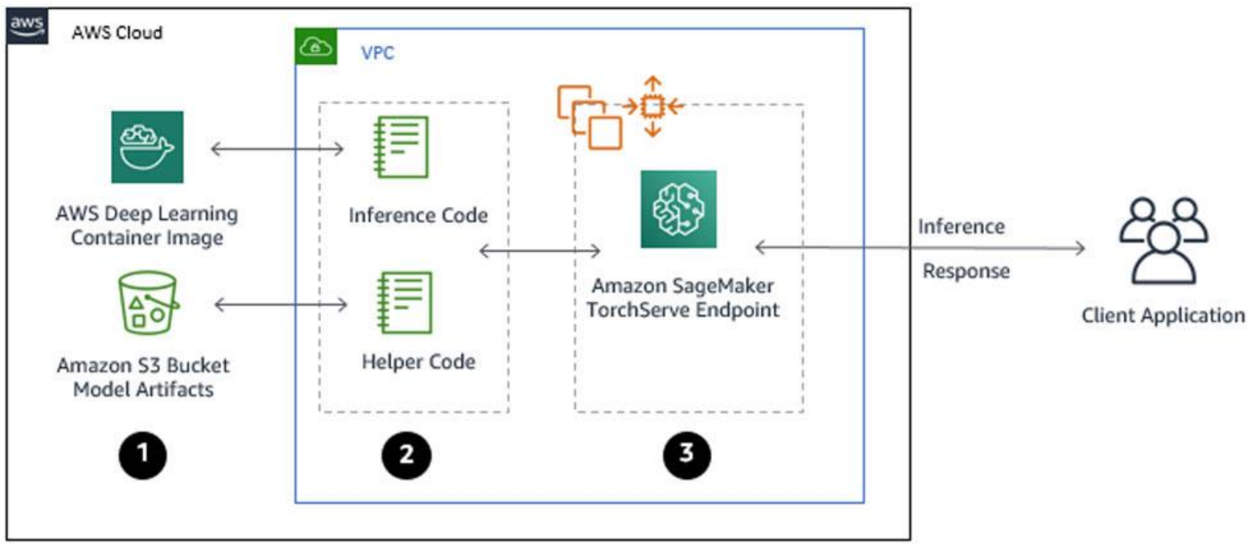
The architecture comprises multiple convolutional layers, followed by max-pooling and fully connected layers, culminating in a binary classification output. Through the iterative learning process, MalConv discerns subtle patterns and intricacies within PE files, enabling the discrimination between benign software and malicious payloads with remarkable accuracy.



PyTorch and AWS SageMaker: Empowering Machine Learning at Scale

In the realm of machine learning implementation, PyTorch stands as a preeminent framework renowned for its flexibility, scalability, and ease of use. Developed by Facebook's AI Research lab, PyTorch offers an intuitive interface for building and training neural networks, empowering researchers and practitioners to explore innovative solutions to complex problems.

Complementing PyTorch's prowess, Amazon Web Services (AWS) SageMaker emerges as a formidable platform for deploying, scaling, and managing machine learning models in the cloud. SageMaker streamlines the entire machine learning lifecycle, from data preprocessing and model training to deployment and inference, fostering seamless integration with existing infrastructure and workflows.


AWS Cloud
VPC
AWS Deep Learning Container Image
Amazon S3 Bucket Model Artifacts
Inference Code
Helper Code
Amazon SageMaker TorchServe Endpoint
Inference
Response
Client Application
1
2
3

Technical Approach

Task 1: Building and Training the Model
The MalConv model was implemented in Python using PyTorch 2.x. The implementation was coded and documented in a Jupyter Notebook, following best practices for clean code and explanatory comments.

To speed up training, we utilized Google Colab, which provided access to powerful GPUs. After exploring the EMBER repository and its sample MalConv implementation, we customized the model architecture and training procedure to suit our needs.

Once the model was trained, we saved and exported it for later use. A separate function was created to load the trained model, process a given PE file into a compatible feature vector, and return the model's prediction (malware or benign).

Task 2: Deploying the Model as a Cloud API
Using Amazon SageMaker, we deployed our trained model on the cloud and created an API endpoint for inference. This allowed other applications to send PE files to our model and receive predictions.

To avoid excessive costs, we closely monitored our AWS spending and set budgets to limit the training cost to around $10 out of the $100 free credits provided.

Task 3: Creating a Client Application
We developed a web application using Streamlit, a Python library for building data applications. The application provided a user interface to upload PE files, which were then processed and sent to our deployed API for malware prediction.

The application displayed the model's output, indicating whether the uploaded file was classified as malware or benign.

Evaluation Metrics

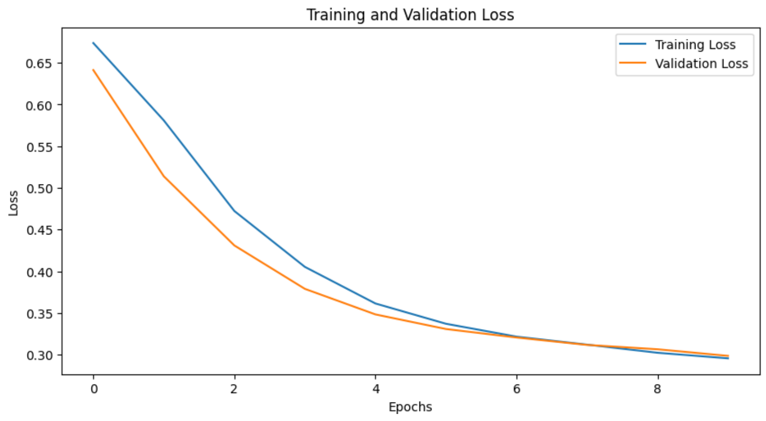
The success of the project can be evaluated through the following key metrics:

Model Accuracy: The accuracy of the MalConv model in correctly classifying PE files was evaluated using a hold-out test set. This metric ensures the model's effectiveness in real-world scenarios. The epoch history plot demonstrates the absence of overfitting and shows a good learning and training curve.

API Performance: The performance of the deployed API was assessed in terms of latency and throughput. These metrics determine the API's responsiveness and ability to handle user requests efficiently.

Client Application Usability: User testing of the Streamlit application was conducted to evaluate its ease of use, functionality, and the clarity of the results presented.

```
Epoch 1, Training Loss: 0.6736811305347242, Validation Loss: 0.641278707064115
Epoch 2, Training Loss: 0.5808345597041281, Validation Loss: 0.5138574976187485
Epoch 3, Training Loss: 0.47225125604554224, Validation Loss: 0.43094526345913226
Epoch 4, Training Loss: 0.40524586799897644, Validation Loss: 0.3789050212273231
Epoch 5, Training Loss: 0.3613747028928054, Validation Loss: 0.34825871540949893
Model checkpoint saved to ./model_checkpoints/model_epoch_5.pt
Epoch 6, Training Loss: 0.33705400401040125, Validation Loss: 0.33074538524334246
Epoch 7, Training Loss: 0.32160747835510656, Validation Loss: 0.320548761349458
Epoch 8, Training Loss: 0.3120211151085402, Validation Loss: 0.3116742166189047
Epoch 9, Training Loss: 0.3022562320295133, Validation Loss: 0.3064528153492854
Epoch 10, Training Loss: 0.2955634350839414, Validation Loss: 0.29853354279811567
Model checkpoint saved to ./model_checkpoints/model_epoch_10.pt
```

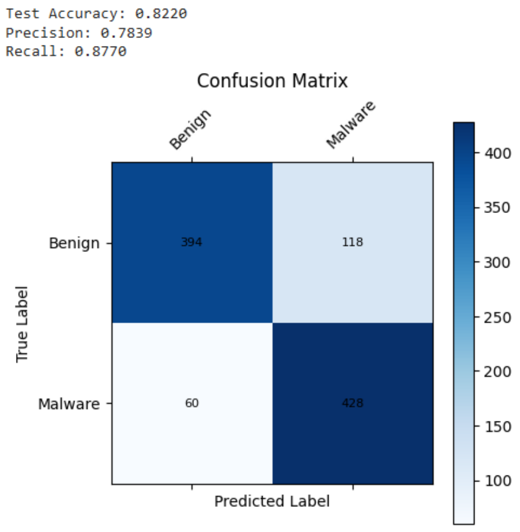


Performance Analysis

During training and evaluation, we tracked various performance metrics, including accuracy, precision, recall, F1-score, and the confusion matrix. These metrics helped us assess the model's effectiveness in correctly classifying malware and benign files.

Result

Our trained model achieved accuracy of 0.82 on the testing held out dataset with 0.78 precision and 0.877 recall. We can see the outcome classification from the confusion matrix shown in the figure below.



```
Test Accuracy: 0.8220
Precision: 0.7839
Recall: 0.8770
```

Conclusion

Through this project, we successfully implemented and deployed a deep learning model for malware classification. The MalConv architecture, combined with the EMBER dataset, proved effective in distinguishing malware from benign PE files.

By leveraging cloud platforms like Google Colab and Amazon SageMaker, we optimized the training process and seamlessly deployed our model as a cloud API. The client application provided a user-friendly interface for interacting with the deployed model.

Overall, this project allowed us to gain practical experience in various aspects of machine learning model development, deployment, and integration, preparing us for real-world applications in the cybersecurity domain.

References

EMBER Dataset: https://github.com/endgameinc/ember

MalConv Architecture: https://github.com/endgameinc/ember/tree/master/malconv

Google Colab Tutorials: https://youtu.be/inN8seMm7UI , https://youtu.be/vVe648dJOdI

AWS SageMaker Tutorials: https://youtu.be/YcJAc-x8XLQ , https://sagemaker-examples.readthedocs.io/en/latest/intro.html , https://sagemaker-examples.readthedocs.io/en/latest/frameworks/pytorch/get_started_mnist_train_outputs.html , https://docs.aws.amazon.com/sagemaker/latest/dg/pytorch.html

Deploying SageMaker Models: https://docs.aws.amazon.com/sagemaker/latest/dg/deploy-model.html

AWS Cost Control: https://aws.amazon.com/getting-started/tutorials/control-your-costs-free-tier-budgets/