



Project Phase 2 Report On

Cloud Based PE Malware Detection

*Submitted in partial fulfillment of the requirements for the
award of the degree of*

Bachelor of Technology

in

Computer Science and Engineering

By

**Nandagovind P (U2003143) Shafer Sulthana (U2003192)
Sheethal Mariya Binoy (U2003195)**

Under the guidance of

Ms. Jomina John

**Department of Computer Science and Engineering
Rajagiri School of Engineering & Technology (Autonomous)
(Parent University: APJ Abdul Kalam Technological University)**

Rajagiri Valley, Kakkanad, Kochi, 682039

April 2024

CERTIFICATE

*This is to certify that the project report entitled "**Cloud Based PE Malware Detection**" is a bonafide record of the work done by **Nandagovind P (U2003143)**, **Shafic Sulthana (U2003192)**, **Sheethal Mariya Binoy (U2003195)**, submitted to the Rajagiri School of Engineering & Technology (RSET) (Autonomous) in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (B. Tech.) in Computer Science and Engineering during the academic year 2023-2024.*

Ms. Jomina John
Asst. Professor
Dept. of CSE
RSET

Ms. Anita John
Asst. Professor
Dept. of CSE
RSET

Dr. Preetha K G
HOD and Professor
Dept. of CSE
RSET

ACKNOWLEDGMENT

We wish to express our sincere gratitude towards **Prof. Dr. Sreejith P S**, Principal of RSET, and **Dr. Preetha K G**, Head of the Department of the Department of Computer Science for providing us with the opportunity to undertake our project, Cloud Based PE Malware Detection.

We are highly indebted to our project coordinators, **Mr. Sajanraj T D** and **Ms. Anita John**, Asst. Professors, Department of Computer Science, for their valuable support.

It is indeed our pleasure and a moment of satisfaction for us to express our sincere gratitude to our project guide **Ms. Jomina John** for her patience and all the priceless advice and wisdom she has shared with us.

Last but not the least, We would like to express our sincere gratitude towards all other teachers and friends for their continuous support and constructive ideas.

Nandagovind P (U2003143)

Shafc Sulthana (U2003192)

Sheethal Mariya Binoy (U2003195)

Abstract

The primary objective of the project is the deployment of machine learning models for PE file classification using the Ember dataset, encompassing various stages from data extraction to real-time classification. The process commences with data extraction, vectorization, and preprocessing, followed by the design and training of a Keras-based neural network model. Various measures are undertaken to optimize the model's performance in distinguishing between malware and benign files. The subsequent task involves deploying the trained model to the AWS cloud platform Sagemaker, thereby establishing an endpoint for seamless real-time classification. The integration of cloud infrastructure not only ensures scalability but also facilitates centralized management of malware detection efforts across diverse environments.

In addition to the technical aspects, the project holds substantial implications in the realm of malware detection. The utilization of deep neural networks introduces automation and scalability, enabling the model to discern intricate patterns within PE files and adapt to evolving malware variants. The deployment on AWS Sagemaker further enhances the system's efficiency by leveraging cloud resources. The inclusion of a Python client script adds a layer of user-friendly accessibility, empowering security analysts to remotely classify PE files without necessitating extensive machine learning expertise. This comprehensive approach, spanning both training and deployment phases, contributes significantly to the advancement of automated malware detection, effectively addressing real-world deployment challenges and fortifying cybersecurity defenses.

Contents

Acknowledgment	i
Abstract	ii
List of Abbreviations	vi
List of Figures	vii
List of Tables	viii
1 Introduction	1
1.1 Background	1
1.2 Problem Definition	2
1.3 Scope and Motivation	3
1.4 Objectives	4
1.5 Challenges	4
1.6 Assumptions	4
1.7 Societal / Industrial Relevance	5
1.8 Organization of the Report	5
2 Literature Survey	6
2.1 Improved Deep Learning Model for Static PE Files Malware Detection and Classification[1]	6
2.2 CloudEyes: Cloud-based Malware Detection with Reversible Sketch for Resource-constrained Internet of Things(IoT) Devices[2]	7
2.3 Malware Detection in Cloud Infrastructures Using Convolutional Neural Networks[3]	8
2.4 Effective Analysis Of Malware Detection In Cloud Computing[4]	9

2.5	Machine Learning based Malware Detection in Cloud Environment using Clustering Approach [5]	10
2.6	Summary and Gaps Identified	10
3	Requirements	13
3.1	Hardware and Software Requirements	13
4	System Architecture	14
4.1	System Overview	14
4.1.1	Feature Extraction and Model Training	15
4.1.2	Model Deployment on AWS SageMaker	15
4.1.3	Prediction with Python Script and AWS SageMaker API	15
4.2	Architectural Design	16
4.3	Module Division	16
4.3.1	Module 1: Data Preparation	16
4.3.2	Module 2: Model Building	18
4.3.3	Module 3: Model Deployment	19
4.3.4	Module 4: Client Creation	19
4.4	Work Breakdown and Responsibilities	20
4.5	Work Schedule - Gantt Chart	20
5	System Implementation	22
5.1	Datasets Identified	22
5.2	Proposed Methodology	23
5.3	Model Architecture Design	24
5.4	Description of Implementation Strategies	26
6	Results and Discussions	29
6.1	Overview	29
6.2	Quantitative Results	31
7	Conclusions & Future Scope	33
	References	34

List of Abbreviations

PE - Portable Executable

API - Application Programming Interface

Adam - Adaptive Moment Estimation

AWS - Amazon Web Services

CLI - Command Line Interface

IoT - Internet Of Things

CNN - Convolutional Neural Network

IaaS - Infrastructure as a Service

VM - Virtual Machine

WFCM-AANN - Weighted Fuzzy K-means Clustering Algorithm with Auto Associative Neural Network

TLSH - Trend Micro Locality Sensitive Hashing

PCA - Principle Component Analysis

FPR - False Positive Rate

SDK - Software Development Kit

IAM - Identity and Access Management

LIEF - Library to Instrument Executable Formats

FNN - Feedforward Neural Network

CSV - Comma-separated values

ReLU - Rectified Linear Unit

S3 - Simple Storage Service

List of Figures

4.1	Architecture Diagram	16
4.2	Sequence Diagram	17
4.3	Work Splitup	20
4.4	Gantt Chart	21
5.1	Model summary	24
6.1	Validation accuracy	29
6.2	Model testing	30
6.3	Endpoint creation	30
6.4	Endpoint name	31
6.5	Endpoint creation	31
6.6	Model accuracy	32

List of Tables

2.1	Comparison of different methods	11
-----	-------------------------------------------	----

Chapter 1

Introduction

In the ever-evolving digital landscape, the increasing reliance of organizations on cloud services for data storage and application hosting has significantly elevated the concern surrounding malware detection in cloud platforms. As businesses migrate their operations to the cloud, the expansive attack surface and interconnected nature of these environments present a ripe opportunity for malicious actors to infiltrate and compromise sensitive data. The escalating potential for malware attacks underscores the paramount importance of deploying robust detection mechanisms to safeguard against evolving threats. This dynamic shift in the technological paradigm necessitates a comprehensive and proactive approach to cybersecurity, blending traditional best practices with cloud-specific strategies. Addressing this challenge requires a nuanced understanding of the unique vulnerabilities and security considerations inherent to cloud environments, coupled with the implementation of cutting-edge technologies and continuous adaptation to emerging threats. In this context, exploring key facets of malware detection, such as endpoint protection, network security, behavioral analysis, and the integration of cloud-native security services, becomes imperative for organizations seeking to fortify their defenses and ensure the integrity of their cloud-based infrastructures. This multifaceted approach serves as a foundational framework to mitigate the risks associated with malware in the cloud, enabling organizations to navigate the digital landscape with resilience and confidence.[6]

1.1 Background

In the contemporary landscape of digital transformation, the widespread adoption of cloud services by organizations has become a defining feature, offering unprecedented advantages in terms of scalability and efficiency. However, this transition has brought forth an equally significant challenge—the escalating risk of malware infiltration in cloud environ-

ments. With an expansive attack surface and interconnectivity between resources, the susceptibility to malicious activities has risen exponentially. The importance of securing cloud platforms cannot be overstated, considering the vast repositories of sensitive data and critical applications hosted within. This project stems from the urgent need to address the vulnerabilities introduced during the migration to the cloud, acknowledging the dynamic and evolving nature of cyber threats. As organizations grapple with the intricacies of securing remote endpoints, ensuring adequate access controls, and meeting stringent regulatory requirements, the imperative to develop and implement effective malware detection mechanisms tailored for cloud environments becomes paramount. By proactively addressing these challenges, this project not only aims to mitigate potential threats but also aligns strategically with organizational goals, contributing to the establishment of a secure, resilient, and trustworthy cloud infrastructure.[6] In examining current scenarios and technological landscapes, the project positions itself at the forefront of safeguarding organizational assets and maintaining the integrity of data in the ever-evolving digital ecosystem.

1.2 Problem Definition

Malware constitutes a significant menace in computer environments, infiltrating through channels like infected data files, insecure APIs, third-party software, and shared resources. This form of cyber threat can lead to dire consequences, including data theft, operational disruptions, and substantial business challenges. The interconnected nature of cloud services facilitates the propagation of malware, making it imperative to address vulnerabilities in data transmission and shared networks. Insecure APIs act as potential gateways for unauthorized access, amplifying the risk of data compromise. Moreover, the use of third-party software introduces an additional layer of susceptibility, with potential vulnerabilities that could be exploited for unauthorized access or data breaches. Understanding and mitigating these multifaceted threats are crucial for fortifying cybersecurity measures in cloud computing environments and ensuring the integrity and security of data and operations.

1.3 Scope and Motivation

The scope of this project is to comprehensively address the multifaceted threat landscape of malware in cloud computing environments. Malicious activities can manifest through various vectors, including infected data files, third-party software vulnerabilities, insecure APIs, and shared resources. The potential consequences are severe, ranging from data theft and business disruption to broader operational challenges. This project aims to develop and implement robust malware detection mechanisms specifically tailored for cloud platforms. By focusing on preventive measures and responsive strategies, the project seeks to safeguard businesses from the diverse means through which malware can infiltrate cloud environments. The scope extends to proactive threat mitigation, emphasizing the significance of protecting against data breaches, operational disruptions, and other associated problems. In essence, the project envisions enhancing the overall security posture of organizations in the cloud, ensuring the integrity of data, and fostering a resilient digital infrastructure in the face of evolving malware threats.

The motivation behind this project lies in the imperative to fortify organizations against the escalating menace of malware in cloud computing environments. As businesses increasingly migrate critical operations to the cloud, the vulnerability to diverse attack vectors such as infected files, third-party software, insecure APIs, and shared resources becomes pronounced. Malware poses a significant threat, capable of stealing sensitive data, disrupting operations, and causing widespread problems. This project is driven by the recognition that a proactive and specialized approach to malware detection in the cloud is crucial for mitigating risks, ensuring data integrity, and preserving the seamless functioning of businesses. By developing targeted detection mechanisms and implementing responsive strategies, the project aims to empower organizations to navigate the digital landscape with enhanced resilience, safeguarding against the pervasive and evolving challenges posed by malware in cloud environments.

1.4 Objectives

- To build and train a deep neural network model by extracting features from an open source dataset EMBER.
- To explore and analyze the EMBER dataset to gain insights into the distribution of features and class balance.
- To optimize hyperparameters of the deep neural network model to improve classification performance.
- To deploy the model into a suitable cloud platform (Sagemaker).
- To classify the Portable executable files as malware or benign using the model endpoint.

1.5 Challenges

The project faces challenges related to data quality and bias, considering the risk of inaccuracies or biases in the labeling of the EMBER dataset. This can impact the model's ability to generalize accurately to new, unseen malware variants, raising concerns about potential overfitting to the training data. Additionally, the deployment on the AWS SageMaker cloud platform introduces risks associated with its reliability, necessitating careful consideration of potential downtimes or disruptions. Furthermore, resource intensiveness for training, particularly with complex models, poses a challenge, requiring a balance between computational needs and budget constraints. Managing these intricacies, from dataset biases to model generalization and cloud platform reliability, will be pivotal for the success and effectiveness of the project in the domain of malware classification.

1.6 Assumptions

- The EMBER dataset is well-labeled, and the labels accurately represent the true nature of the PE files.
- The model will perform well in distinguishing between benign and malicious files.

- The cloud platform selected is suitable for deploying machine learning models and it offers the necessary scalability and infrastructure for hosting the model.
- Sufficient computational resources are available for training, deploying, and maintaining the model

1.7 Societal / Industrial Relevance

This project holds profound societal and industrial relevance by addressing the critical need for enhanced cybersecurity in the era of widespread cloud adoption. As businesses and individuals increasingly entrust sensitive data to cloud platforms, the effective detection and mitigation of malware threats become paramount to safeguarding privacy, economic stability, and the overall integrity of digital ecosystems. The project's outcomes have the potential to bolster public trust in cloud services, protect against economic losses resulting from cyber incidents, and contribute significantly to the collective resilience of societies and industries in the face of evolving cybersecurity challenges.

1.8 Organization of the Report

Introduction of the project is covered in this chapter. The rest of the report is organized as follows: Chapter 2 briefs about the different literature available related to the project. Chapter 3 includes requirements. System architecture is explained in Chapter 4. Chapter 5 includes detailed system implementation and Chapter 6 consists of results and discussions. Chapter 7 concludes the report which is followed by References.

In today's digital world, as more businesses rely on the cloud, there's a rising threat of malware causing problems like stealing data and disrupting operations. This project is about creating better ways to detect and stop malware entering in our system with the help of cloud. By doing this, it helps keep important data safe and ensures businesses can run without interruptions, making our digital space more secure for everyone.

Chapter 2

Literature Survey

2.1 Improved Deep Learning Model for Static PE Files Malware Detection and Classification[1]

This research project delves into the enhancement of static analysis and malware detection, addressing critical issues such as the imbalance in datasets and the time-consuming nature of feature extraction. The conventional challenge associated with static analysis lies in the skewed distribution of data, leading to inaccurate result metrics. Furthermore, the extraction of features from raw binaries is a time-intensive process, especially when employing methods like neural networks that require substantial training time.

To overcome these challenges, the proposed model focuses on feature extraction rather than solely emphasizing the intricacies of model construction. The model efficiently constructs a feature set from the dataset and adeptly classifies static PE (Portable Executable) files.[1] The research underscores the significance of well-extracted features, emphasizing that feeding these features into neural networks with minimal layers can lead to better results. The rationale is that using fewer layers enhances the model's performance, requiring fewer resources and less time for processing and evaluation.

For this study, the EMBER datasets,[1] which encompass PE file information, were utilized. Rigorous feature extraction, data standardization, and cleaning techniques were implemented to rectify imbalances and impurities within the dataset. A total of 2381 features were extracted and pre-processed from both the 2017 and 2018 datasets.

Subsequently, the pre-processed data was input into a deep learning model designed with dense and dropout layers. This architecture aimed to minimize resource strain on the model while delivering accurate results in a more time-efficient manner. Experimental results on the EMBER v2017 and v2018 datasets demonstrated impressive accuracy rates of 97.53% and 94.09%, respectively, surpassing the performance of existing models.

The model’s precision metric achieved 98.85%, outperforming established benchmarks. Notably, the model was trained for ten epochs with a learning rate of 0.01, taking four minutes per epoch, which is a minute less than the Decision Tree model. These outcomes highlight the efficacy of the proposed approach, showcasing both time and precision improvements in static malware analysis and detection.

2.2 CloudEyes: Cloud-based Malware Detection with Reversible Sketch for Resource-constrained Internet of Things(IoT) Devices[2]

Amid the surging frequency of malware attacks on Internet of Things (IoT) devices, particularly those with limited resources, the need for robust security measures has become imperative. Traditional host-based security solutions are proving inadequate in the face of evolving malware threats, and cloud-based alternatives face challenges in simultaneously achieving high-performance detection and safeguarding data privacy. In response to these challenges, this paper introduces CloudEyes, a sophisticated cloud-based anti-malware system designed to deliver efficient and reliable security services for resource-constrained devices.

CloudEyes employs a two-fold approach, addressing both the cloud server and client sides. On the cloud server, it introduces the concept of suspicious bucket cross-filtering, a novel signature detection mechanism rooted in a reversible sketch structure.[2] This innovative approach provides a retrospective and accurate identification of malicious signature fragments, enhancing the system’s ability to detect and combat malware effectively.

On the client side, CloudEyes implements a lightweight scanning agent. This agent utilizes the digest of signature fragments, significantly reducing the computational load and expanding the efficiency of the malware scanning process. The use of signature fragment digests helps streamline the matching process, ensuring a more resource-efficient and faster response to potential threats.

Crucially, CloudEyes prioritizes data privacy and cost-effective communications. It achieves this by transmitting sketch coordinates and employing modular hashing techniques.[2] This ensures that communication between devices and the cloud is not only secure but also resource-efficient, addressing concerns related to data privacy in cloud-based security solutions.

The evaluation of CloudEyes involved real-world scenarios, including campus suspicious traffic and normal files. The results demonstrate the effectiveness and practicality of CloudEyes, showcasing its ability to outperform existing systems in terms of both time efficiency and communication consumption. The findings highlight CloudEyes as a promising solution for fortifying resource-constrained IoT devices against malware threats, offering a nuanced and comprehensive approach to cloud-based anti-malware security.

2.3 Malware Detection in Cloud Infrastructures Using Convolutional Neural Networks[3]

This paper addresses a significant challenge in Infrastructure as a Service (IaaS) clouds, focusing on the vulnerability to malware and the potential for rapid spread within a data-center, leading to disruptive consequences for cloud service providers and their clients. The proposed solution introduces an effective malware detection approach in cloud infrastructure using Convolutional Neural Network (CNN), a deep learning methodology.

The initial implementation involves a standard 2D CNN trained on metadata associated with each process in a virtual machine (VM), extracted through the hypervisor. To enhance classifier accuracy, the paper introduces a novel 3D CNN, where inputs consist of collections of samples over a specified time interval.[3] This innovative approach proves beneficial in reducing mislabeled samples during data collection and training.

Experiments conducted on data obtained by running various malware types, primarily Trojans and Rootkits, on VMs showcase the effectiveness of the proposed models. Notably, the selection of malware for experiments is random, mitigating biases associated with known highly active malware that could be easily detectable. The 2D CNN model achieves an accuracy of 79%, and the introduced 3D CNN model significantly improves accuracy to 90%.[3]

This research not only highlights the potency of CNNs in detecting malware within cloud infrastructure but also emphasizes the practicality of the proposed 3D CNN model in enhancing accuracy.[3] The approach addresses the dynamic nature of malware and contributes to reducing false positives, showcasing its potential applicability for robust malware detection in IaaS clouds.

2.4 Effective Analysis Of Malware Detection In Cloud Computing[4]

In the realm of cloud services, where constant availability and substantial data storage are paramount, the critical considerations of security and adaptability come to the forefront. The accessibility of cloud environments through internet services, however, introduces a vulnerability wherein information stored in the cloud becomes susceptible to both internal and external malware threats. The existing arsenal of malware detection systems, often reliant on machine learning techniques, has faced challenges, particularly in achieving high detection rates.

In response to this challenge, work propose a novel approach – the Weighted Fuzzy K-means clustering algorithm with Auto Associative Neural Network (WFCM-AANN).[4] This approach is designed to enhance the efficacy of malware detection in cloud environments. The integration of the Weighted Fuzzy K-means clustering algorithm provides a sophisticated means of grouping and categorizing data points, allowing for a nuanced understanding of potential threats. The Auto Associative Neural Network further refines the classification process, leveraging its ability to learn intricate patterns and associations within the data.

Through extensive experimentation, the proposed classifier has demonstrated notable success in identifying malware instances. The framework’s distinguishing feature lies in its ability to detect anomalies with a high degree of precision, showcasing a significant improvement over existing classifiers.[4] This consolidated approach not only addresses the limitations of previous machine learning-based malware detection systems but also raises the bar in terms of accuracy and effectiveness.

In essence, the Weighted Fuzzy K-means clustering algorithm, coupled with the Auto Associative Neural Network, offers a comprehensive and advanced solution to the evolving challenges posed by malware in cloud environments. By achieving high detection precision, this proposed framework establishes itself as a promising tool for bolstering the security of cloud services, providing a robust defense against the ever-adapting landscape of malware threats.[4]

2.5 Machine Learning based Malware Detection in Cloud Environment using Clustering Approach [5]

In the context of cloud platforms, ensuring security and resilience poses a critical challenge, given the multitude of diverse applications operating on shared resources. The need for an effective security analysis system within the cloud infrastructure, capable of detecting threats and malware, becomes imperative. While machine learning-driven malware analysis has been extensively researched, existing models often suffer from computational complexity issues and limitations in detection accuracy. In response to these challenges, work present a novel malware detection system leveraging clustering and Trend Micro Locality Sensitive Hashing (TLSH).[5]

This approach involves the utilization of the Cuckoo sandbox, a dynamic analysis tool providing reports on file behavior within an isolated environment. To extract essential features from the malware reports obtained from the sandbox, work introduce a novel feature extraction algorithm. Subsequently, they employ principal component analysis (PCA), random forest, and Chi-square feature selection methods to identify the most critical features for analysis.[5]

The experimental phase involves assessing the performance of the model using clustering and non-clustering approaches on three classifiers: Decision Tree, Random Forest, and Logistic Regression. The results showcase superior classification accuracy and a lower false positive rate (FPR) compared to state-of-the-art works and non-clustering approaches, all achieved at a significantly reduced computation cost.

In summary, the proposed malware detection system introduces a unique blend of clustering and TLSH, overcoming computational complexity challenges and enhancing detection accuracy.[5] Leveraging feature extraction algorithms and selective feature methods, the model outperforms existing approaches in terms of accuracy and computational efficiency. This innovative system contributes to reinforcing the security and resilience of cloud platforms against malware threats while minimizing computational overhead.

2.6 Summary and Gaps Identified

Various advantages and disadvantages of the four papers discussed above are given in the following table.

Paper	Methodology	Advantage	Disadvantage
[3]	2D CNN combining with 3D CNN structure	Realistic data collection	Considered as “black box” model hence lack of explainability
[4]	Unified WFCM-AANN system	High detection sensitivity	Complex, Resource intensive
[2]	CloudEyes: suspect bucket cross filtering; a signature detection mechanism	Reduced time consumption	Not effective at detecting more complex malware threats
[5]	Based on the clustering and trend micro-locality sensitive hashing	Identify and group similar malware samples	Large dataset

Table 2.1: Comparison of different methods

The lack in the current state of art are as follows:

- **Computational Complexity:** Many existing malware detection systems exhibit high computational complexity, impacting their efficiency, especially in real-time scenarios within cloud environments.
- **Detection Accuracy:** Some machine learning-driven models in the current state of the art may suffer from relatively lower detection accuracy, making them less reliable in identifying sophisticated and evolving malware threats.
- **Adaptability to Dynamic Threats:** There is a notable challenge in effectively addressing the dynamic nature of malware. Many current systems struggle to adapt promptly to the rapid evolution of malicious tactics, techniques, and procedures.
- **Integration with Threat Intelligence Feeds:** Comprehensive integration with dynamic threat intelligence feeds is lacking in certain systems, limiting their ability to stay updated with the latest threat information and proactively respond to emerging threats.

- Behavioral Analysis: Some systems may lack advanced behavioral analysis capabilities, which are crucial for understanding the runtime behaviors of files and identifying unknown threats based on their actions.
- User Engagement and Education: The current state of the art may not fully leverage user-friendly interfaces and educational features to empower end-users with information about potential threats, limiting overall cybersecurity awareness.

Addressing these limitations is pivotal to advancing the effectiveness and efficiency of malware detection systems in enhancing cybersecurity measures.

In conclusion, this chapter highlighted the imperative of advancing malware detection systems in the face of evolving threats. While current systems show promise, there are notable challenges in computational complexity, detection accuracy, and adaptability to dynamic threats. Integrating advanced techniques, such as dynamic threat intelligence feeds and behavioral analysis, is crucial for proactive threat detection. The future success of these systems relies on addressing these shortcomings and fostering a more resilient cybersecurity landscape.

Chapter 3

Requirements

3.1 Hardware and Software Requirements

Software:

- Ember dataset: Obtain the Ember dataset for training and evaluating the machine learning model for malware classification.
- Jupyter Notebook or Google Colab or any Python IDE: Choose a suitable integrated development environment for coding, model development, and experimentation.
- Deep Learning framework: TensorFlow, Keras: Utilize TensorFlow and Keras for designing, building, and training the deep neural network model for PE file classification.
- AWS CLI: Install and configure the AWS Command Line Interface to interact with AWS services, facilitating the deployment and management of the model on the AWS cloud.
- AWS Sagemaker SDK: Use the AWS Sagemaker SDK to streamline the deployment process and manage the endpoint for real-time classification in the cloud.

Hardware:

- Powerful CPU/GPU: Employ a machine with a robust central processing unit (CPU) and, if possible, a graphical processing unit (GPU) to accelerate the training of the deep neural network.
- Minimum 8 GB RAM: Ensure a minimum of 8 gigabytes of Random Access Memory (RAM) to support the processing and storage requirements during model development and training.

Chapter 4

System Architecture

The project aim to tackle the challenge of malware classification using PE (Portable Executable) files by leveraging the EMBER-2017 v2 dataset.[1] Approach encompasses feature extraction from PE files, the design and training of a deep neural network for effective malware classification, and the subsequent deployment of the trained model on Amazon Web Services (AWS) SageMaker for real-time predictions. To initiate the process, we employ the EMBER-2017 v2 dataset to extract relevant features from PE files, employing the Ember library.[7] Subsequently, design a deep neural network for malware classification, train it using the extracted features, and evaluate its accuracy. Following successful training, save the model for future use. Moving into the cloud infrastructure, setup AWS services, create an S3 bucket, and establish an IAM role with necessary permissions for SageMaker. The trained model is then pushed to the S3 bucket for seamless integration with SageMaker. After deploying the model on AWS SageMaker, create a Python script to convert a given PE file into a feature vector using Ember. Finally, integrate this script with the SageMaker API to obtain predictions, enabling us to efficiently classify the nature of a PE file as either benign or malicious.

4.1 System Overview

The comprehensive system designed for malware classification involves various components and services seamlessly integrated into a coherent workflow. The system overview encompasses the feature extraction process, model training, deployment on AWS SageMaker, and the creation of a Python script for real-time predictions.[1]

4.1.1 Feature Extraction and Model Training

Feature extraction in the project utilizes the Ember-LIEF approach, where PE files undergo processing with the Ember library, leveraging the LIEF project library for extracting diverse features such as header information, sections, functions, and resources. The extracted features are then vectorized and stored as CSV files. Moving to the deep neural network design and training phase, a Feedforward Neural Network (FNN) is crafted using Keras. The input layer aligns with nodes representing the extracted features, as per Ember-LIEF processing. Hidden layers incorporate ReLU activation functions to capture complex relationships, while the output layer employs sigmoid activation for binary classification. Training the model involves using scaled data from the EMBER dataset, and hyperparameters are optimized to enhance performance. Following the training, the model is saved and uploaded to Google Drive for future use. The integration with AWS involves setting up services such as Amazon S3 for storage, AWS SageMaker for model deployment, and IAM for access management, providing a comprehensive infrastructure for deploying and managing the machine learning model in the cloud.

4.1.2 Model Deployment on AWS SageMaker

Amazon S3 storage serves as the repository for the saved model and its related resources, providing a scalable and durable storage solution. The model, once saved, is seamlessly uploaded to AWS SageMaker for deployment, where it transforms into an easily accessible endpoint capable of delivering real-time predictions. To fortify the security of the deployed model and associated resources, Identity and Access Management (IAM) protocols are employed. IAM ensures stringent access control, allowing only authorized entities to interact with the model, thereby safeguarding against unauthorized access and bolstering the overall security posture of the deployed machine learning solution.

4.1.3 Prediction with Python Script and AWS SageMaker API

A Python script for PE file analysis has been crafted, leveraging Ember for feature extraction and scaling, and incorporating the boto3 library to establish a connection with the AWS SageMaker API, as documented in [1]. This script seamlessly integrates with the AWS SageMaker API, transmitting the feature vector to the endpoint, thereby

invoking the model to deliver real-time predictions concerning the nature of the PE file, as elucidated in [6]. Following the analysis, the script succinctly prints the results retrieved from the cloud API, providing clear indications of whether the PE file is classified as Malware or Benign. The approach outlined in this script ensures efficient and remote PE file classification, demonstrating the practical implementation of the model deployed on the AWS SageMaker platform.

This system architecture ensures a streamlined and automated workflow for malware classification, from feature extraction and model training to cloud-based deployment and real-time prediction through a user-friendly Python script. The modular design enables flexibility, scalability, and ease of maintenance in addressing evolving cybersecurity challenges.

4.2 Architectural Design

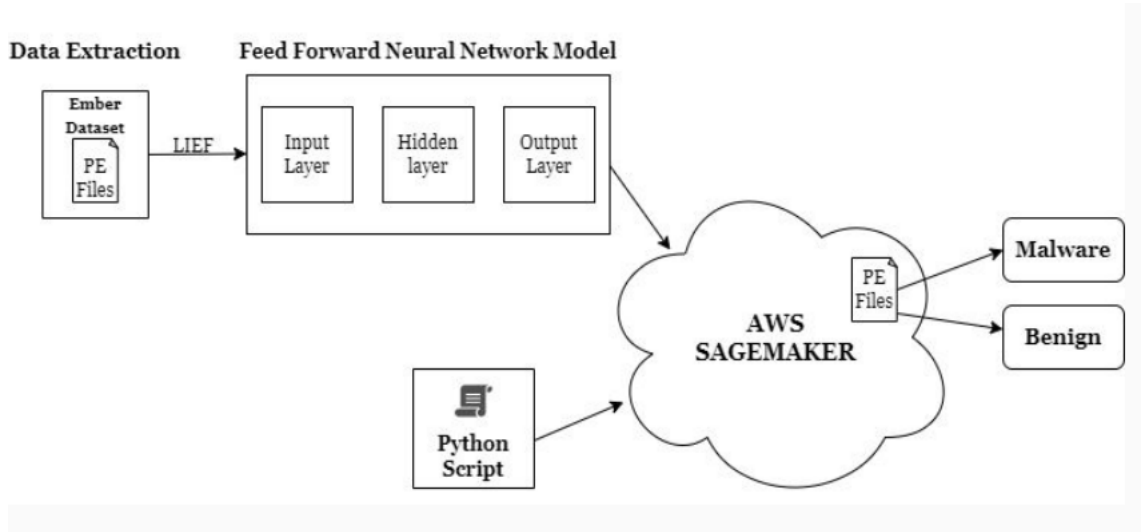


Figure 4.1: Architecture Diagram

4.3 Module Division

4.3.1 Module 1: Data Preparation

To facilitate effective malware classification, we employ the EMBER-2017 v2 dataset and utilize the Ember library, which seamlessly integrates with the LIEF project library for extracting diverse features from PE (Portable Executable) files. The extracted features cover

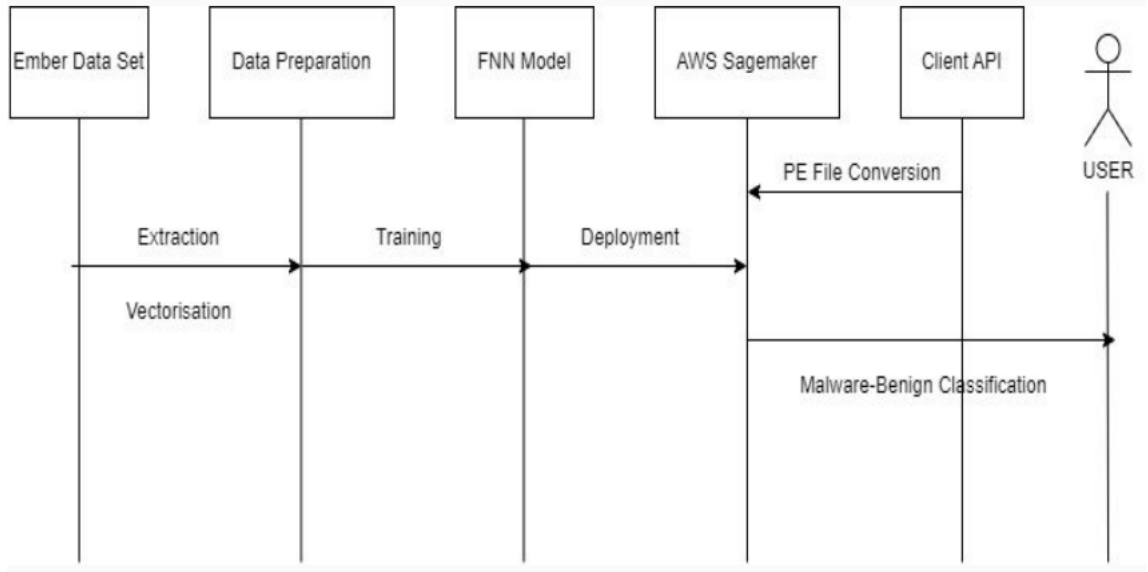


Figure 4.2: Sequence Diagram

crucial aspects such as header information, sections, functions, and resources. Leveraging the LIEF project library ensures a comprehensive and standardized extraction process, consolidating relevant PE file attributes into a structured JSON format.[8]

Once the features are extracted, we proceed to vectorize them into a binary format, a crucial step in preparing the data for machine learning applications.[7] This transformation allows us to represent the diverse set of features as a cohesive vector, creating a format suitable for subsequent analysis and model training. The vectorized features are then saved in CSV files, providing a convenient and universally supported format for storage and further processing.

To optimize the storage and access of the dataset, we employ the h5py library for serializing the data. Serialization enhances the efficiency of data storage and retrieval, ensuring that the dataset remains accessible and manageable throughout the various stages of model development and deployment.

Furthermore, to enhance the robustness of the dataset, we incorporate scaling techniques using Scikit Learn's scalars, such as the Robust Scalar. This ensures that the features are appropriately scaled, mitigating the impact of outliers and variations in the data distribution. Scaling is a crucial preprocessing step that contributes to the stability and performance of machine learning models by ensuring consistent and meaningful feature representations.

In summary, our approach involves leveraging the Ember library with the LIEF project for feature extraction, vectorizing the extracted features into a binary format and saving them as CSV files, serializing the dataset using h5py, and applying scaling techniques to enhance the robustness and effectiveness of the dataset for subsequent deep learning model training and deployment.[7]

4.3.2 Module 2: Model Building

To address the challenge of malware classification, we design a robust Feedforward Neural Network (FNN) architecture using the Keras deep learning framework. The input layer is defined to accommodate nodes representing the various extracted features from PE files. This neural network architecture incorporates one or more hidden layers, utilizing the rectified linear unit (ReLU) activation function to capture intricate relationships within the data. For the output layer, we employ the sigmoid activation function, suitable for binary classification tasks like malware detection.[6]

During the training phase, the model is fed with scaled data from the EMBER dataset, ensuring that the features are appropriately normalized for effective learning. Hyperparameters, including the learning rate, number of hidden layers, and nodes per layer, are optimized to enhance the model's performance. The Adam optimizer, known for its efficiency in adapting learning rates, is employed during the compilation of the model.

As a performance metric, accuracy is utilized to evaluate the model's effectiveness in correctly classifying malware instances. After successful training, the model is saved and uploaded to Google Drive for future use, ensuring easy access and reproducibility.

To predict the nature of a PE file using the trained model, a dedicated method is developed. This method takes a PE file as input, extracts the relevant features, scales them appropriately, and feeds them into the trained FNN for prediction. The outcome provides insight into whether the PE file is benign or malicious, offering a practical and efficient solution for real-world applications in cybersecurity.

In summary, our approach involves the design of a Feedforward Neural Network using Keras, optimization of hyperparameters for enhanced performance, training on scaled data from the EMBER dataset, utilization of accuracy as a performance metric, saving and uploading the trained model to Google Drive, and developing a method for predicting the nature of PE files based on the trained FNN architecture.[1]

4.3.3 Module 3: Model Deployment

To deploy our trained malware classification model on AWS, we begin by setting up the necessary services for seamless integration and deployment. First, we utilize Amazon S3 (Simple Storage Service) to store our saved model and any related resources. Amazon S3 provides a scalable and secure object storage solution, ensuring efficient access to the model artifacts.

Next, we leverage AWS SageMaker, a comprehensive service designed to simplify the entire machine learning workflow, including building, training, and deploying machine learning models. In this context, we upload the saved model to AWS SageMaker, utilizing its capabilities to manage and deploy machine learning models in a scalable and cost-effective manner. SageMaker facilitates the seamless deployment of models, enabling us to transition from training to inference with minimal effort.[4]

To control access and permissions within the AWS environment, we employ IAM (Identity and Access Management). IAM allows us to define roles and access policies, ensuring that only authorized entities can interact with the deployed model and associated resources. This security measure enhances the overall integrity of the deployment and safeguards sensitive data and model artifacts.

In summary, the model deployment on AWS involves the utilization of Amazon S3 for storing model artifacts, AWS SageMaker for building and deploying ML models, and IAM for robust identity and access management. This orchestrated setup streamlines the deployment process, ensuring a reliable and secure infrastructure for serving our malware classification model in a production environment.[5]

4.3.4 Module 4: Client Creation

To seamlessly integrate the malware classification model into the cloud and obtain predictions for a given PE file, a Python script is developed to convert the PE file into a feature vector compatible with the trained model. The script leverages the Ember library for feature extraction, ensuring consistency with the training data. Additionally, the boto3 library is employed to establish a connection to the AWS SageMaker API.[4]

The Python script follows a structured approach, beginning with the extraction of features from the PE file using Ember’s feature extraction capabilities. Subsequently, the feature vector is constructed and appropriately scaled to match the input requirements of the model. The script then establishes a connection to the AWS SageMaker API using the boto3 library, enabling seamless communication with the deployed model.

Upon sending the feature vector to the cloud API, the script captures the predictions returned by the model.[1] The results, indicating whether the PE file is classified as Malware or Benign, are printed as an output. This streamlined process ensures that PE files can be easily analyzed and categorized without the need for intricate manual intervention.

In summary, the Python script developed integrates Ember for feature extraction, scales the feature vector, connects to the AWS SageMaker API using the boto3 library, sends the feature vector for prediction, and prints the results obtained from the cloud API. This script serves as a practical and efficient solution for obtaining real-time predictions on the nature of PE files based on the deployed malware classification model.[1]

4.4 Work Breakdown and Responsibilities

NANDAGOVIND P	SHAFK SULTHANA	SHEETHAL MARIYA
<ul style="list-style-type: none"> • Base paper selection • Dataset extraction • Model evaluation • Model saving • Endpoint creation 	<ul style="list-style-type: none"> • Documentation • Dataset vectorization • Model training • Model tuning • Model upload 	<ul style="list-style-type: none"> • Documentation • Data preprocessing • Model architecture design • Implementation • AWS setup

Figure 4.3: Work Splitup

4.5 Work Schedule - Gantt Chart

In this chapter, we have outlined a comprehensive system for malware classification, integrating various components and leveraging both local and cloud-based services. The



Figure 4.4: Gantt Chart

process begins with feature extraction from Portable Executable (PE) files using the Ember library, with feature vectors stored in CSV format. A Feedforward Neural Network (FNN)[1] is designed using Keras, trained on scaled data from the EMBER dataset, and optimized for enhanced performance. The trained model is saved and uploaded to Google Drive for future use. AWS services, including Amazon S3 for storage, AWS SageMaker for model deployment, and IAM for access management, are utilized for cloud integration. A Python script is developed to extract features from PE files using Ember, scale the feature vector, and connect to the AWS SageMaker API for real-time predictions.[4] The system ensures a seamless workflow, combining effective feature extraction, robust model training, cloud-based deployment, and user-friendly real-time prediction capabilities. This integrated approach provides a scalable and efficient solution for addressing malware classification challenges in cybersecurity.

Chapter 5

System Implementation

This chapter delves into the datasets that are relevant to the creation of a deep neural network model that will identify whether Portable Executable (PE) files are malicious or not. The main dataset that is being examined is the EMBER dataset, which is a large collection of dangerous and benign PE files that have been carefully cleaned up and organized to make it easier to train and test malware detection algorithms. We prepared the groundwork for reliable model training and evaluation by carefully analyzing the feature sets and version upgrades of the EMBER dataset. The suggested technique is then described, along with the steps that must be followed in order to create, train, and use the deep neural network model. In order to enable real-time PE file classification, special emphasis is given on the integration of machine learning with cloud deployment, specifically on Amazon SageMaker. In conclusion, we offer a comprehensive analysis of the model architectural design, clarifying the nuances of the 'myModel' neural network and its optimization for effective PE file processing and precise categorization. We highlight the usefulness and efficacy of cybersecurity apps by laying the foundation for a workable and scalable malware detection solution with this thorough investigation.

5.1 Datasets Identified

The EMBER dataset, formally named the Elastic Malware Benchmark for Empowering Researchers (EMBER), is a comprehensive collection comprising both malicious and benign Portable Executable (PE) files. This meticulously processed and structured dataset is designed to aid in training and evaluating the effectiveness of malware detection algorithms. It includes a substantial training set of 900,000 samples and a testing set of 200,000 samples. The release of EMBER dataset version 2 (v2) in 2018 marked a significant advancement. This version introduced a new feature set called Data Directory,

enhancing the dataset’s usefulness for researchers and cybersecurity practitioners.

Within the EMBER dataset, [6] various feature sets are provided, such as Byte Histogram (BH), Byte-Entropy Histogram (BE), String Information (ST), General File Information (GE), Header Information (HE), Section Information (SE), Imported Functions (IM), Exported Functions (EX), and Data Directory (DD). These feature sets are abbreviated for ease of reference and analysis. Feature extraction is a critical step in utilizing the dataset effectively. The EMBER dataset offers code to vectorize the data, allowing researchers to convert raw file data into structured feature vectors suitable for machine learning models. The Hashing Trick is a dimension reduction technique with a strong theoretical basis, is employed in feature extraction to manage dataset complexity and enhance computational efficiency.

To ensure consistency and balance among feature sets during model training, feature scaling is implemented. This process adjusts the scale of individual features to maintain uniformity and prevent any single feature set from disproportionately impacting model performance. Researchers utilizing the EMBER dataset are actively exploring methods to optimize model performance, including investigating approaches to reduce the number of feature sets for improved efficiency and effectiveness in malware detection tasks. In summary, the EMBER dataset serves as a valuable tool for advancing cybersecurity research and developing robust malware detection solutions.

5.2 Proposed Methodology

The project is to create and implement a deep neural network model that can identify malicious or benign Portable Executable (PE) files. Features are taken from the EMBER dataset, trained on it, and then the model is put to use on Amazon SageMaker for real-time categorization.

The proposed methodology involves a number of stages in order to create, train, implement, and use a deep neural network model to categorize PE files as either benign or malicious. To begin with, the EMBER dataset is used, and features are taken out of it to describe the portable executable (PE) files. In order to get these features ready for input into the neural network model, preprocessing techniques like vectorization and scaling are applied. With input layers, hidden layers, and activation functions, the model’s se-

quential architecture is specifically crafted to efficiently train and categorize the features collected from the PE files. The model is built using the Adam optimizer during training, and its performance is assessed using a 480,000 sample training set and a 120,000 sample validation set spread across 30 epochs with a batch size of 256.

The model is saved and ready for deployment on the Amazon SageMaker cloud platform when training yields adequate accuracy. In this case, the model is made available through an endpoint, enabling real-time PE file classification via a cloud API. To make the process of reading external PE files, transforming them into feature vectors that the model can use, and getting predictions from the deployed model endpoint easier, a Python script is created. The process of classifying PE files is streamlined by this machine learning and cloud deployment integration, providing a workable and scalable malware detection system. All things considered, the project highlights a thorough methodology with an emphasis on usability and practicality, including everything from model creation and training to cloud deployment and real-time categorization.

5.3 Model Architecture Design

Model: "sequential"		
Layer (type)	Output Shape	Param #
dropout (Dropout)	(None, 2381)	0
dense (Dense)	(None, 1000)	2382000
dropout_1 (Dropout)	(None, 1000)	0
dense_1 (Dense)	(None, 1)	1001
Total params: 2383001 (9.09 MB)		
Trainable params: 2383001 (9.09 MB)		
Non-trainable params: 0 (0.00 Byte)		

Figure 5.1: Model summary

With an emphasis on malware identification, the 'myModel' neural network is designed for binary classification tasks. Its architecture, which minimizes overfitting by combining layers like batch normalization, dense, and dropout, is painstakingly created to extract and analyze features from input data. Model seeks to maximize performance and precisely forecast class labels with a high degree of precision by utilizing methods such as regularization and sigmoid activation.

Particularly for binary classification applications, the 'myModel' neural network architecture is optimized for efficient processing and categorization of input data. The first step involves defining an input layer with an input shape of 2381, which signifies the anticipated quantity of features present in the input data. This input layer provides the first stage of processing by acting as the point of entry for data into the network. Dropout layers are incorporated strategically after the input layer in order to reduce overfitting. During training, these dropout layers randomly deactivate a percentage of neurons to introduce noise and keep the network from becoming unduly dependent on any one attribute. Through the promotion of learning more diverse representations in the data, this dropout regularization improves the generalization performance and robustness of the model.

Keeping with the design, the network is made up of multiple thick layers, each of which has the goal of gradually extracting and processing information from the incoming data. With 1000 units, the first dense layer makes use of the sigmoid activation function. Each neuron's output is squashed by sigmoid activation to a range of 0 to 1, allowing the model to predict binary outcomes with probability. Furthermore, this layer's activity is subjected to L2 regularization, which penalizes large weights in an effort to discourage overfitting and encourage generalization. Another dropout layer is added after the first dense layer to improve regularization even more by deactivating neurons at random.

The input data processing is continued with 512 units and sigmoid activation in the next dense layer. Capturing more intricate patterns and representations in the data is the responsibility of this layer. After the dense layer, a batch normalization layer is added to speed up and stabilize the training process. By normalizing the activations of the preceding layer, batch normalization guarantees that the standard deviation and mean activation are nearly equal. During training, this normalization strategy helps to increase gradient flow and accelerate convergence.

Afterwards, a 256-unit dense layer with sigmoid activation is implemented. By further honing the learnt characteristics and input data representations, this layer may be able to capture higher-level abstractions. Last but not least, a single neuron with sigmoid activity makes up the output layer. In a binary classification scenario, this neuron outputs the likelihood that the input data belongs to the positive class. The Adam optimizer, which has a learning rate of 0.001, is used for optimization, making weight changes during training more effective. The major assessment parameter is accuracy, and the model is trained using binary cross-entropy loss, which quantifies the difference between real labels and projected probabilities.

Dropout regularization, batch normalization, and suitable activation functions are just a few of the elements included in the 'myModel' architecture, which efficiently processes and categorizes input data while preventing overfitting and guaranteeing model stability and convergence during training.

5.4 Description of Implementation Strategies

The purpose of this project is to deploying machine learning models for malware classification. This project comprises of three tasks. The initial task is to train a deep neural network to classify PE files as malware or benign using Ember opensource dataset, EMBER-2017 v2 available at [9]. The second task deals with deploying the model to cloud and creating an endpoint (API) to the model. As a final task, create a client nothing but a python script that loads a PE file and classify it as malicious or benign. The requirements of this project are access to Google CoLab and Amazon Sagemaker. Working in these services is advisable for this project. This project is implemented task wise. So, the details in the report are also given based on the tasks performed. This task has three parts. They are data extraction & preprocessing, model architecture & training and Testing the model. The parts of this task are detailed below.

Initially, a python notebook is created in Google Collaboratory to perform the required executions for this task. As an initial part of this task, the data needs to be extracted and vectorized for the neural network training. For this, Ember uses LIEF project library to extract features from PE files included in the EMBER dataset. Raw features are extracted to JSON format. Vectorized features can be produced from these raw features and saved

in binary format from which they can be converted to CSV, dataframe, or any other format. By using Ember's library, the dataset is extracted and vectorized features and is stored in four CSV files, training dataset features, training dataset labels, testing dataset features and testing dataset labels. The train dataset has 900k samples and the test dataset has 200k samples. The number of features in this dataset are 2381 excluding the label or target data. ign and malicious. They are represented as -1, 0 and 1 respectively. The train dataset is equally divided among the three categories i.e., 300k samples of each category. It is the same in test dataset also. But it can be seen that the test dataset has only benign and malicious samples. So, the test dataset has 100k benign samples and 100k malicious sample. In this project, unlabeled samples are ignored from the train dataset for the better performance of the model. By executing the required code snippets to just load and vectorize the data in Google CoLab notebook, the notebook session kept on crashing. This results in rerunning the entire process again. So, as soon as the data files are loaded, they are pickled to avoid RAM crashes and the long execution time. The pickled data files are then saved in the system for easy access. The dataset is huge and so when pickling the data files, can see that this process used most of the 25GB RAM available in CoLab. So, even though the datasets are pickled, the RAM crashes. The alternative for this is to create HDF5 files for all the four dataset files as shown in the image. The h5py package is a Pythonic interface to the HDF5 binary data format. The serialized h5 files are uploaded into Google drive or can also be downloaded into local computer for future use. As a part of preprocessing the vectorized features, scaling of the data is done using Scikit Learn's different scalars like Standard Scalar, Minmax Scalar and Robust Scalar. Out of them, Robust Scalar was picked for the feature scaling of this dataset. Both the training and testing feature dataset are scaled separately using Robust Scalar. Then the scaled data is loaded into the neural network model. One can even serialize the above obtained scaled feature data and store it in Google drive or download to local computer for future use.

Now comes designing the architecture of the neural network model in Keras. For the model, it is build it with a simple and typical dense layers and dropout layers. The dropout layers in the model helps in avoiding overfitting of the model and makes the model more generalized. The number of input nodes to the model is same as the features count in the dataset i.e., 2381. The model architecture is wide instead of deep so, I chose

only 2 dense layers and 2 dropout layers. Apart from the shown, the activation functions used are relu for hidden layer and sigmoid for the output layer. The optimizer used while model compilation is Adam, loss function is binary crossentropy and the performance metric is accuracy. The model training is done in batches of 256 samples. During the training, the train data splits into train and validation data in the ratio of 8:2 i.e., out of 600k samples, 480k samples are training data and remaining 120k samples comprise of validation data. Then the model is trained with 30 epochs. The hyperparameter choice of the model is made by testing different combination of hyperparameters and finalized them based on the better model performance accuracy. The model is tested on the test data. The trained mode is saved and uploaded to Google Drive for future use. As a part of testing the model, create a function that takes the PE files as an argument, runs it through the trained model and returns the nature of the PE files, Malware (1) or Benign (0). To do this testing, download Anaconda PE file using wget command and passed it to the function created to test the file. To deploy the model to cloud (AWS), create a notebook instance in AWS Sagemaker and create a notebook where all the executions are done. Then import the required libraries for the creation of the endpoint for the model. Upload the saved model and model weights to the notebook instance. The creation of endpoint took nearly 9 min. The endpoint name needs to be noted to use it further.

In the next task, a python code is created that takes PF file as argument and returns the nature of the file. All the within operations requires to reach the result are done in cloud API created in the previous task. Connection to the AWS Sagemaker API can be done by using boto3 library and specifying required keys and token ids of the AWS CLI. The PE file is parsed and features are extracted from it using the Ember's feature extractor class and the data is dumped into the endpoint using the obtained endpoint from the previous set. Before executing this file, one should install all the requirements needed for the execution of Ember libraries. Then download any PE file, Anaconda's, Git's and Grammarly's PE files are chosen and execute the python file.

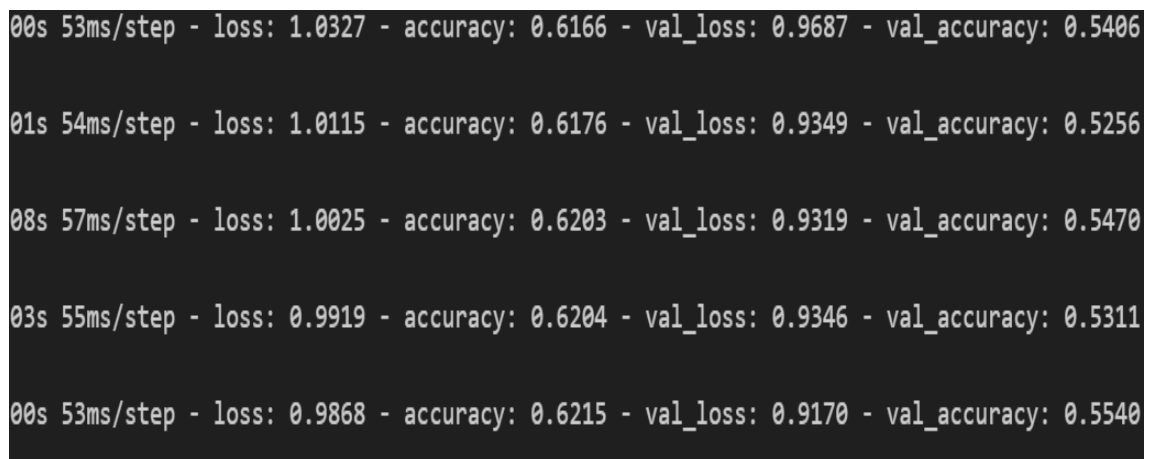
Chapter 6

Results and Discussions

This chapter presents the deployment and evaluation of a cloud-based PE malware detection system on Amazon SageMaker. The deployment process involved creating an endpoint for API access with the generated endpoint. The model's performance accuracy was found to be 62.87%. These results demonstrate the system's effectiveness in real-world applications, offering insights into its deployment and performance evaluation.

6.1 Overview

The model performance accuracy obtained in overall is 62% with a fluctuating train accuracy of 62% and validation accuracy of 55%. The hyperparameter choice of the model is made by testing different combination of hyperparameters and finalized them based on the better model performance accuracy. The model is tested on the test data and the obtained model performance accuracy is 62.87%.



Time	Step	loss	accuracy	val_loss	val_accuracy
00s	53ms/step	1.0327	0.6166	0.9687	0.5406
01s	54ms/step	1.0115	0.6176	0.9349	0.5256
08s	57ms/step	1.0025	0.6203	0.9319	0.5470
03s	55ms/step	0.9919	0.6204	0.9346	0.5311
00s	53ms/step	0.9868	0.6215	0.9170	0.5540

Figure 6.1: Validation accuracy

The model is then tested on some external PE files such as Anaconda and Git PE files respectively. See Fig 6.2. The model predicted those files as benign files.

```
testPE("Anaconda3-2020.02-Windows-x86_64.exe")
✓ 44.1s

WARNING: EMBER feature version 2 were computed using
WARNING: lief version 0.13.2-2d9855fc found instead
WARNING: in the feature calculations.
1/1 [=====] - 0s 183ms/step

'benign'

testPE("Git-2.43.0-64-bit.exe")
✓ 7.9s

WARNING: EMBER feature version 2 were computed using
WARNING: lief version 0.13.2-2d9855fc found instead
WARNING: in the feature calculations.
1/1 [=====] - 0s 416ms/step

'benign'
```

Figure 6.2: Model testing

The model was successfully deployed on Amazon SageMaker, and an endpoint was created for API access. The deployment process took approximately 4 minutes and 24 seconds. The endpoint name 'sagemaker-tensorflow-serving-2024-04-06-18-33-45-013' was generated, indicating successful creation. This endpoint allows other applications to interact with the deployed model, enabling cloud-based PE malware detection.

```
%%time
predictor = sagemaker_model.deploy(initial_instance_count=1,
| | | | | | | | | instance_type='ml.m5.large')

----!CPU times: total: 3.75 s
Wall time: 4min 24s
```

Figure 6.3: Endpoint creation

The endpoint name can be obtained as shown:

```
predictor.endpoint

WARNING:sagemaker.deprecations:The endpoint attribute has been deprecated.
See: https://sagemaker.readthedocs.io/en/stable/v2.html
'sagemaker-tensorflow-serving-2024-04-06-18-33-45-013'
```

Figure 6.4: Endpoint name

We can see the endpoint instance have been created in the AWS console as well.

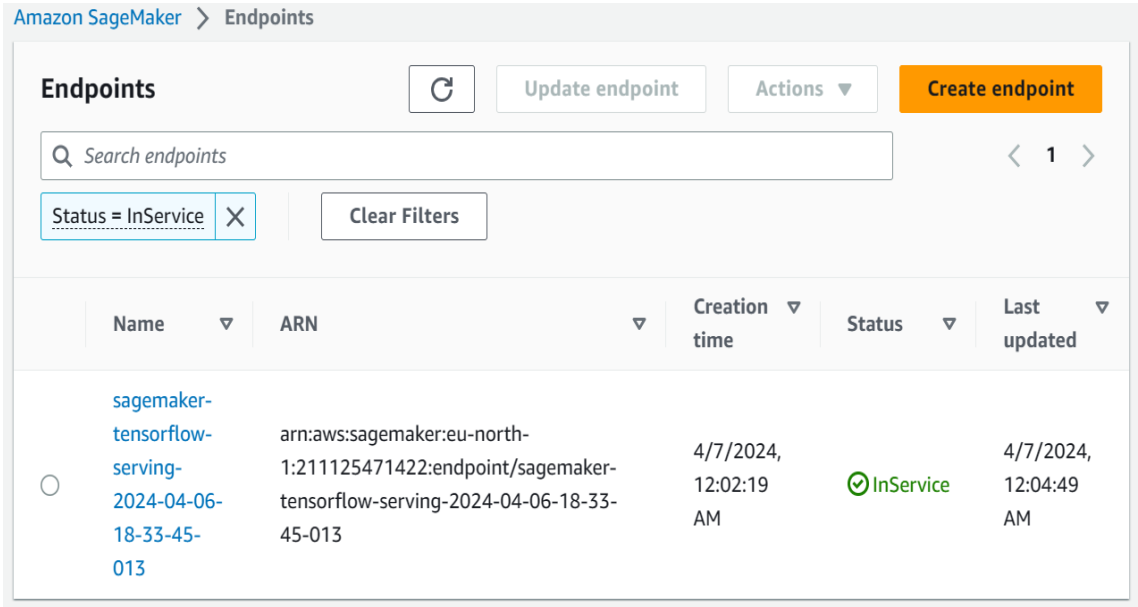
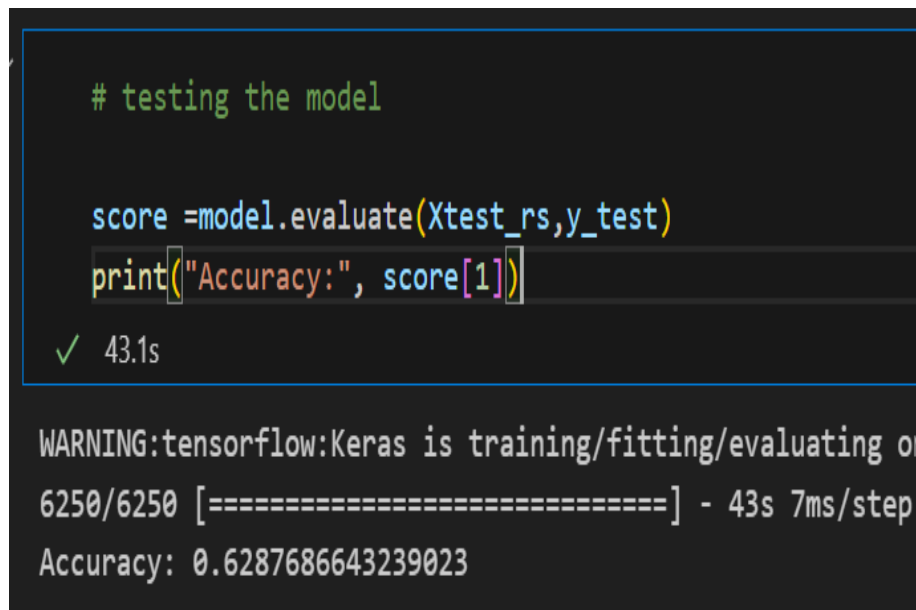


Figure 6.5: Endpoint creation

6.2 Quantitative Results

The `model.evaluate()` method provided by TensorFlow Keras is used to assess the performance of the model. The function calculates the model's performance metrics by using the test data (`Xtest_rs` and `y_test`) as input. After that, the model's accuracy is taken out of the calculated scores (`score[1]`) and displayed on the console. This stage helps evaluate

the trained model's suitability for practical applications by offering insightful information about how well it generalizes to new data.



```
# testing the model

score = model.evaluate(Xtest_rs, y_test)
print("Accuracy:", score[1])

✓ 43.1s

WARNING:tensorflow:Keras is training/fitting/evaluating on
6250/6250 [=====] - 43s 7ms/step
Accuracy: 0.6287686643239023
```

Figure 6.6: Model accuracy

The LIEF library version required for the feature extraction process in the earlier module was 0.9.0. But that version got outdated and there is no build wheel available for that version currently. All the ember inbuilt features for feature extraction and data processing are based on this version of LIEF. In this project, a newer version of it is used. Hence there are slight inconsistencies in the whole result. Also the version incompatibility of python libraries with LIEF, Ember, Boto3 and other required libraries also triggered unexpected warnings and it has also affected the connection to the AWS platform.

Chapter 7

Conclusions & Future Scope

This project underscores practical competence in deploying machine learning models for real-time malware classification, specifically targeting Portable Executable (PE) files. The ability to promptly categorize these files as benign or malicious carries wide-ranging implications, spanning threat detection, network security, and the protection of sensitive information. Leveraging the synergy of machine learning and cloud computing, our tool represents a potent solution to enhance security measures. Through the training of machine learning models on relevant datasets and their deployment on a scalable cloud environment, we aim to showcase the practical application of advanced technologies in addressing contemporary cybersecurity challenges.

By integrating machine learning and cloud computing, our approach not only demonstrates the versatility of these technologies but also offers a tangible solution to combat malware threats efficiently. The project's focus on practical implementation highlights its relevance in providing effective, scalable, and prompt responses to potential security risks in dynamic digital environments. This initiative contributes to the ongoing evolution of cybersecurity practices, emphasizing the crucial role of machine learning and cloud computing in fortifying defenses against ever-evolving cyber threats.

The project's future scope involves advancing machine learning with deep and reinforcement learning for more accurate malware classification. Integration with dynamic threat intelligence feeds ensures real-time adaptability to emerging threats. Behavioral analysis will complement static file analysis, offering insights into runtime behaviors. Adopting containerization enhances scalability, and user-friendly interfaces empower end-users with threat information. Extending platform support beyond PE files ensures a comprehensive solution for diverse cybersecurity challenges.

References

- [1] S. Lad and A. Adamuthe, “Improved deep learning model for static pe files malware detection and classification,” *International Journal of Computer Network and Information Security*, vol. 14, pp. 14–26, 04 2022.
- [2] H. Sun, X. Wang, R. Buyya, and J. Su, “Cloudeyes: Cloud-based malware detection with reversible sketch for resource-constrained internet of things(iot) devices,” *Software Practice and Experience*, vol. 47, 05 2016.
- [3] M. Abdelsalam, R. Krishnan, Y. Huang, and R. Sandhu, “Malware detection in cloud infrastructures using convolutional neural networks,” 07 2018, pp. 162–169.
- [4] R. Yadav, “Effective analysis of malware detection in cloud computing,” *Computers Security*, vol. 83, 12 2018.
- [5] R. Kumar, K. Sethi, N. Prajapati, R. Rout, and P. Bera, “Machine learning based malware detection in cloud environment using clustering approach,” 07 2020, pp. 1–7.
- [6] C. Connors and D. Sarkar, “Machine learning for detecting malware in pe files,” 12 2022.
- [7] Q. Trinh, “1.55m api import dataset for malware analysis,” 2021. [Online]. Available: <https://dx.doi.org/10.21227/98jc-y909>
- [8] R. Thomas, “Lief - library to instrument executable formats,” <https://lief.quarkslab.com/>, apr 2017.
- [9] e. Ember, “Elastic malware benchmark for empowering researchers,” <https://github.com/endgameinc/ember>, 2022.

Appendix A: Presentation

Cloud Based PE Malware Detection

Final Presentation

Project Guide: Ms. Jomina John

Group 2:
Nandagovind P
Shafc Sulthana
Sheethal Mariya Binoy

April 30, 2024

Contents

- 1 Problem Definition
- 2 Project Objective
- 3 Novelty of Idea and Scope of Implementation
- 4 Literature Review
- 5 Methodology
- 6 Sequence Diagram
- 7 Architecture Diagram
- 8 Results
- 9 Task Distribution
- 10 Conclusion
- 11 Future Scope
- 12 References
- 13 Status of Paper Publication

Problem Definition

- Malware can enter our system through various means such as infected data files, insecure downloads, third party software, insecure APIs, shared resources etc.
- It can be used to steal data from business, disrupt their operations and cause a lot of problems.

Project Objective

- To build and train a deep neural network model by extracting features from an open source dataset EMBER.
- To depoly the model into a suitable cloud platform (Sagemaker).
- To classify the Portable executable files as malware or benign using the model endpoint.

Novelty of Idea and Scope of Implementation

- Integration of ML with cloud deployment & client-side implementation.
- Use of EMBER dataset for robust model training.
- Provision of Python script for local PE file classification.
- Offers a real-time prediction for malware classification.
- Emphasis on practical deployment and usability, setting it apart from existing projects.

Literature survey

PAPER	METHODOLOGY	ADVANTAGE / DISADVANTAGE
[1]	2D CNN combining with 3D CNN structure	<ul style="list-style-type: none"> • Realistic data collection • Considered as "black box" model hence lack of explainability
[2]	Unified WFCM-AANN system	<ul style="list-style-type: none"> • High detection sensitivity • Complex, Resource intensive

Literature survey

PAPER	METHODOLOGY	ADVANTAGE / DISADVANTAGE
[3]	CloudEyes: suspect bucket cross filtering; a signature detection mechanism	<ul style="list-style-type: none"> • Reduced time consumption • Not effective at detecting more complex malware threats
[4]	Based on the clustering and trend micro-locality sensitive hashing	<ul style="list-style-type: none"> • Identify and group similar malware samples • Large dataset

Methodology

- Downloaded Ember dataset and extracted features
- Vectorization of extracted features and storing it into variables
- Serialization of dataset using h5py library
- Scaling of data using Scikit Learn's Robust scalar
- Designed a sequential model with input layer, hidden layers and activation function
- Model compiled using Adam optimizer and saved the model

- Training of the model using the scaled data
- Training on 480000 samples and validation on 120000 samples for 30 epochs with batch size 256
- Obtained the accuracy of the model using model.evaluate() function
- Saved the model and model weights as json files
- Defined a testPE() function to test the model for an external PE file, model predicted the file as benign
- AWS account setup
- Creating an endpoint for the model using AWS Sagemaker

- Create python code that loads a PE file
- Converts it to a feature vector that is compatible with the model
- Run the feature vector on cloud API
- Obtain results predicted by the model
- Print results

Sequence Diagram

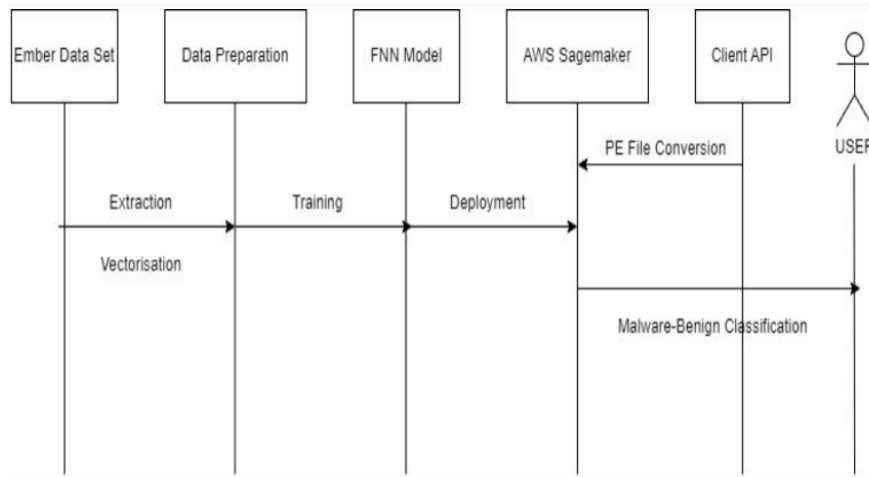


Figure: Sequence Diagram

Architecture Diagram

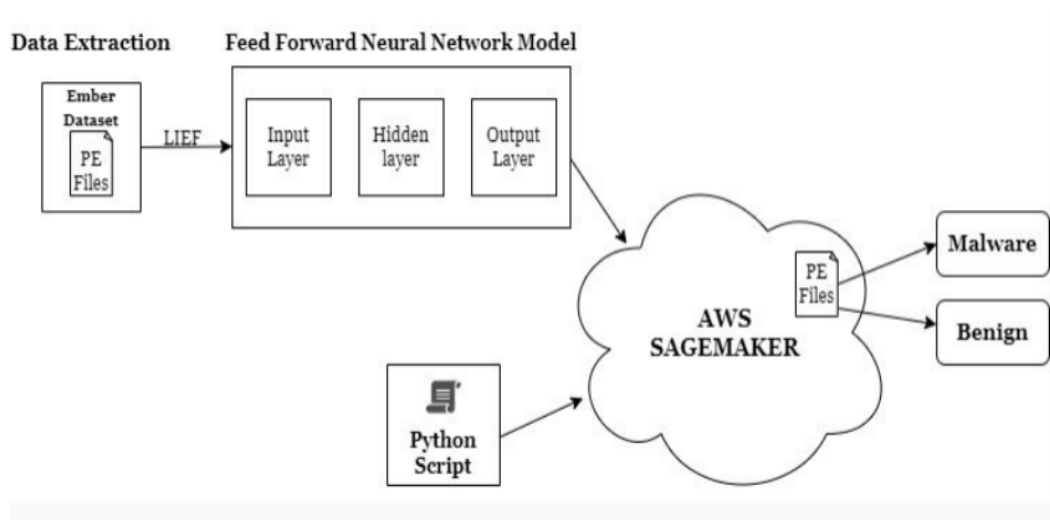


Figure: Architecture Diagram

Results

```
# testing the model

score = model.evaluate(Xtest_rs,y_test)
print("Accuracy:", score[1])

✓ 43.1s

WARNING:tensorflow:Keras is training/fitting/evaluating on
6250/6250 [=====] - 43s 7ms/step
Accuracy: 0.6287686643239023
```

Figure: Model Accuracy

```
testPE("Anaconda3-2020.02-Windows-x86_64.exe")
✓ 44.1s

WARNING: EMBER feature version 2 were computed using
WARNING: lief version 0.13.2-2d9855fc found instead
WARNING: in the feature calculations.
1/1 [=====] - 0s 183ms/step
'benign'

testPE("Git-2.43.0-64-bit.exe")
✓ 7.9s

WARNING: EMBER feature version 2 were computed using
WARNING: lief version 0.13.2-2d9855fc found instead
WARNING: in the feature calculations.
1/1 [=====] - 0s 416ms/step
'benign'
```

Figure: Model Testing

```
%%time
predictor = sagemaker_model.deploy(initial_instance_count=1,
                                   instance_type='ml.m5.large')

----|CPU times: total: 3.75 s
Wall time: 4min 24s
```

Figure: Endpoint Creation

```
predictor.endpoint

WARNING:sagemaker.deprecations:The endpoint attribute has been deprecated.
See: https://sagemaker.readthedocs.io/en/stable/v2.html
'sagemaker-tensorflow-serving-2024-04-06-18-33-45-013'
```

Figure: Endpoint Name

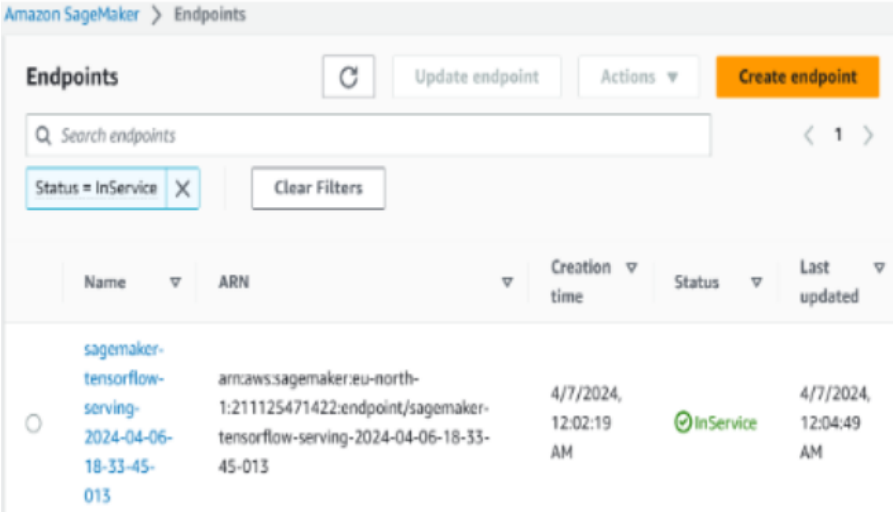


Figure: Endpoint Creation

Task Distribution

NANDAGOVIND P	SHAFIC SULTHANA	SHEETHAL MARIYA
-EMBER DATA SET EXTRACTION -PREPROCESSING -SCALING -MODEL EVALUATION AND SAVING -DOCUMENTATION	-VECTORIZATION -AWS SETUP -END POINT CREATION -DOCUMENTATION	-MODEL ARCHITECTURE DESIGN -MODEL TRAINING AND TESTING -DOCUMENTATION

Conclusion

- Demonstrate our practical skills in implementing and deploying machine learning models for malware classification.
- The ability to classify PE files as benign or malicious in real time has a broad application, including threat detection, network security and protecting sensitive information.

Future Scope

- Develop a real-time malware detection system that continuously monitors incoming files and quickly responds to potential threats. Integrate with existing security infrastructure to automatically quarantine or mitigate identified malware.
- Incorporate feedback loops to iteratively improve the model over time based on real-world feedback and evolving malware threats.

References

- Connors, C., & Sarkar, D. (2022). Machine Learning for Detecting Malware in PE Files. arXiv preprint arXiv:2212.13988.
- Lad, Sumit & Adamuthe, Amol. (2022). Improved Deep Learning Model for Static PE Files Malware Detection and Classification. International Journal of Computer Network and Information Security. 14. 14-26. 10.5815/ijcnis.2022.02.02.
- LIEF - Library to Instrument Executable Formats. 2020. Lief.Quarkslab.Com. <https://lief.quarkslab.com/download>. [Accessed 14 August 2020]
- Quynh Trinh. (2021). 1.55M API IMPORT DATASET for MALWARE ANALYSIS. IEEE Dataport. <https://dx.doi.org/10.21227/98jc-y909>

References

- Abdelsalam, Mahmoud & Krishnan, Ram & Huang, Yufei & Sandhu, Ravi. (2018). Malware Detection in Cloud Infrastructures Using Convolutional Neural Networks. 162-169. 10.1109/CLOUD.2018.00028.
- (2018). Effective Analysis Of Malware Detection In Cloud Computing. Computers & Security. 83.10.1016/j.cose.2018.12.005.
- Sun, Hao & Wang, Xiaofeng & Buyya, Rajkumar & Su, Jinshu. (2016). CloudEyes: Cloud-based Malware Detection with Reversible Sketch for Resource-constrained Internet of Things(IoT) Devices. Software Practice and Experience. 47. 10.1002/spe.2420.
- Kumar, Rahul & Sethi, Kamalakanta & Prajapati, Nishant & Rout, Rashmi & Bera, Padmalochan. (2020). Machine Learning based Malware Detection in Cloud Environment using Clustering Approach. 1-7. 10.1109/ICCCNT49239.2020.9225627.

Paper Publication Status

- Conference: International Conference on Engineering & Technology (ICET-24)
- Conference Date: 20th April 2024
- Venue: Bangalore, India
- Status: Paper presented, Publication results soon to be received
- Paper ID: NIER_10832
- Paper Publisher: National Institute For Engineering and Research (NIER)

