SECURING ANDROID DEVICES: A ROBUST APPROACH TO AUTOMATED MALWARE DETECTION THROUGH ENSEMBLE LEARNING

A PROJECT REPORT

Submitted by

JOHN PRAVEEN J [211420104113]

KIRAN M P [211420104132]

PRABAAKARAN C M [211420104196]

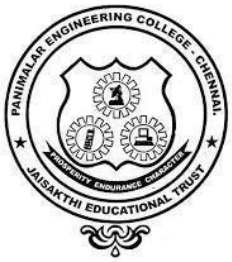
in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING



PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

APRIL 2024

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report **"SECURING ANDROID DEVICES: A ROBUST APPROACH TO AUTOMATED MALWARE DETECTION THROUGH ENSEMBLE LEARNING"** is the bonafide work of **JOHN PRAVEEN J [211420104113], KIRAN M P [211420104132]**, **PRABAAKARAN C M [211420104196]** who carried out the project work under my supervision.

SIGNATURE
Dr L.JABASHEELA M.E., Ph.D.,
HEAD OF THE DEPARTMENT,

Department of Computer Science and Engineering
Panimalar Engineering College,
Chennai - 123

SIGNATURE
Mr A.N SASI KUMAR M.E.,
ASSISTANT PROFESSOR(Grade I),
SUPERVISOR

Department of Computer Science and Engineering,
Panimalar Engineering College,
Chennai - 123

Certified that the above candidate(s) were examined in the End Semester Project Viva-VoceExamination held on .....

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION BY THE STUDENT

We **JOHN PRAVEEN J [211420104113] , KIRAN M P [211420104132] PRABAAKARAN C M [211420104196] hereby declare that this project report titled "SECURING ANDROID DEVICES: A ROBUST APPROACH TO AUTOMATED MALWARE DETECTION THROUGH ENSEMBLE LEARNING " , under the** guidance of **Mr.A.N.SASI KUMAR** is the original work done by us and we have not plagiarized or submitted to any other degree in any university by us.

JOHN PRAVEEN J

KIRAN M P

PRABAAKARAN C M

ACKNOWLEDGEMENT

Our profound gratitude is directed towards our esteemed Secretary and Correspondent, **Dr. P. CHINNADURAI, M.A., Ph.D**., for his fervent encouragement. His inspirational support proved instrumental in galvanizing our efforts, ultimately contributing significantly to the successful completion of this project

We wish to express our deep gratitude to our Directors, **Tmt . C.VIJAYARAJESWARI**, **Dr. C. SAKTHIKUMAR, M.E., Ph.D., and Dr.SARANYA SREE SAKTHIKUMAR, B.E., M.B.A., Ph.D.,** for graciously affording us the essential resources and facilities for undertaking of this project.

We also express our gratitude to our Principal, **Dr. K. MANI, M.E., Ph.D.,** who facilitated us in completion the project.

We express our heartfelt thanks to **Dr. L. JABASHEELA, M.E., Ph.D.,** Head of the Department of Computer Science and Engineering, for granting the necessary facilities that contributed to the timely and successful completion of project.

We would like to express our sincere thanks to **Project Coordinator Dr.G Senthil Kumar M.C.A.,M.Phil.,M.B.A.,M.E.,Ph.D.,** and **Project Guide Mr.A.N Sasi Kumar,M.E** and all the faculty members of the Department of CSE for their unwavering support for the successful completion of the project.

JOHN PRAVEEN J
KIRAN M P
PRABAAKARAN C M

PROJECT COMPLETION CERTIFICATE

Pantech e Learning
DIGITAL LEARNING SIMPLIFIED

CIN: U80902TN2021PTC141464

www.pantechelearning.com

21/03/2024

COMPLETION CERTIFICATE

This is to acknowledge that students from **"PANIMALAR ENGINEERING COLLEGE"** has completed **Project** on the **title** of **"ANDROID MALWARE DETECTION"** at our concern from **DEC 2023** to **MAR 2024.**

1. **JOHN PRAVEEN J**
2. **KIRAN M P**
3. **PRABAAKARAN C M**

For Pantech e learning.,

Authorized Signatory

Pantech eLearning Pvt Ltd.,
II Floor, Kotta Srinivasiah Charities Building,
Thanjavur Street, Near Duraisamy Subway, T.Nagar, Chennai – 600017
Phone: 91 44 42606470 | hr@pantechmail.com

TABLE OF CONTENTS

ABSTRACT

In this work, The focus on the malicious call prevention problem without relying on any particular underlying telephony network infrastructure. Thus the first challenge is how to gather effective information. The first contribution of this work is to collect information about malicious callers in order to build an effective prevention mechanism, using Machine Learning Algorithms. We are collecting the dataset based on consumer data. In that, we are having three types of classes like telemarketing, Unwanted(Malicious), rob calls.

With the exponential growth of mobile applications on the Android platform, the threat landscape for malware has become increasingly complex and sophisticated. This project aims to enhance the security of Android devices by developing an advanced malware detection system using machine learning techniques.

The proposed solution leverages the power of machine learning algorithms to analyze and classify Android applications based on their behavior and characteristics. Features such as app permissions, system calls, API usage patterns, and code anomalies will be extracted and processed to train a robust machine learning model. The model will be trained on a diverse dataset containing both benign and malicious Android applications to ensure its effectiveness in real-world scenarios.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

KNN Algorithm – K-Nearest Neighbour Algorithm

SVM Algorithm – Support Vector Machine

RF Classifier – Random Forest Classifier

CHAPTER 1
INTRODUCTION

CHAPTER 1

INTRODUCTION

As mobile devices continue to play an integral role in our daily lives, safeguarding the Android ecosystem becomes paramount. This project endeavors to provide users with a proactive defense against evolving malware threats, ultimately fostering a more secure and resilient environment for Android users globally. In the realm of cybersecurity, the proliferation of Android malware presents a formidable challenge, necessitating innovative solutions. Leveraging machine learning techniques offers a promising avenue to detect and mitigate these threats effectively. By analyzing patterns and anomalies within app behavior, machine learning algorithms can discern malicious intent. Feature engineering plays a crucial role in extracting relevant attributes from Android applications for model training. Supervised learning methods like Support Vector Machines and Neural Networks are commonly employed for malware detection. Unsupervised learning techniques, including clustering, aid in identifying novel threats without labeled data. Transfer learning and ensemble methods further enhance model robustness and adaptability. Performance evaluation metrics gauge the effectiveness and efficiency of detection systems. Despite progress, challenges persist, signaling the need for ongoing research and development to fortify Android security. This convergence of Android security and machine learning promises to bolster defenses against evolving cyber threats in the mobile ecosystem.

Key Objectives:

- Understanding Android Malware Diversity: We delve into the multifaceted nature of Android malware, examining the various types and strategies employed by attackers.

- Feature Extraction and Selection: Machine learning models rely heavily on the features extracted from datasets. We explore the identification and extraction of relevant features from Android applications, considering factors such as code behavior, permissions, network activities, and anomalies.

- Model Training and Evaluation: The paper delves into the intricacies of training machine learning models for Android malware detection. We discuss the selection of appropriate algorithms, the creation of robust datasets, and the evaluation metrics used to assess the model's effectiveness in real-world scenarios.

- Dynamic Analysis and Behavioral Modeling: Recognizing the limitations of static analysis, we explore the integration of dynamic analysis and behavioral modeling into the machine learning framework. This approach allows for the detection of malware based on real-time behavior, enhancing the model's adaptability to evolving threats.

- Real-world Implementation and Challenges: We discuss the practical challenges associated with implementing machine learning-based malware detection on Android devices, considering factors such as resource constraints, false positives, and the need for continuous model updates to combat emerging threats.

By amalgamating the power of machine learning with the intricacies of Android security, this research aims to contribute to the ongoing efforts to create a robust and adaptive defense mechanism against the ever-evolving landscape of Android malware.

Traditional methods of malware detection have struggled to keep pace with the sophistication and diversity of modern Android malware. As malware continues to evolve, adopting innovative and adaptive approaches becomes imperative for effective defense. In this context, machine learning emerges as a promising solution, leveraging the power of artificial intelligence to detect and combat Android malware with greater accuracy and efficiency.

However, this widespread adoption has also attracted the attention of malicious actors seeking to exploitvulnerabilities for various nefarious purposes. Android malware poses a significant threat to user privacy, data security, and overall device functionality.By harnessing the capabilities of machine learning, we aim to enhance the resilience of Android devices against the dynamic landscape of malicious software.

1.1.PROBLEM DEFINITION

The objective of this study is to develop a prevention mechanism for malicious calls without relying on specific telephony network infrastructure. It aims to collect effective information through Machine Learning Algorithms, utilizing consumer data to classify calls into telemarketing, Unwanted (Malicious), and rob calls.The escalating prevalence of Android malware presents a formidable challenge to the security and integrity of mobile devices. Conventional methods of malware detection, primarily reliant on static signatures and heuristics, are proving inadequate in effectively combating the dynamic and sophisticated nature of modern malware threats. As a result, there is an urgent need for a more adaptive and intelligent solution to detect and mitigate Android malware.

- Dynamic Nature of Malware
- Zero-Day Threats
- False Positives and Negatives
- Feature Richness and Context
- Resource Efficiency
- Adaptability to Emerging Threats
- User Awareness and Interaction
- The current methods of Android malware detection are not robust enough to effectively identifyand mitigate emerging threats.
- There is a need for an intelligent and adaptive system that can proactively detect Android malwareusing machine learning techniques.
- Dynamic nature of Android malware requires continuous updating of detection model.

CHAPTER 2

LITERATURE SURVEY

CHAPTER 2
LITERATURE SURVEY

S.no	Year	Title	Author	Technics	Disadvantages
1	2019	Title: Software defectprediction techniques using metrics based on neuralnetwork classifiers. Cluster Computing, 22(1), pp.77-88.	Jayanthi, R. and Florence, L.	Software industriesstrive forsoftwarequality improvement by consistent bug prediction, bug removal andprediction of fault-prone module.	Not possible in multiple stages
2	2017	Integrated approach to software defect prediction. IEEE Access, 5, pp.21524-21547.	Felix, E.A. and Lee, S.P	Software defect predictionprovides actionable outputs to software teams while contributing to industrial success.	Needs high resources
3	2015	Multiple kernel ensemble learning for software defect prediction. Autom. Softw. Eng. 23, 569–590	Wang, T., Zhang, Z., Jing, X., Zhang, L.	Software defect prediction aims to predict the defect proneness of new with the historical defect data so as to improve the quality of a software system.	Needs a high end processing platform

4	2016	IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Suita, pp. 370–381	Xu, Z., Xuan, J., Liu, J., Cui, X.: MICHAC.	Defect prediction aims to estimate software reliabil-ity via learning from historical defect data.	It is very limited because it is unable to handle a number of large existing data,
5	2019	Effective multi-objective naïve Bayes learning for cross-project defect prediction. Appl. Soft Comput. 49, 1062	Ryu, D., Baik, J	A piece of maximum information, datacorrelation-based technique is proposed to tackle this problem.	Require high memory for processing

CHAPTER 3
THEORETICAL BACKGROUND

CHAPTER 3

THEORETICAL BACKGROUND

3.1 IMPLEMENTATION ENVIRONMENT

Hardware:

Processor	: Intel i3
Hard Disk	: 500 GB
RAM	: 2 GB
Operating System	: Windows7 or above

Software:

Python IDLE

Anaconda Jupyter

Visual Studio Code,

Command PromptIDE**:**

Anaconda Navigator

Tool**:** Jupiter Notebook

DatasetPython Version:

Python 3

Dependencies

Streamlit =1.10.0

Pandas =1.1.3

Numpy =1.19.2

scikit-learn =0.23.2

python-dotenv =0.14.0,

pickle

3.2 SYSTEM ARCHITECTURE

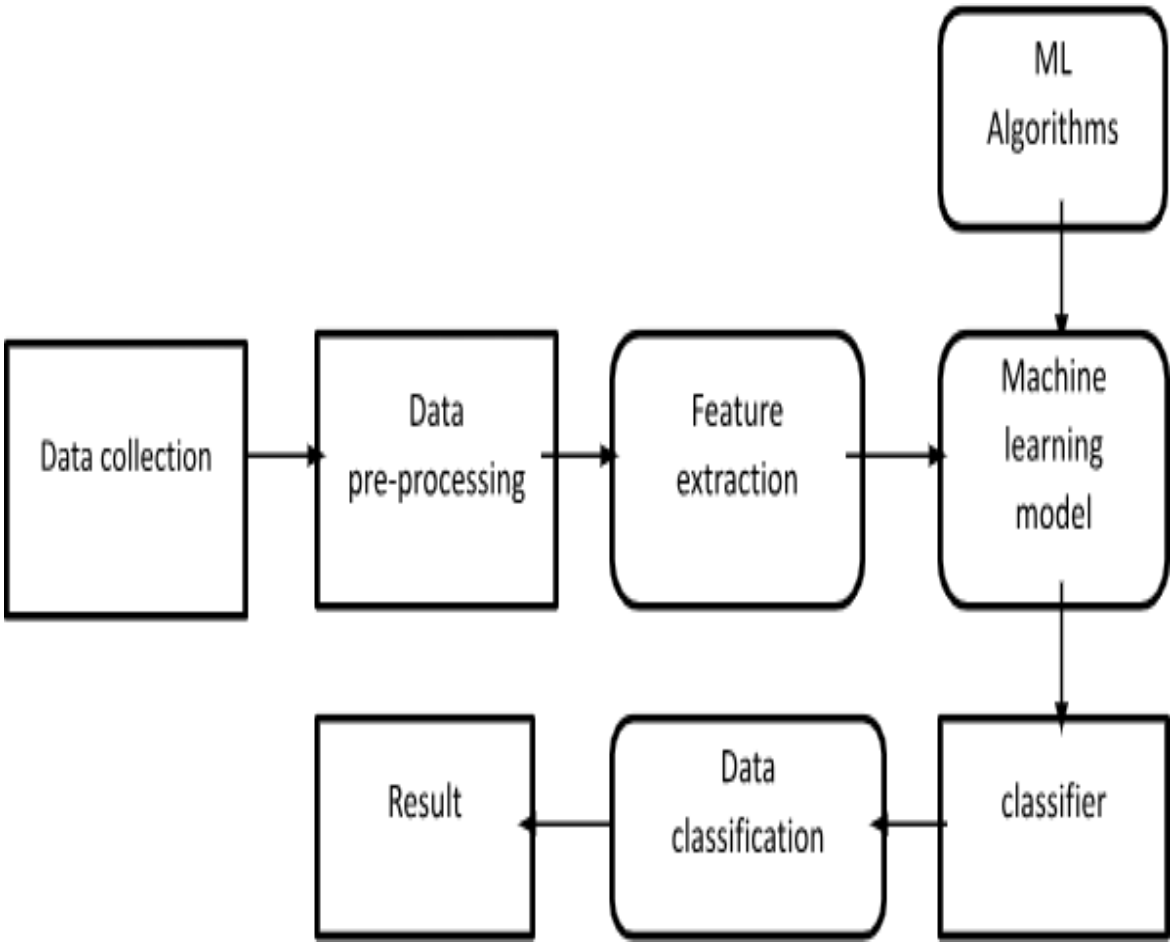


Fig 3.2.1 System Architecture

3.3.PROPOSED METHODOLOGY

First, we are preprocessing the data and splitting as a two part. One is a training and test part. Then what are the features we need from dataset, we are selecting that attributes.Then we are applying the classification technique using machine learning Boosting algorithm.At last we are finding the result as accuracy and confusion matrix results. Then getting AoC and RoC accuracy.When implementing some of the new approaches for suggesting new methodologies, the stagesthat follow are as follows,DataCollection,DataPreprocessing,FeatureExtraction,ModelSelection,Trainin g the Model,Evaluation,Integration into Real-time System,Continuous Monitoring and Updates

Incorporate dynamic analysis by executing apps in a controlled environment (sandbox) to observe runtime behavior, detect anomalies, and improve the model's accuracy.

Deployment:

- Integrate the trained model into an Android security solution or antivirus application.
- Regularly update the model with new data to adapt to evolving malware threats.Continuous Improvement:
- Periodically review and update the methodology, incorporating new features, algorithms, and techniques to improve detection capabilities.

Remember, the effectiveness of the proposed methodology will depend on the quality and diversity of the dataset, the chosen machine learning model, and the adaptability of the system to emerging malware threats. Regular updates and improvements are crucial to maintaining a robust and effective malware detection system.

3.3.1 DATABASE DESIGN/DATA SET DESCRIPTION

Android malware detection using machine learning involves the use of various algorithms and techniques to identify malicious software specifically designed for the Android operating system. The goal is to develop a robust and accurate system that can automatically classify apps as either malicious or benign based on their features and behavior.

Dataset:

Creating a dataset for Android malware detection involves collecting a diverse set of Android applications, both malicious and benign, to train and evaluate the machine learning model. Here are key components of a dataset for Android malware detection:

Malicious Samples:

Obtain samples of known Android malware from sources like malware repositories, securityresearch reports, or threat intelligence feeds.

Ensure a diverse set of malware types, such as spyware, ransomware, adware, and trojans, to capture thevariety of threats.

Benign Samples:

- Collect a representative set of benign Android applications from official app stores likeGoogle Play.
- Ensure the benign samples cover a wide range of app categories and functionalities tomimic real-world usage.

Feature Extraction:

- Extract relevant features from the apps for training the machine learning model. These featurescan include:
- Permissions requested by the app.
- API calls made by the app.

- Manifest file information.
- Code analysis results.
- Network behavior.
- Resource usage patterns.

Labeling:

Label each sample as either malicious or benign based on its actual nature. This labeling is crucial for supervised machine learning.

Data Splitting:

Split the dataset into training, validation, and testing sets. A common split might be70% fortraining, 15% for validation, and 15% for testing.

Preprocessing:

Perform necessary preprocessing steps, such as normalizing feature values, handling missingdata, and converting categorical data into a suitable format.

Machine Learning Models:

Train various machine learning models, such as decision trees, random forests, support vectormachines, or deep neural networks, on the training dataset.

Evaluation:

Evaluate the models on the validation set to fine-tune hyperparameters and select the best-performing model.

Testing:

Assess the final model on the testing set to measure its real-world performance in detecting Android malware.

Deployment:

Once satisfied with the model's performance, deploy it for real-time Android malware detectionin a suitable environment.

3.3.2 INPUT DESIGN

Data Collection:

Gather a diverse dataset of Android applications, including both benign and malicious samples.

This dataset should represent various categories and versions of apps.

Feature Extraction:

Extract relevant features from the collected samples. Features may include permissions requested, API calls, system calls, file operations, and other behavioral patterns. Additionally, features from static analysis, such as code structure and permissions, can be useful.

Data Preprocessing:

Clean and preprocess the extracted features. This involves handling missing data, normalizing values, and converting categorical data into numerical format. Data preprocessing is crucial for the performance of machine learning models.

Labeling:

Assign labels to the dataset indicating whether each sample is benign or malicious. Ensure a balanced distribution of classes to avoid bias in the model.

Model Selection:

Choose a suitable machine learning algorithm for your Android malware detection task. Common choices include decision trees, random forests, support vector machines (SVM), and deep learning approaches like neural networks.

Training the Model:

Split the dataset into training and testing sets. Train the machine learning model using the training set. Tweak hyperparameters to optimize performance, and evaluate the model on the testing set to ensure generalization.

Evaluation Metrics:

Use appropriate metrics to evaluate the performance of your model. Common metrics for binary classification tasks include accuracy, precision, recall, F1 score, and area under the receiver operating characteristic (ROC) curve.

Feature Importance Analysis:

Analyze the importance of different features in the model's decision-making process. This can provide insights into the characteristics that distinguish between benign and malicious apps.

Integration with Android Environment:

Develop a mechanism to integrate your trained model into the Android environment. This may involve creating an Android application or integrating the model into an existing security framework.

Real-time Monitoring:

Implement real-time monitoring of applications on Android devices. This involves continuouslyanalyzing app behavior and flagging potential threats based on the trained model.

Update Mechanism:

Implement a mechanism for regularly updating the model to adapt to emerging threats. This mayinvolve periodic retraining using new data and features.

User Feedback and Reporting:

Provide a user-friendly interface for reporting and handling potential threats. Users should be informed about detected threats and have the option to take appropriate actions.

Collaboration and Information Sharing:

Collaborate with the security community and consider sharing threat intelligence:

Information Sharing Platforms:

Contribute to or leverage platforms that share information on emerging threats, allowing yoursystem to stay updated on the latest malware signatures.

Community Involvement:

Engage with the broader security community, participate in conferences, and share insights toimprove the overall state of Android malware detection.

By addressing these aspects, you can create a more robust and effective Android malware detection system that adapts to the evolving threat landscape. Regular updates and collaboration with the security community will be critical for long-term success.

3.3.3 MODULE DESIGN

1.EVALUATING THE MODEL

The model development process includes a step called model validation. Finding a model that best represents the data and predicts how well the model will perform in the future is useful. In data science, it is not acceptable to evaluate model performance using the training data because this can quickly lead to overly optimistic and over fitted models. Hold-Out and Cross-Validation are two techniques used in data science to assess models. Both approaches use a test set (unseen by the model) to assess model performance in order to prevent overfitting.Based on its average, each categorization model's performance is estimated. The outcome will take on the form that was imagined. graph representation of data that has been categorized.

Proposed Approach Steps

1. We start by using the dataset of malware attacked data.
2. Filter the dataset in accordance with the needs, then construct a new dataset attributes thatcorrespond to the analysis to be performed.
3. Pre-process the dataset before using it.
4. Distinguish training from testing data.
5. Analyze the testing dataset using the classification algorithm after training the model with training data.
6. You will receive results as accuracy metrics at the end.

The proposed approach for Android malware detection leverages the power of machine learning to enhance the effectiveness and efficiency of identifying malicious software on mobile devices. In this innovative method, a comprehensive set of features extracted from Android applications, such as permissions requested, API calls made, and code behaviors, serves as input to a machine learning model.

DATA FLOW DIAGRAM

LEVEL 0

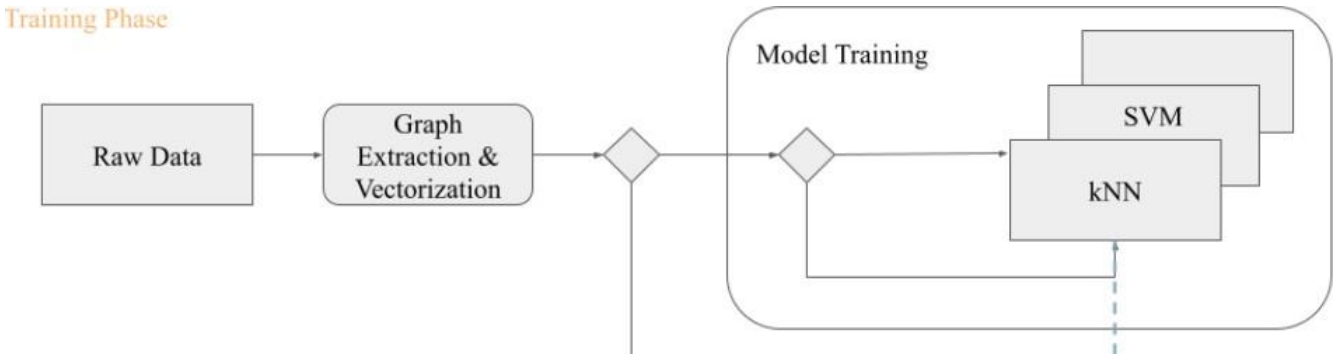


Fig 3.3.3.0 Level 0 Diagram

LEVEL 1

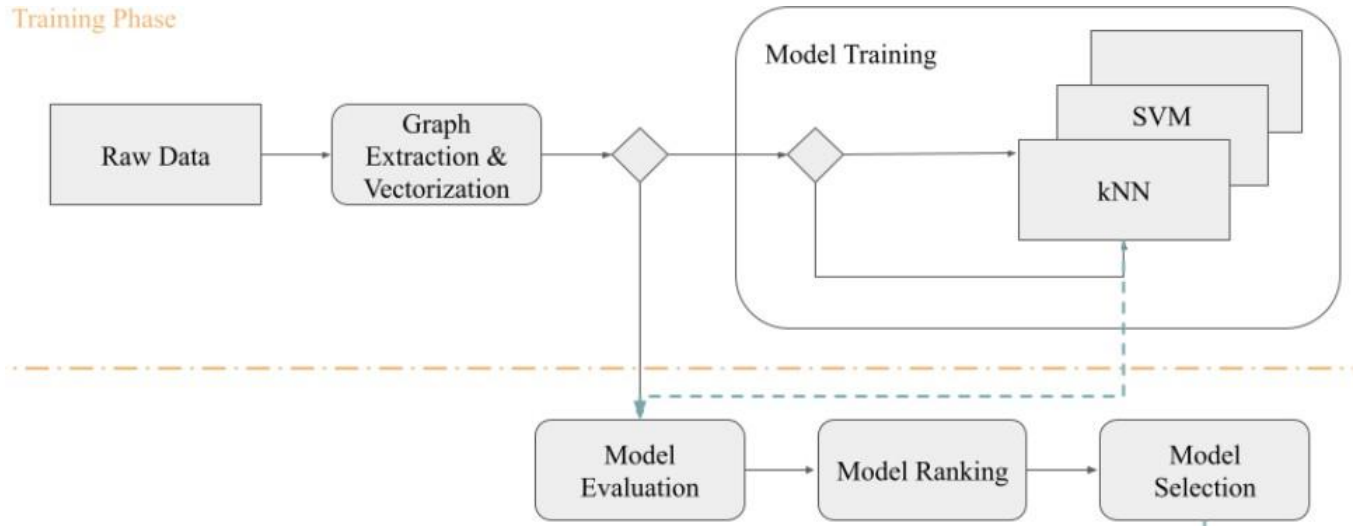


Fig 3.3.3.1 Level 1 Diagram

LEVEL 2

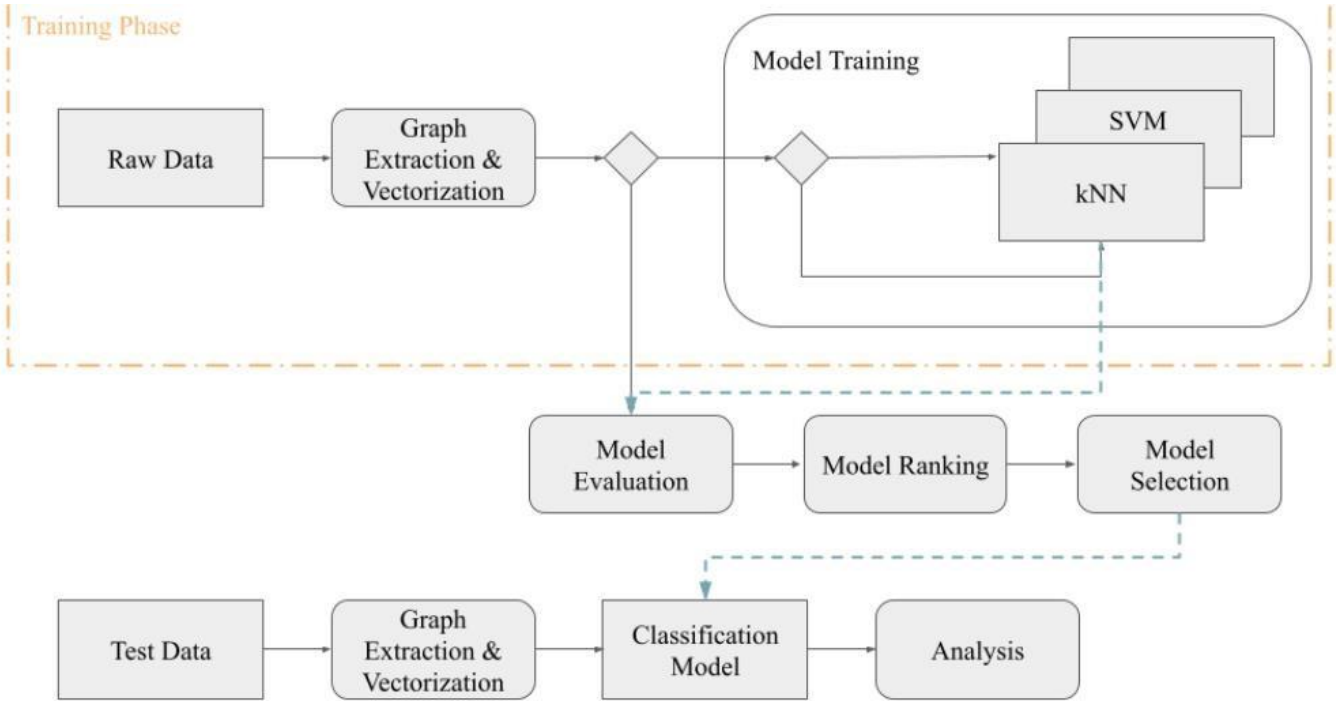


Fig:3.3.3.2 Level 2 Diagram

UML diagram

Unified Modelling Language (UML) is used to specify, visualize, modify, build, and document the artifacts of object-oriented software-intensive systems under development. UML provides a standard way to visualize a system's architectural blueprint, including elements such as:

- Actor
- Business process
- (logical) components
- Activities
- programming language statements
- Database schema and Reusable software components.

UML combines the best practices of data modeling (entity-relationship diagrams), business modeling (workflows), object modeling, and component modeling. It can be used in any process, across different implementation technologies, throughout the software development lifecycle. UML synthesized Booch's method, the Object Modeling Technique (OMT), and the Object Oriented Software Techniques (OOSE) notation by unifying them into one popular and widely used modeling language. UML aims to be a standardized modeling language that can model concurrent and distributed systems.

Sequence Diagram

Sequence diagram Represents the objects involved in an interaction horizontally and vertically in time. A use case is a type of behavior classifier that represents a declaration of provided behavior. Each use case specifies specific behavior. This may include variants that the subject can perform cooperatively with one or more of her actors. A use case defines the behavior provided by a subject without reference to the internal structure of the subject. These actions, including interactions between actors and subjects, can lead to changes in the subject's state and communication with its environment. A use case can have many variations on the basic behavior, such as anomalous behavior and error handling.

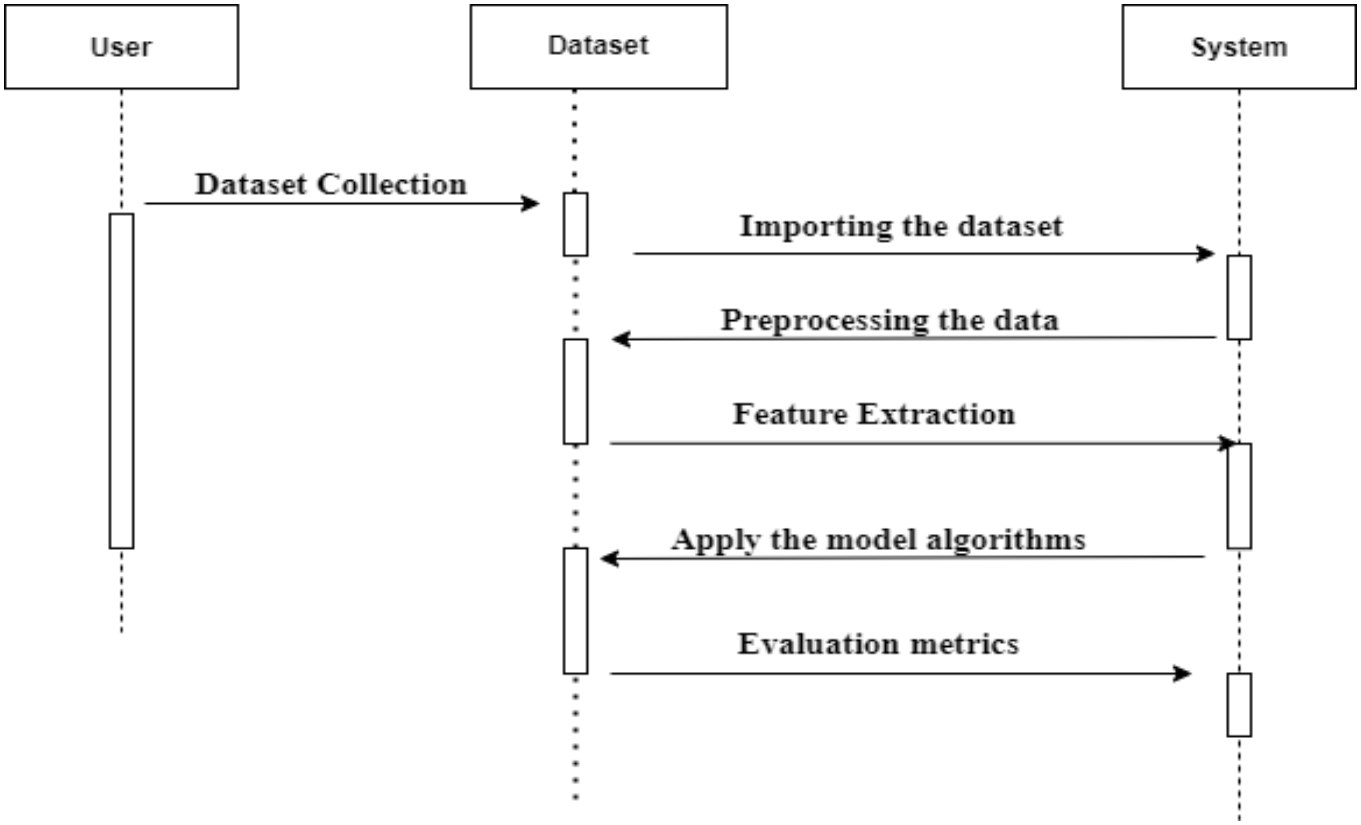


Fig:3.3.3.4 Sequence Diagram

Activity Diagram

Activity diagrams are graphical representations of workflows containing step-by-step activities and actions that support selection, repetition, and concurrency. Unified Modeling Language allows you to use the activity diagrams to step-by-step describe the business and operational workflows of the components in your system. Activity diagrams show the overall control flow of this project.

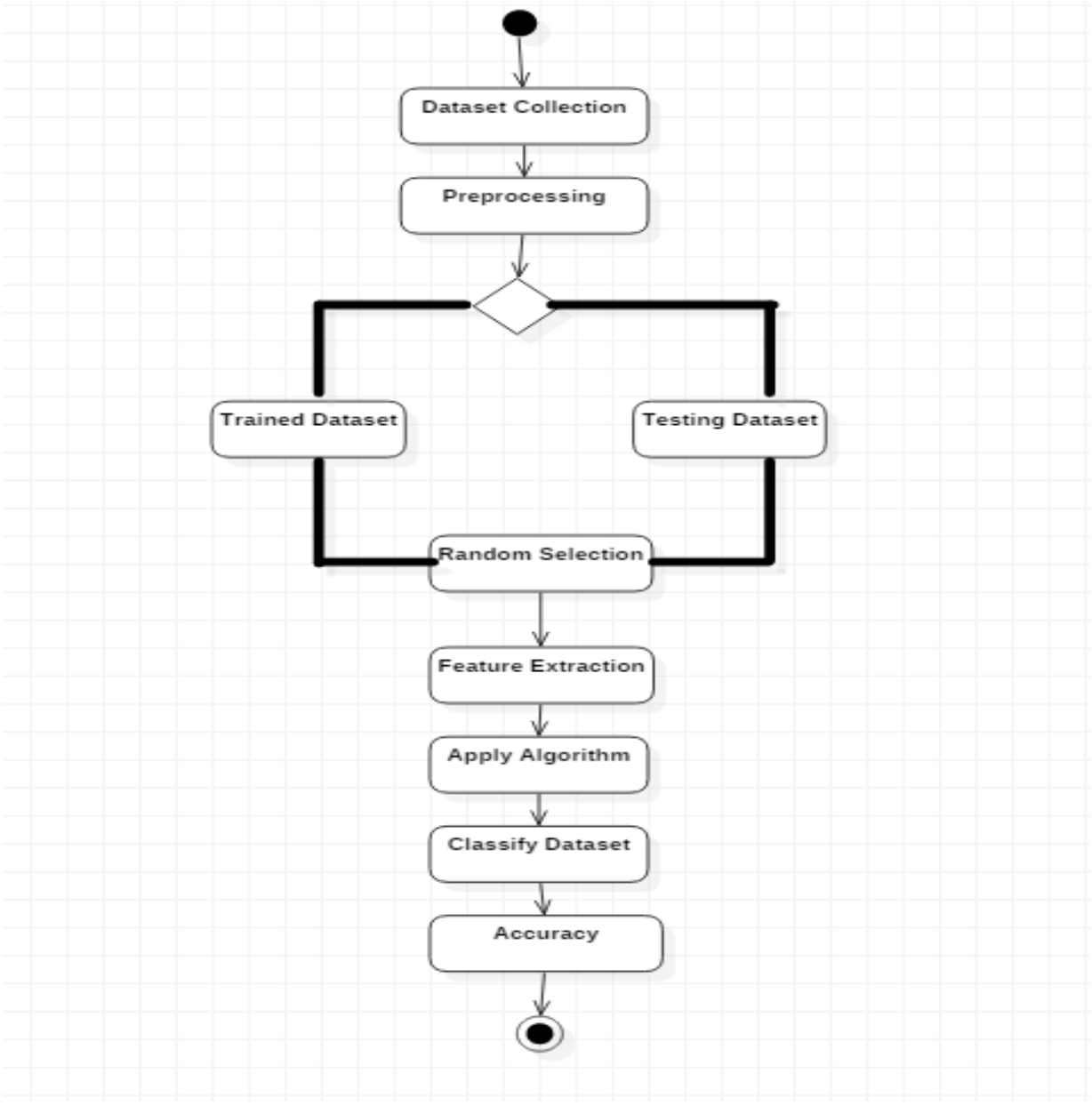


Fig:3.3.3.5 Activity Diagram

Use case diagram

• UML is a standard language for specifying, visualizing, building, and documenting the artifacts of software systems. • UML was developed by the Object Management Group (OMG) and the UML 1.0 draft specification was submitted to OMG in January 1997. • OMG is continuously committed to creating true industry standards. • UML stands for Unified Modeling Language. • UML is a visual language used to create software designs.

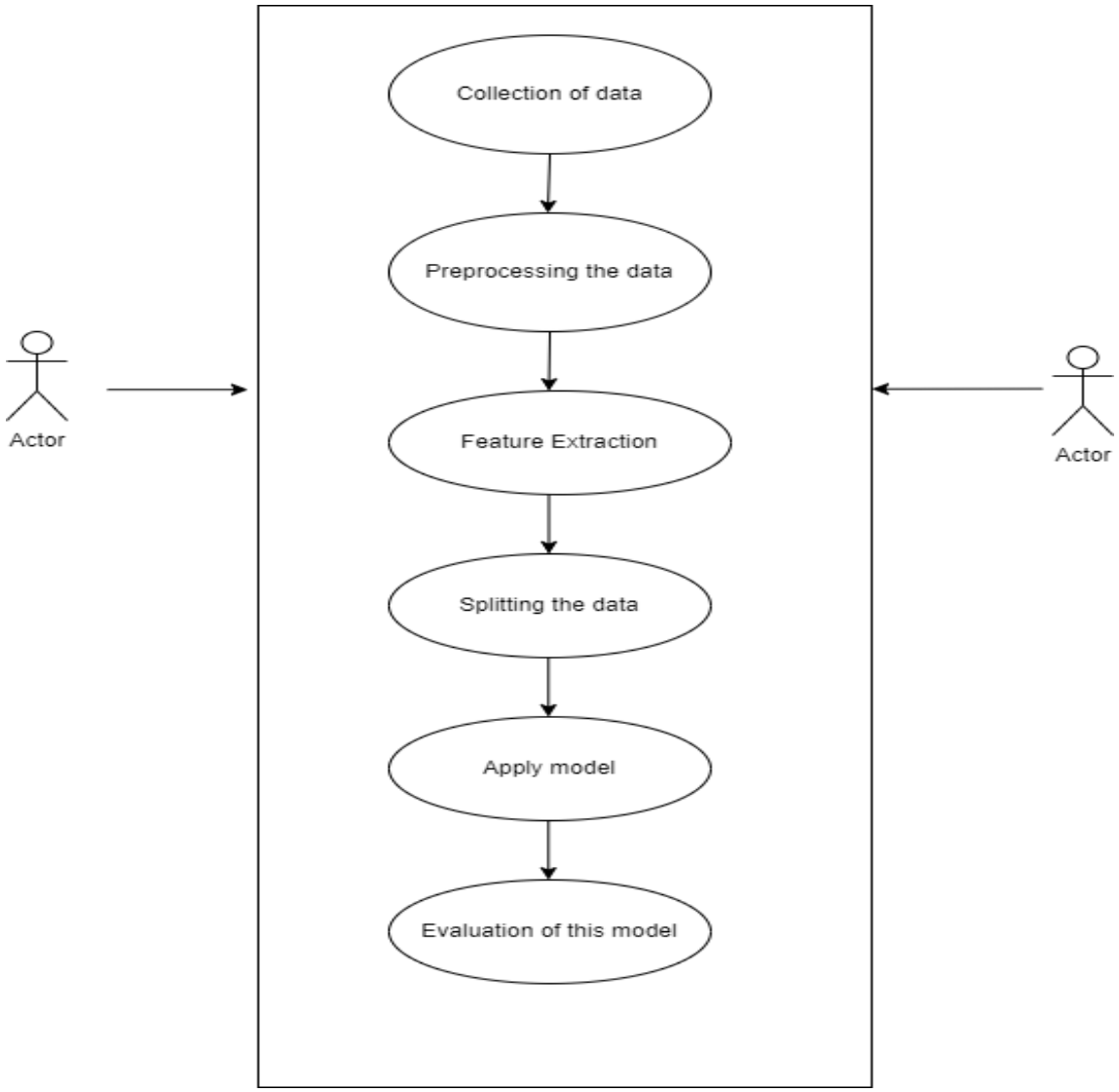


Fig:3.3.3.6 Use case Diagram

Class diagram

Class diagrams are a key building block of object-oriented modeling. It is used for general conceptual modeling of the application's system and detailed modelling by translating the model into programming code. Class diagrams can also be used for data modelling. Classes in class diagrams represent both major elements, interactions within the application, and classes to be programmed. In the diagram,a class is represented by a boxcontaining three compartments.

1. The top compartment contains the name of the class. Make it bold, cantered, and capitalize the firstletter.
2. The centre compartment consists of the magnificence attributes. They are left aligned and the primaryletter is lowercase.
3. The backside compartment consists of operations that the magnificence can perform. They also are leftaligned and the primary letter is lowercase.

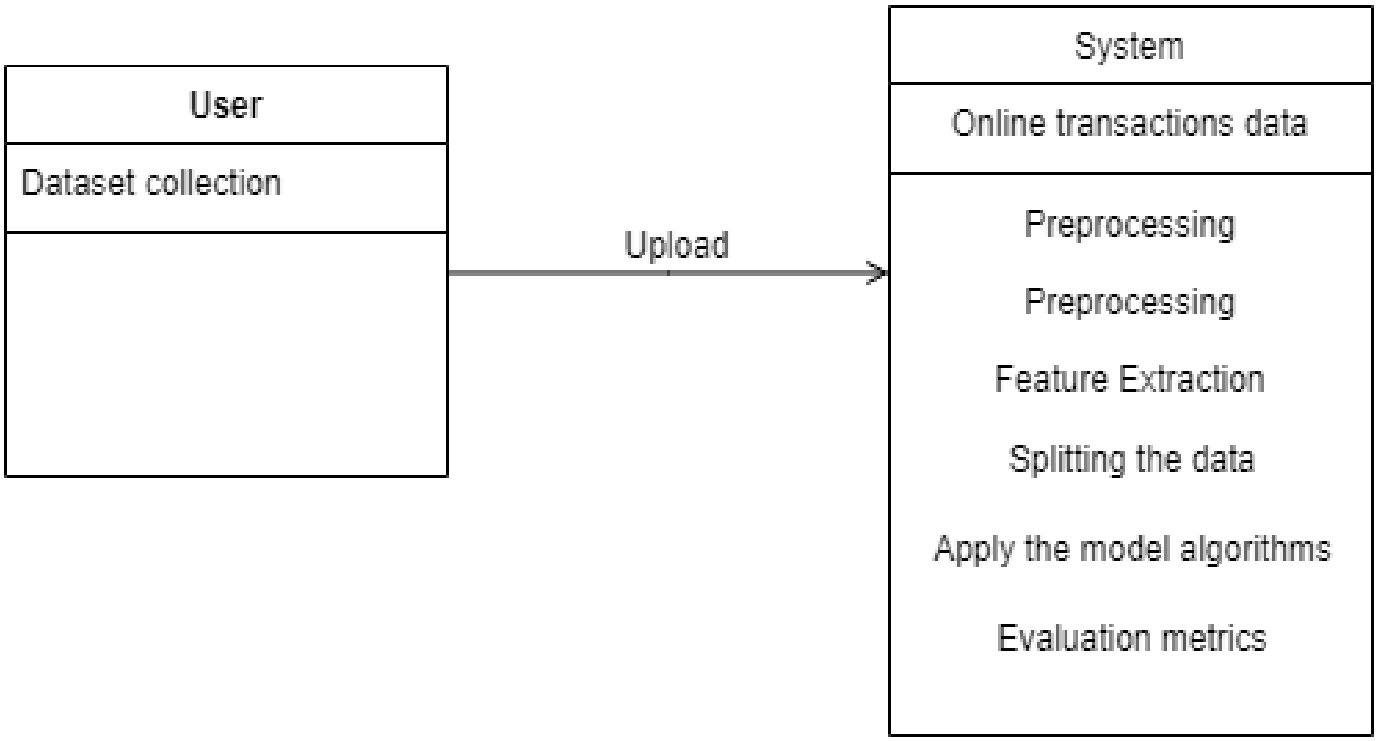


Fig:3.3.3.7 Class Diagram

CHAPTER 4
SYSTEM IMPLEMENTATION

CHAPTER 4

SYSTEM IMPLEMENTATION

4.1ALGORITHM DESCRIPTION 1

- COLLECTION OF DATA
- PRE-PROCESSING THE DATA
- EXTRACTION OF FEATURES
- EVALUATING THE MODEL

1.COLLECTION OF DATA

Data collection is a process that gathers information on malware attacked data from a variety of sources, which I utilized to create machine learning models. A set of malware attacked data with features is the type of data used in this work. The selection of the subset of all accessible data that you will be working with is the focus of this stage. Ideally, ML challenges begin with a large amount of data (examples or observations) for which you already know the desired solution. Label data isinformation for which you already know the desired outcome.

2.PRE-PROCESSING THE DATA

Format, clean, and sample from your chosen data to organise it. There are three typical steps indata pre-processing:

Formatting

It's possible that the format of the data you've chosen is not one that allows you to deal with it. The data may be in a proprietary file format and you would like it in a relational database or text file, or the data may be in a relational database and you would like it in a flat file.

Cleaning

Data cleaning is the process of replacing missing data. There can be data instances that are insufficient and lack the information you think you need to address the issue. These occurrences might need to be eliminated.

Sampling

You may have access to much more data than you actually need that has been carefully chosen. Algorithms may require more compute and memory to run as well as take significantly longer to process larger volumes of data. You can choose a smaller representative sample of the chosen data, which may be much faster for exploring and testing ideas, rather than thinking about the complete dataset.

2. EXTRACTION OF FEATURES

The next step is to A process of attribute reduction is Feature extraction. Feature extraction actually alters the attributes as opposed to feature selection, which ranks the current attributes according to their predictive relevance. The original attributes are linearly combined to generate the changed attributes, or features. Finally, the Classifier algorithm is used to train our models. We make use of the acquired labelled dataset. The models will be assessed using the remaining labelled data we have. Pre- processed data was categorised using a few machine learning methods. Random forest classifiers were selected.

4.2ALGORITHM DESCRIPTION 2

Algorithms use training data and human feedback to learn the relationship between specific inputs and specific outputs. For example, a medical practitioner might use marketing costs and weather forecasts as input data to predict can sales. Supervised learning can be used when the output data is known. Algorithms predict new data. There are two categories of supervised learning.

Classification

Suppose you want to predict the gender of a commercial customer. Collect data about height, weight, occupation, salary, shopping cart, etc. from your customer database. You know the gender of each customer, but only male or female. The purpose of the classifier is to assign probabilities of whether you are male or female (i.e. a label) based on information (i.e. features collected from you). Once the model learns to recognize males or females, it can use new data to make predictions. For example, suppose you just received new information from an unknown customer and want to know if the customer is male or female. If the classifier predicts Male = 70%, it means that the algorithm has 70% confidence that this customer is male and she is 30% female. A label can consist of two or more classes. The example above has only two classes, but there are dozens of classes (glass, table, shoes, etc.) if the classifier needs to predict an object. Each object represents a class).

Recurrence

If the output is continuous, the task is regression. For example, a financial analyst may need to predict the value of stocks based on many characteristics such as stocks, historical stock performance, macroeconomic indicators, and more.

Linear regression

A machine learning algorithm built on supervised learning is linear regression. Activate a regression task. Run a regression task. Regression models target predictors based on independent variables. Different regression models differ based on the type of relationship between dependent and independent variables considered and the number of independent variables used. The dependent variable in regression has many names. This is sometimes called the outcome variable, criterion variable, endogenous variable, or regression sand. Independent variables are sometimes called exogenous variables, predictor variables, or regressors.

Logistic regression

Logistic regression is one of the most popular machine learning algorithms that falls under supervised learning techniques. Logistic regression predicts the output of a categorical dependent variable. Using a specific collection of independent factors, it is used to predict a categorical dependent variable. As a result, the outcomes must be discrete or categorical. can be true or false, 0 or 1, yes or no, and so forth. But instead of giving exact values as 0 and 1, it gives probability values between 0 and 1. Logistic regression is very similar to linear regression except for how it is used. Linear regression is used to solve regression problems and logistic regression is used to solve classification problems.

Decision Tree

Decision trees are a supervised learning technique that can be used for both classification and regression problems, but they are mostly suitable for solving classification problems. It is a tree- structured classifier, with internal nodes representing characteristics of the data set, branches representing decision rules, and each leaf node representing a result. A decision tree has his two nodes, a decision node and a leaf node. Decision nodes are used to make decisions and have multiple branches, while leaf nodes are the result those decisions and contain no further branches. A decision or test is made based on the characteristics of a particular data set. Graphical representation to get all possible solutions to decision based on given conditions.

Unsupervised learning

In unsupervised learning, algorithms examine input data without explicit output variables (for example, examine customer demographics to identify patterns). It can be used when you don't knowhow to classify your data and want the algorithm to find patterns and classify your data.

CHAPTER 5

RESULT AND DISCUSSION

CHAPTER 5

RESULT AND DISCUSSION

5.1 PERFORMANCE PARAMETER/TESTING

Once your model is built, you can test its performance against unprecedented data. The new data is transformed into feature vectors and the model is run to make predictions. That's the great thing about machine learning. No need to update rules or retrain models. A previously trained model can be used to make inferences on new data. The lifespan of a machine learning program is straightforward and can be summed up in the following points:

1. Define the question
2. Collect data
3. Visualize your data
4. Algorithm training
5. Test your algorithm
6. Gather feedback
7. Improve your algorithm

Use the model to make predictions. Once the algorithm is able to draw the correct conclusions, it applies that knowledge to new data sets. Machine learning algorithms and where to use them Machine learning can be divided into two broad learning tasks supervised and unsupervised.Software testing is research conducted to provide interested parties with information about the quality of the product or service being tested. Software testing also provides an objective and independent view of software, enabling organizations to assess and understand the risks involved in implementing software.

Software testing can also be described as the process of validating and verifying softwareprograms/applications/products.

- Meet business and technical requirements that guide design and development.
- Works as expected and can be implemented with the same characteristics. Test

Components	Machine learning	Deep learning
Training dataset	Small	Large
Choose features	Yes	No
Number of algorithms	Many	Few
Training time	Short	Long

Table 5.1.1 Training Model Survey

Method

1.Functional test

Functional testing systematically demonstrates the availability of tested functionality according tobusiness and technical requirements, system documentation, and user guides.

Function

Must perform the identified function.

- Output: A software output of the identified class must be performed.
- Systems/procedures: Systems must function properly.

2. Integration testing

Software integration testing is incremental integration testing of two or more software components integrated on a single platform, producing errors caused by interface flaws.

A test case for checking an Excel spreadsheet:

Machine learning here deals with data sets in the form of Excel spreadsheets. So if you want a test case, you should check the Excel file. Classification is then performed on the corresponding columnsof the dataset.

Test Case 1: Set of csv input file

Process: A few specifics about the Android device components are included in the dataset after whichthe feature selection process is effectively finished.

Output: 'MALWARE'

Test Case 2: Another set of csv input file

Process: A few specifics about the Android device components are included in the dataset after which the feature selection process is effectively finished.

Output : 'BENIGN'

5.2 RESULT AND DISCUSSION

Conclusion:

In conclusion, this study highlights the significance of utilizing Machine Learning Algorithms to combat malicious calls, safeguarding privacy and security in telephony networks. By collecting and analyzing consumer data, an effective prevention mechanism can be developed to classify calls accurately, ultimately enhancing trust and reliability in communication systems.

Future enhancement:

Future enhancements in Android malware detection through machine learning could involve deep learning advancements tailored to Android app analysis, addressing adversarial attacks to bolster model robustness, and ensuring explainability of AI-driven decisions for improved trust. Real-time detection capabilities could be augmented via online learning and distributed computing, while privacy-preserving techniques like federated learning could safeguard user data. Behavioral analysis in runtime could provide richer insights into malware activities. Extending detection to IoT devices and leveraging mobile edge computing would enhance comprehensive protection. Cross-platform compatibility efforts could broaden defense capabilities, and collaborative mechanisms would facilitate shared threat intelligence for proactive defense strategies. Ethical considerations regarding transparency and fairness in model deployment remain paramount.

CHAPTER 6

CONCLUSION AND FUTURE WORK

CHAPTER 6
CONCLUSION AND FUTURE WORK APPENDICES

A.1 SDG Goals

The Sustainable Development Goals (SDGs) are a collection of 17 global goals set by the United Nations General Assembly in 2015. These goals aim to address various global challenges and promote sustainable development worldwide. While there are 17 SDGs in total, Here we provide a list of 16 SDGs along with how they can be related to Android malware detection using machine learning techniques:

No Poverty:Machine learning-based malware detection can help protect users from financial fraud and scams, thereby contributing to reducing economic vulnerabilities.

Zero Hunger:By ensuring the security of mobile devices against malware, individuals can access food-related information and services without disruptions, contributing indirectly to zero hunger.

Good Health and Well-being:Malware-free devices lead to a safer online environment, promoting the well-being of individuals by preventing health-related scams, phishing attacks, and unauthorized data access.

Quality Education:Protecting educational resources and platforms from malware ensures uninterrupted access to quality education through mobile devices, benefiting students

Gender Equality:Ensuring the security of mobile devices and digital platforms promotes equal access to technology and online opportunities for people of all genders, contributing to gender equality in the digital space.

Clean Water and Sanitation:While not directly related, efficient use of technology, including secure mobile devices, can indirectly contribute to optimizing water

Affordable and Clean Energy:Secure mobile devices enable efficient energy management through smart applications, contributing to affordable and clean energy usage in the digital era.

Decent Work and Economic Growth:Protecting devices from malware ensures the integrity of digital transactions and business operations, supporting economic growth and creating a conducive environment for decent work opportunities.

Industry, Innovation, and Infrastructure:Machine learning in malware detection fosters innovation in cybersecurity, strengthening digital infrastructure and promoting a secure digital ecosystem for technological advancements..

Reduced Inequality:Access to secure digital platforms and protected mobile devices reduces the digital divide and inequalities related to cyber threats, safe online experience.

Sustainable Cities and Communities:Secure mobile devices contribute to smart city initiatives by safeguarding data and digital services, supporting sustainable urban development and community resilience.

Responsible Consumption and Production:By preventing malware attacks on digital platforms, responsible consumption and production practices can be promoted through secure online transactions and data management.

Climate Action:While not directly linked, efficient use of technology, including secure mobile devices, can indirectly support climate action through data-driven insights

Life Below Water:Malware detection and prevention contribute to protecting marine-related data and resources, indirectly supporting efforts for the conservation of life below water in the digital realm.

Life on Land:Secure mobile devices and digital platforms help in the conservation of land-related data, biodiversity information, and environmental monitoring, supporting initiatives for life on land.

Peace, Justice, and Strong Institutions:Machine learning-based malware detection strengthens cybersecurity measures, promoting peace and stability in digital environments and ensuring the integrity of institutions and justice systems.

A.1 SOURCE CODE

DEPENDENCIES / PACKAGES TO IMPORT

- pip install streamlit
- pip install pickle-mixin
- pip install numpy
- pip install pandas
- pip install -U scikit-learn
- pip install python-dotenv
- pip install TIME-python
- pip install os-sys
- pip install pillow

app.py

```
import
time
import os
from PIL import
Imageimport
pickle
import numpy as
np import
pandas as pd
import streamlit
as stimport os
from dotenv import load_dotenv, find_dotenv
load_dotenv(find_dotenv())
# load user details from dotenv file
# userName = os.environ.get("USER")
# password = os.environ.get("PASSWORD")
```

```
userName = "Admin"
password = "Pass"logStatus = False title = st.empty()

if logStatus == False:
title.markdown("<h1 style='text-align: center; color: #ffff;'>Admin
Login</h1>",unsafe_allow_html=True)

userContainer =
st.empty()
passwordContainer =
st.empty()
user = userContainer.text_input('Admin ID')
passCode = passwordContainer.text_input('Input Password', type="password”


# check log in credentials
if user != "" and passCode != "" and user == userName and
passCode == password:title.empty()
logStatus = True

elif user != "" and passCode != "" and (user != userName or passCode !=
password):st.warning('Wrong ID or Password! please try again')
if user == userName and passCode == password:

userContainer.empty()
passwordContainer.empty()
time.sleep(1)
```

```
# while the log in credentials(logStatus)
is TRUEif logStatus:
image = Image.open("dynamic-file-analysis.jpg")
st.image(image, use_column_width=True)
st.markdown("<h1 style='text-align: center; color: #ffff;'> Android Malware Detection</h1>",
unsafe_allow_html=True)
st.write("""
#####  This detects whether the Application is Malicious or
Benign""")st.sidebar.header('Input Android App Features')
# Collects user input features into
dataframeuploaded_file =
st.sidebar.file_uploader( "Upload
your input CSV file", type=["csv"])
if uploaded_file:
data =
pd.read_csv(uploaded_file)
test = data.astype(str)
st.dataframe(test)
if st.button('Feature Selection'):
# Performe Feature Extraction
dataToUse = data[['SEND_SMS', 'android.telephony.SmsManager',
'READ_PHONE_STATE', 'RECEIVE_SMS', 'READ_SMS',
'android.intent.action.BOOT_COMPLETED', 'TelephonyManager.getLine1Number',
'WRITE_SMS',
'WRITE_HISTORY_BOOKMARKS', 'TelephonyManager.getSubscriberId',
'android.telephony.gsm.SmsManager', 'INSTALL_PACKAGES',
'READ_HISTORY_BOOKMARKS', 'INTERNET',
'AUTHENTICATE_ACCOUNTS',
'android.os.IBinder', 'IBinder', 'Binder', 'CAMERA',
'READ_SYNC_SETTINGS',  'Ljava.lang.Class.forName',
'Runtime.getRuntime', 'Ljava.lang.Object.getClass', 'mount',
'android.intent.action.SEND',
```

'USE_CREDENTIALS', 'MANAGE_ACCOUNTS',
'Ljavax.crypto.Cipher', 'System.loadLibrary',
'DexClassLoader', 'getCallingUid', 'SecretKey',
'Ljava.lang.Class.getMethod', 'HttpGet.init', 'KeySpec',
'android.content.pm.PackageInfo',
'Ljavax.crypto.spec.SecretKeySpec', 'getBinder',
'GET_ACCOUNTS', 'Ljava.lang.Class.getDeclaredField',
'Landroid.content.Context.unregisterReceiver',
'Ljava.lang.Class.getField',
'Landroid.content.Context.registerReceiver',
'ClassLoader', 'android.content.pm.Signature',
'Ljava.lang.Class.getMethods',
'Ljava.lang.Class.cast', 'Ljava.net.URLDecoder',
'Ljava.lang.Class.getCanonicalName',
'attachInterface', 'android.os.Binder',
'ServiceConnection', 'bindService',
'onServiceConnected', 'transact']]
title2 = st.empty()
title2.write("Selecting
feature...")my_bar =
st.progress(0)
for percent_complete in range(100):
time.sleep(0.05)
my_bar.progress(percent_complete +
1)my_bar.empty()
title2.empty()
my_bar.success('Feature Selected
Successfully!')time.sleep(2)
my_bar.empty()
# Reads in saved classification model
dataModel =
pickle.load(open('stacking.pki', 'rb'))#

```
Apply model to make predictions
prediction = dataModel.predict(dataToUse)
prediction_proba =
dataModel.predict_proba(dataToUse)
modelDetection = np.array(['Benign',
'Malicious'])
# st.write(modelDetection[prediction])
st.subheader('Prediction Probability')
st.write(prediction_proba)
if (modelDetection[prediction][0] ==
"Benign"):st.write("### STATUS:
'BENIGN' ", )
else:
st.write("### STATUS: 'MALWARE' ", )
else:
st.write('Click 👆 To Perform Feature selection')
else:st.write("Waiting for CSV file to be uploaded...")
```

Android malware-Checkpoint-ipynb

```
# Basic Import
import time, datetime # to track our Time
Complexityimport os # accessing directory
structure
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g.
pd.read_csv)import matplotlib.pyplot as plt # for plotting
import seaborn as sns # for Visualizing
our dataimport streamlit.web.cli as
stcli
sns.set_style('whitegrid') # change seaborn style to white grid


# Preprocessing
from sklearn.preprocessing import
LabelEncoder from
sklearn.model_selection  import
train_test_split


# MACHINE LEARNING  ALGORITHMS
from sklearn.neighbors import
KNeighborsClassifierfrom sklearn.svm
import SVC
from sklearn.ensemble import
RandomForestClassifierfrom
sklearn.ensemble import StackingClassifier
```

```
# For Metrics Performance and Result Check
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import confusion_matrix,
    classification_report,accuracy_score, f1_score,precision_score, recall_score

from sklearn.metrics import ConfusionMatrixDisplay

from sklearn import model_selection, tree, preprocessing, metrics,
linear_model# for dumping our model

import pickle

#Import Dependencies
%matplotlib inline

data = pd.read_csv("drebin-215-dataset-5560malware-9476-benign.csv")
```

2. Data Cleaning

We going to start the data cleaning process from the target variable
"class"data['class'].nunique() # Number of unique values of the
Target class
[4]:

So we can see that the class is obviously in string therefore we going to be converting it to an integer

```
    classes,count =
    np.unique(data['class'],return_counts=True)
    #Perform Label Encoding
    lbl_enc = LabelEncoder()
    print(lbl_enc.fit_transform(classes),classes)
    data = data.replace(classes,lbl_enc.fit_transform(classes))
```

So we can see that the class is obviously in string therefore we going to be converting it to an integer

```
classes,count = np.unique(data['class'],return_counts=True)
#Perform Label Encoding
lbl_enc = LabelEncoder()
print(lbl_enc.fit_transform(classes),classes)
data = data.replace(classes,lbl_enc.fit_transform(classes))

#Dataset contains special characters like "?' and 'S'. Set them to NaN and use
dropna() to removethem
data=data.replace('[?,S]',np.NaN,regex=True)
print("Total missing values : ",sum(list(data.isna().sum())))
data.dropna(inplace=True)
for c in data.columns:
  data[c] = pd.to_numeric(data[c])
data.isnull().sum()

print("Total missing values : ",sum(list(data.isna().sum())))
sns.heatmap(data.isnull(),cbar=False,yticklabels=False,cmap = 'viridis')
```

3. Exploratory Data Analysis

```
data.info()
<class
'pandas.core.frame.DataFrame'>
Index: 15031 entries, 0 to 15035
Columns: 216 entries, transact
to classdtypes: int32(1),
int64(215)
memory usage: 24.8
MBdata.shape
```

```
data.describe()
# Confirm to make sure all column are of length
15031counts = True
result = []
dataToCheck = data.columns
startTime = time.time()

def
  necessaryChecks
  (): for i in
  range(0, 216):
    if data[data.columns[i]].count() == 15031:
      result.append(True)
    else:
      result.append(False)
  return result
print(necessaryChecks())
checkTime = (time.time() - startTime)
print("Running Time: %s" %
datetime.timedelta(seconds=checkTime))data.corr()
#Let us first analyze the distribution of the target
variableMAP={}
labels = ["Malware", "Benign"]
#Let us first analyze the distribution of the target
variableMAP={}
labels = ["Malware", "Benign"]
for e, i in
  enumerate(data["class"].unique())
  :MAP[i]=labels[e]
#MAP={0:'Not-Survived',1:'Survived'} df1 = data.copy()
```

```
df1["class"]=df1["class"].map(MAP)
explode=np.zeros(len(labels))
explode[-1]=0.1
print('\033[1mTarget Variable Distribution'.center(55))
plt.pie(df1["class"].value_counts(),
                                    labels=df1["class"].value_counts().index,
                                    counterclock=False, shadow=True,
    explode=explode, autopct='%1.1f%%', radius=1,
startangle=0)plt.show()
```

4. Feature Selection

```
#### We would be making use of the pearson correlation to extact
our featuredataTargetFeature = data.corr()['class'][:-1]
dataTargetFeature

Number of Features having correlation >0.5 with target variable
data_feature_selection_corr = dataTargetFeature[abs(dataTargetFeature)
                              >0.2].sort_values(ascending=False)
data_feature_selection_corr

data_feature_selection_corr.index

data_feature_selection = data[data_feature_selection_corr.index]
data_feature_selection=data_feature_selection.join(data['class'])
data_feature_selection.head()
# Extract selected data to csv format
data_feature_selection.to_csv("Feature.csv",
index=False)
data_feature_selection.corr()
k = 15
```

#number of variables for heatmap
cols = data_feature_selection.corr().nlargest(k,
'class')['class'].indexcm =
data_feature_selection[cols].corr()
plt.figure(figsize=(10,6))
sns.heatmap(cm, annot=True, cmap = 'viridis')
data_feature_selection.shape

6. Predictive Modeling

Train Test Split

**So its time to split the dataset into training set and
testing test**X =
data_feature_selection.drop('class',axis=1)
y = data_feature_selection['class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=101, shuffle=True) print("Train features size : ",len(X_train))
print("Train labels size :
",len(y_train)) print("Test
features size : ",len(X_test))
print("Test features size :
",len(y_test))
Let's Extract the column we going to be using for our tests Data
during Deploymentmalware = data[data['class'] == 1]
benign = data[data['class']
== 0]malware.head()

benign.head()

malwareFirst =
malware.iloc[0:1]benignFirst =
benign.iloc[0:1]

```
malwareFirst
benignFirst
malwareFirst.to_csv("testSet1.csv", index=False)
benignFirst.to_csv('testSet2.csv', index=False)
```

function to print out our classification

```
def Classification_Summary(pred, i, elapsed_time):
   Evaluation_Results.iloc[i]['Accuracy']=round(accuracy_score(y_test,
   pred),4)*100
   Evaluation_Results.iloc[i]['Precision']=round(precision_score(y_test,
   pred),4)*100  #
   Evaluation_Results.iloc[i]['Recall']=round(recall_score(y_test,
   pred),4)*100 # Evaluation_Results.iloc[i]['F1-
   score']=round(f1_score(y_test, pred),4)*100 #
   Evaluation_Results.iloc[i]['Time(sec)']= round(elapsed_time, 2)
   print('{}{}\033[1m Evaluating {} \033[0m{}{}\n'.format('<'*3,'-
'*35,Evaluation_Results.index[i], '-'*35,'>'*3))
   print('Accuracy = {}%'.format(round(accuracy_score(y_test,
   pred),4)*100))print('F1 Score =
   {}%'.format(round(f1_score(y_test, pred),4)*100)) #
   print('\n \033[1mConfusiton Matrix:\033[0m\n',confusion_matrix(y_test, pred))
   print('\n\033[1mClassification Report:\033[0m\n',classification_report(y_test, pred))
   sns.heatmap(confusion_matrix(y_test, pred), annot=True, cmap="Blues", fmt="d")
```

Let us create first create a table to store the results of various models

```
Evaluation_Results = pd.DataFrame(np.zeros((4,5)), columns=['Accuracy', 'Precision','Recall',
'F1 -score', 'Time(sec)'])
Evaluation_Results.index=['Support Vector Machine (SVM)','K Nearest Neighbours
(KNN)','RandomForest Classifier (RF)',
                  "Stacking Classifier"]
```

Evaluation_Results

A.SUPPORT VECTOR MACHINE (SVM)

```
st = time.time()
# # Building Support Vector Machine Classifier
SVM_Model = SVC()
SVM_Model.fit(X_train,y_train)
SVM_Pred  =
SVM_Model.predict(X_test)

et = time.time()
svm_time = et - st
print("Running Time: %s" % datetime.timedelta(seconds=svm_time))
Classification_Summary(SVM_Pred,0, svm_time)
```

A. K Nearest Neighbour (KNN)

```
import numpy as np
start_time = time.time()
X_train_np  = np.array(X_train)
X_test_np = np.array(X_test)
y_train_np = np.array(y_train)

KNN_Model =KNeighborsClassifier(n_neighbors=40)
fitKnn = KNN_Model.fit(X_train_np, y_train_np)
KNN_Pred = KNN_Model.predict(X_test_np)
knn_time = (time.time() - start_time)
Classification_Summary(KNN_Pred,1,  knn_time)
```

C. RANDOM FOREST CLASSIFIER (RF)

```
# Random Forest Classifier
start_time = time.time()
RF_Model =    RandomForestClassifier(n_estimators=1)
RF_Model.fit(X_train, y_train)
RF_Pred = RF_Model.predict(X_test)
rf_time = (time.time() - start_time)
```

```
print("Running Time: %s" % datetime.timedelta(seconds=rf_time))
Classification_Summary(RF_Pred,2, rf_time)
Evaluation_Results
```

	Accuracy	Precision	Recall	F1-score	Time(sec)
Support Vector Machine (SVM)	96.41	97.49	92.83	95.10	5.66
K Nearest Neighbours (KNN)	94.88	95.44	90.71	93.01	2.47
Random Forest Classifier (RF)	95.81	95.23	93.54	94.38	0.08
Stacking Classifier	0.00	0.00	0.00	0.00	0.00

Gridsearch

Finding the right parameters, this idea of creating a 'grid'of parameters and just trying out all the possible combinations is called a Gridsearch, this method is common enough that Scikit-learn has this functionality built in with GridSearchCV! The CV stands for cross-validation which is theGridSearchCV takes a dictionary that describes the parameters that should be tried and a model to train.

```
param_grid = {'C': [0.1,1, 10, 100, 1000], 'gamma': [1,0.1,0.01,0.001,0.0001], 'kernel': ['rbf']}
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=3)
svcgrid = grid.fit(X_train,y_train)
grid.best_params_
pip show scikit-learn
import numpy as np
import scipy as sp
print("NumPy version:", np.__version__)
print("SciPy version:", sp.__version__)
import numpy as np
X_train = np.array(X_train)
X_test = np.array(X_test)
y_train = np.array(y_train)
y_test = np.array(y_test)
X_train = np.ascontiguousarray(X_train)
X_test = np.ascontiguousarray(X_test)
```

Choosing a K Value

Let's go ahead and use the elbow method to pick a good K Value:pip install --upgrade scikit-learn

```
start_time =
time.time()
error_rate = []
for i in range(1,40):

knn = KNeighborsClassifier(n_neighbors=i)
knn.fit(X_train,y_train)
pred_i = knn.predict(X_test)
error_rate.append(np.mean(pred_i != y_test))

knn_time = (time.time() - start_time)

print("Running Time: %s" % datetime.timedelta(seconds=knn_time))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue',linestyle='dashed',marker='o',
      markerfacecolor='red',markersize=10)
plt.title = ('Error Rate vs K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

Stacking the bagging classifiers

Define estimators

```
start_time =
time.time()
estimator_list = [
```

```
    ('svm_rbf', SVC(gamma=0.1, C=10)),
    ('knn',
    KNeighborsClassifier(n_neighbors=4)),
    ('rf',
    RandomForestClassifier(n_estimators=100
    ))
]

# Build stack model
stack_model = StackingClassifier(
    estimators=estimator_list,final_estimator=RandomForestClassifier(n_estimators=100)
)


# Train stacked model
stack_model.fit(X_train,
y_train) stack_time =
(time.time() - start_time)
print("Running Time: %s" %
datetime.timedelta(seconds=stack_time))stack_pred =
stack_model.predict(X_test)
Classification_Summary(stack_pred,3, stack_time)
Evaluation_Results.sort_values(by='Accuracy',
ascending=False)
```

Comparing all the models Scores

```
print('\033[1mML  Algorithms
Comparison'.center(130))plt.figure(figsize=[10,6])
sns.heatmap(Evaluation_Results, annot=True, vmin=95, vmax=100, cmap='YlGnBu',
fmt='.1f')plt.show()
pickle.dump(stack_model,  open("stacking.pki",  "wb"))
```

A.3 SCREENSHOTS
LOGIN PAGE

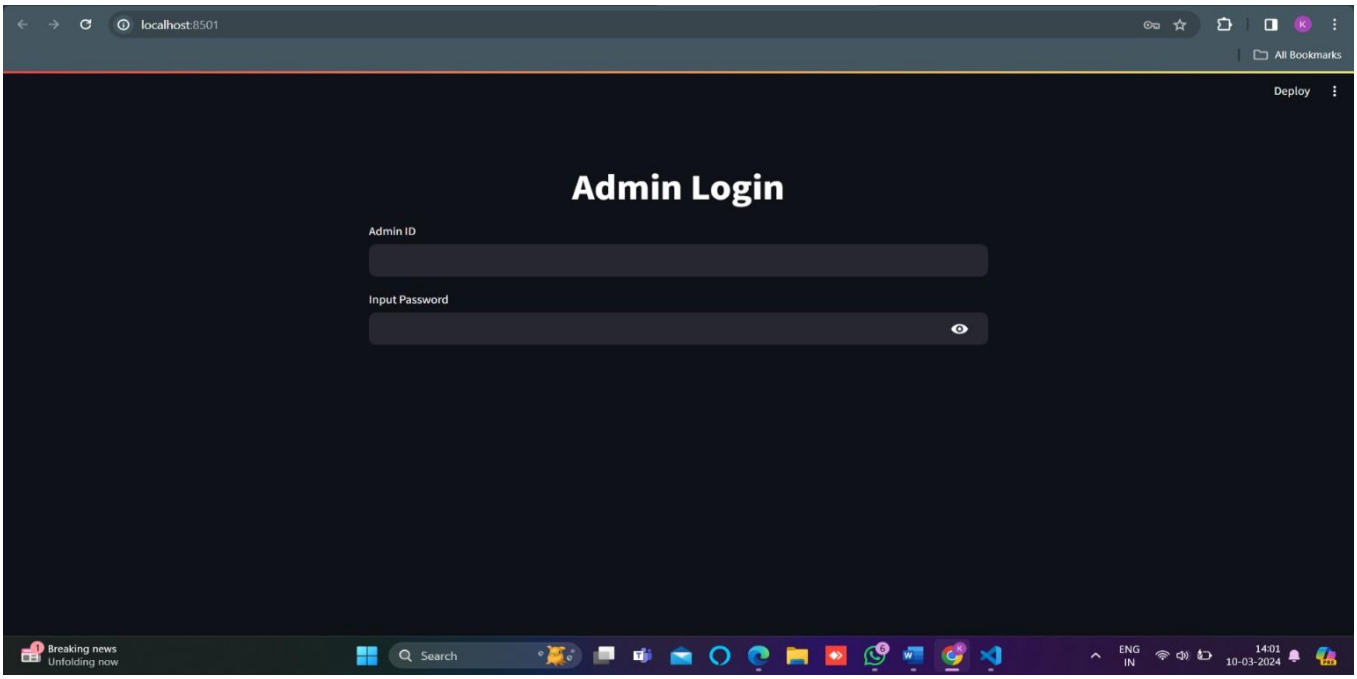


Fig A.3.1 Login Page

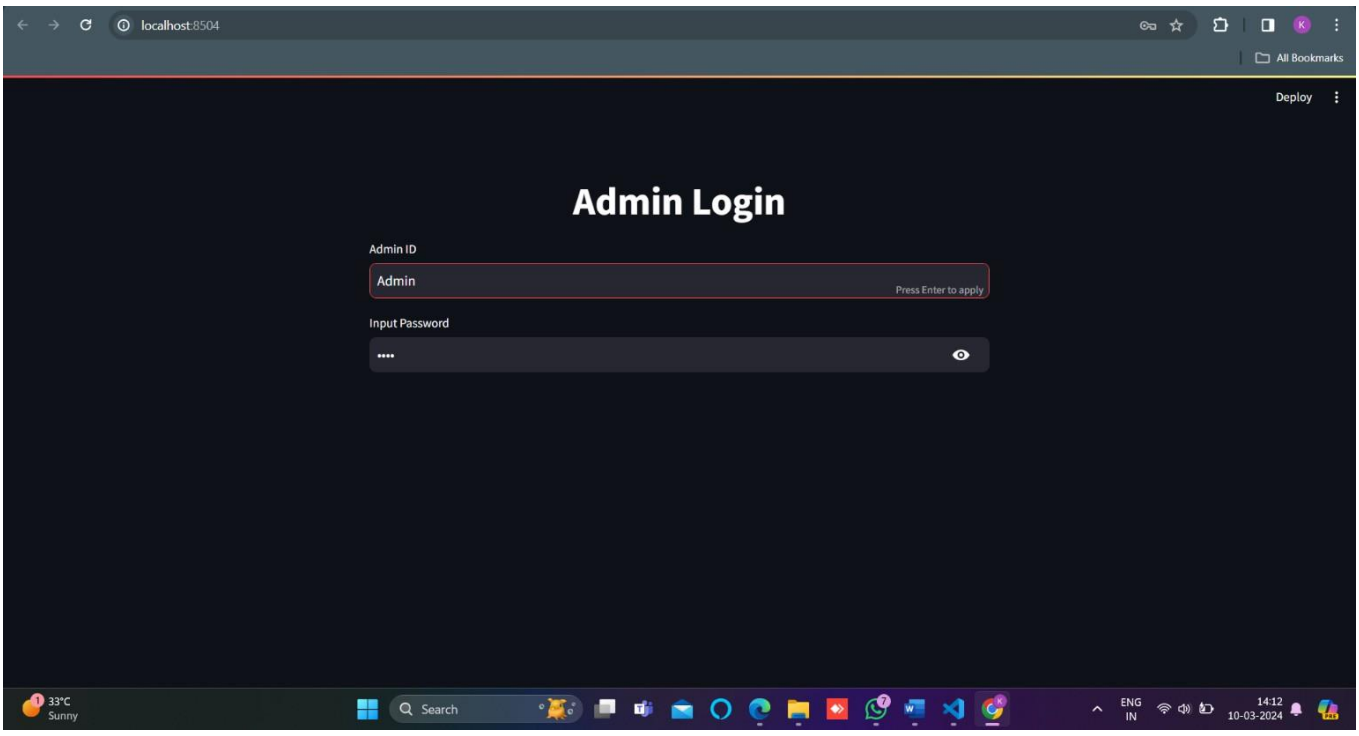


Fig:A.3.2 Login Credentials

INPUT SELECTION

Fig:A.3.3 Browsing DataSet

Fig:A.3.4 Selecting DataSet

OUTPUT

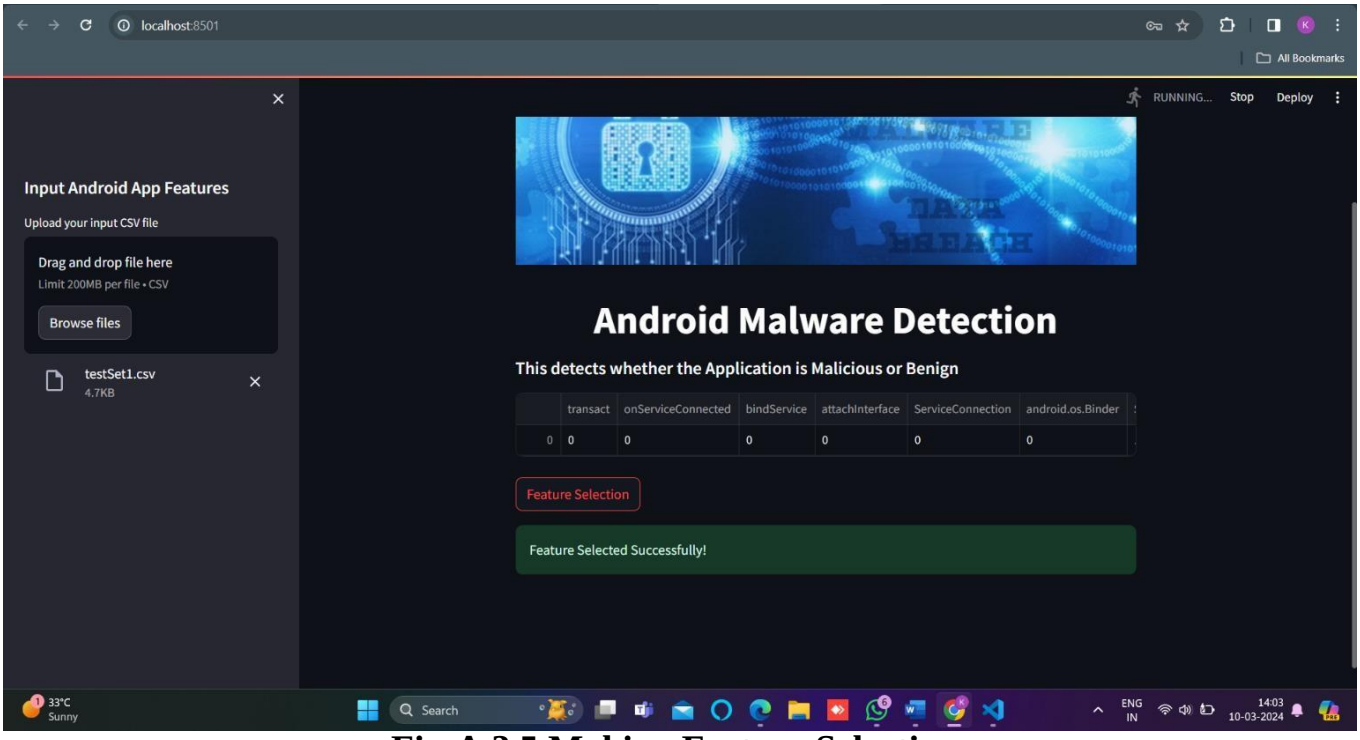


Fig:A.3.5 Making Feature Selection

Fig:A.3.6 Prediction Status 1

Fig:A.3.7 Feature Selection

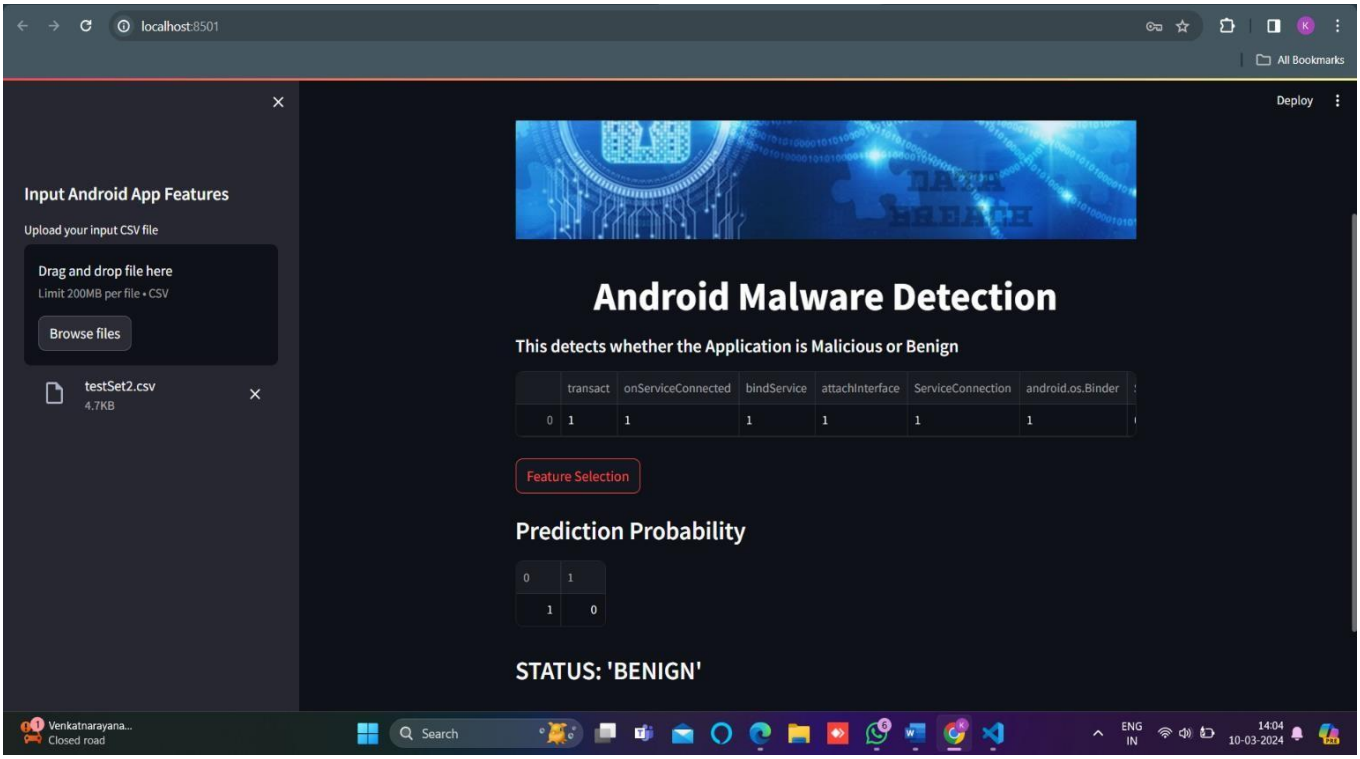


Fig:A.3.8 Prediction Status 2

A.4 PLAGIARISM REPORT

P
by John Smith

Submission date: 23-Mar-2024 06:00AM (UTC+0000)
Submission ID: 223610564
File name: copypaste.txt (23.57K)
Word count: 3431 Character count: 20088

A.5 PAPER PUBLICATION

SECURING ANDROID DEVICES: A ROBUST APPROACH TO AUTOMATED MALWARE DETECTION THROUGH ENSEMBLE LEARNING

Mr.Sasi Kumar AN
Computer science dept
Panimalar Engineering
College,Chennai, India

John Praveen J
Computer Science Dept
Panimalar Engineering
College,Chennai, India

Kiran M P
Computer Science dept.
Panimalar Engineering
College,Chennai, India

Prabaakaran C M
Computer Science Dept
Panimalar Engineering
College, Chennai, India

ABSTRACT

Malware has been used for hacks for a long time. Smartphones have emerged as a major target for malware makers due to their widespread utilization and the volume of private and delicate information they contain. Android has always been the most popular smartphone operating system.An intriguing platform for malware developers, and an abundance of Android malware species abuse vulnerable users on a daily basis, making manual malware research unfeasible. Machine learning techniques for malware forensics could be useful for cyber forensic technicians to counteract malicious programs. In this study, we describe two machine learning aided approaches to a mobile application static analysis: permissions analysis-based and source code analysis leveraging a bag of content representation.
.

Our initial code-based categorization method produced a factor score of 95.1%, while the authorization name strategy only produced an F-measure of 89%. Our technology delivers an automated method for high-accuracy recognition of malware and static code analysis, reducing the time required to analyze malware on smartphones. In this investigation, we tackle the harmful call prevention problem without utilizing any particular basic telephone network technology. Thus, the initial challenge is to obtain accurate data. The main contribution of this venture is gathering data on fraudulent callers in order to use machine learning techniques to develop an efficient preventative device. We have assembled the dataset using information about customers. It incorporates three distinct class types: rob calls, undesirable (malicious) calls, and telemarketing.

Keywords: Machine learning · Android · Static malware analysis

1.INTRODUCTION

According to Ericsson, the total quantity of smartphone users broadly is projected to rise from 2.6 billion in 2014 to a staggering 6.1 billion by 2020 (Boxall 2015). Users had the ability to have a pocket-sized device with the full features of a conventional mobile phone, coupled with an adequate amount of processing capacity and internet connectivity, through mobile phones. Despite the fact that a great deal of private data was saved on mobile devices, nefarious individuals and cybercriminals found them to be a captivating target. Mobile devices are a valuable target for hackers because users use them to access

online payment systems, bank accounts, and social media. Attackers could, however, use the enormous processing power of handheld gadgets for cryptocurrency mining or Dos operations (Pan et al. 2014; Felt et al. 2011).

One in five mobile users fell victims to a cyberattack at least once a year, estimates Kaspersky Lab (Interpol and Kasperski lab 2014). Ten percent of users have at least once been the victim of money-stealing software. Trojans the fact that sent SMS messages were the most prominent perilous software all through the reporting period. Between August 2013 and March 2014, the monthly attack count rose by almost 10 times, from 69,000 in August 2013 to 644,000 in March 2014. According to Intel Security, which states that the yearly cost of cybercrime losses fluctuate between $375 to $575 billion (McAfee, Center for Strategic & International Studies 2014), cybercriminals are focusing growing increasingly on mobile media apps coming in second and third (Cyberedge 2014Along with the upsurge in popularity of mobile smart devices has come an increase in the prevalence of malware and attacks (Shaerpour et al. 2013). Smartphones have been abused as objects, subjects, or tools in a number of cybercrimes (Mohtasebi and Dehghantanha 2013).Malware has already been used as a tool for illegal conduct during numerous preceding security breaches (Damshenas et al. 2013).A cyber attacker can launch denial-of-service campaigns, install or remove programs, modify files, download private data and use it to assume the identity of the owner of infected devices, upload files, record user keystrokes and actions, capture the victim's screen, use the camera to recover pictures or videos, and more using a system that has malware installed on it (Skoudis and Zeltser 2004Nevertheless operating systems with greater market penetration are the ones that malware authors are most likely to attack. As of the time of writing, 82% of mobile users were on Android devices (Kitagawa et al. 2015).Android users have been a prime target for malware developers due to the substantial market penetration of the Android operating system and Google's loose publication expectations on Google Play, the official Android software store. Several malware releases, including Trojan horses, have centered on Google Play (Reina et al. 2013; Viennot et al. 2014). In the first quarter of 2015, approximately 5,000 distinct Android malware files emerged daily, according to security firm G-Data (GData 2015).

In order to prevent malware infection, malware analysis and tools that does malware analysis are essential (Daryabar et al. 2013). According to Dezfouli et al. (2013), there are two basic methods for analyzing malware: dynamic malware analysis and static malware analysis. The foundation of static analysis is the examination and inspection of binaries and source code to look for unusual patterns. However, dynamic analysis, also known as behavioral-based analysis, entails running the software under analysis in a separate setting while keeping an eye on and tracking behavior (Christodorescu and Jha 2006;Schmidt et al. 2009a; Shabtai et al. 2010; Rieck et al. 2008; Christodorescu et al. 2008; Lee and Mody 2006). Finding abnormalities in battery usage that might be brought on by malware activity was one of the early methods for identifying mobile malware (Buennemeyer et al. 2008; Jacoby et al. 2004; Kim et al.2008, 2011). Operating system events that could be useful for dynamic malware detection include resource locks, I/O requests, API calls, and battery usage (Schmidt et al. 2009a,b, 2010; Bl¨asing et al. 2010). According to Eck et al. (2014), TaintDroid is a malware detection system that tracks the behavior of applications and finds irregularities. In order to differentiate malicious code, Canfora et al. (2015) developed a system that tracks six Android Dalvik op-codes and analyzes their frequency. applications from those that are benign. To prevent the performance of mobile devices from declining, collaborative analysis and distributed computing-based solutions for both static and dynamic(Cheng et al. 2007; Schmidt et al. 2009a; Shamili et al. 2010) suggested malware analysis. On the server, M0Droid is generating signatures by examining the system calls made by Android applications. Afterwards, the devices receive signatures so they may alert users to potential dangers (Damshenas et al. 2015).Keeping up with the constant influx of new malware on mobile platforms becomes challenging. Research is still being done on mobile malware detection systems, which are still in their infancy (Enck et al. 2014).Traditional methods of detecting malware are either dependent on following the actions of malicious apps, which is useless against malware that exhibits unique behavior, or on signature detection, which is ineffectual against malware that is encrypted, metamorphosed, or polymorphed

II. RELATED WORKS

Shaukat et al. [11] create a novel method of identifying malware that corresponds to DL. Through the integration of static and dynamic analysis, it generated better results compared to traditional approaches. By using image-based deep

learning, Geremias et al. [12] introduced a novel multi-view malware detection for Android strategy that can be applied in three sections. Apps were initially evaluated using the various feature sets available in the multi-view settings, which expanded the quantity of information available for categorisation. Second, when the final feature set is transformed into picture formats, the data for the classifier task is kept while retaining the vital aspects of data distribution. Thirdly, the pictures produced are all exhibited together in a single framework inside a specified image channel, facilitating the deployment of DL structure.
Kim et al. [13] modelled MAPAS, a malware detection system that achieves improved accuracy and flexible use of computing. capacity. MAPAS employed CNN to evaluate the malicious apps' performance based on their API call graphs. Rather than employing a classifier built using CNN, the MAPAS method that is offered uses CNN to identifycommon features of the call graph for the API used by the malware. Fallah and Bidgoly [14] developed an LSTM-based malware detection method that is capable of recognizing and detect new and unreported malware variants in addition to discriminating between malicious and benign samples.

The author of this study conducted numerous experiments to demonstrate the capabilities of the method being given, including novel malware families. detection, identification of malware, identification of malware families, and evaluation of the least amount of time required to locate malware.
DL-based Android malware identification with DYnamic characteristics (De-LADY), a robust obfuscation technique, was first presented by Sihag et al. [15]. Utilizing an emulated environment, it has employed behavioral features from dynamic analysis of an application. A hybrid approach pertaining to DAE and CNN is presented by Wang et al. [16]. In order to improve the accuracy of malware detection, the author first rebuilt the high-dimensional features of programs and employed many CNN to locate malware for Android. Second, the author employed DAE as a pre-training strategy for CNN in order to shorten thtrainingtime.Using the DAE and CNN method of consolidation (DAE-CNN)Volume 11, 72510, 2023 According to H. Alamro et al., AAMD-OELAC can swiftly analyze flexible patterns. A performance evaluation of 26 currently available pretrained CNN algorithms for Android malware detection was presented by Yadav et al. [17]. Based on the results, an EfficientNet-B4 CNN-based method was developed using an image-based malware representation of the Android DEX file in order to identify Android malware. EfficientNet-B4 collected relevant attributes from the malware pictures. Masum and Shahriar [18] developed the Droid-NNet DL structure for malware classification. However, Droid-NNet is a deep learner method that outperforms current state-of-the-art machine learning techniques.PIndroid, a novel permission- and intent-based framework for identifying Android malicious applications, is examined by Idrees et al. [19]. As far as we are aware, the main solution is called Pyndroid, and it makes use of a set of additional rights and purposes.using ensemble methods to accurately detect malware.The authors of [20] demonstrate that after discussing the idea of drift, permissions produce malware detection techniques that are durable and effective. A method for classifying Android malware based on GA and optimizing ensemble learning is presented by Taha and Barukab [21]. In order to

maximize the accuracy of the Android malware classifier, the parameter settings from the RF approach were optimized using the GA. Sabanci et al.'s [22] goal was to use CNN techniques to classify pepper seeds that belonged to different cultivars. The pre-training CNN approaches' features are fused, and feature selection has been applied to the fused features, which are secondary and diverse in the first place. The authors of [23] look at new algorithms used with Android. Malware identification. Consequently, an overview of the Android system revealed the fundamental workings and issues with its security framework.

IV. EXISTING SYSTEM

Many strategies and methods have been implemented by machine learning for Android malware detection. An active machine learning-based approach for Android malware detection has beenoutlinedhere.
Aggregation of Data: Creating a large dataset of

Android apps is the first step towards establishing a malware detection system that uses machine learning. This dataset should contain both benign (non-malicious) initiatives and harmful software. Data can be gathered through a variety of sources that include public repositories, app marketplaces, andvirusanalysisplatforms.

Extraction of features: The features of each application must be extracted after the dataset has been gathered. Resource files, code systems, API calls, requests for permissions made by the program, and more might all be regarded as features. Because it provides data for evaluating and educating machine learning models,

Preprocessing and Labeling: For supervised machine learning, every dataset sample must have a label designating it as benign or malicious. The data is preprocessed using methods including feature selection, scaling, and normalization before being used to train machine learningmodels.

Model Selection: For Android malware detection, a variety of machine learning techniques can be applied, such as but not restricted to: Random Forest Gradient Boosting Machines (GBM) Support Vector Machines (SVM) Deep Learning models (such as Recurrent and Convolutional neural networks)

Instruction and Verification: A training set and a validation set are created from the labeled preprocessed dataset. To adjust hyperparameters and assess performance, the machine learning models are trained on the training set and validated using the validation set. There are several existing systems for Android malware detection that utilize machine learning techniques. These systems typically employ various machine learning algorithms to analyze the characteristics and behaviors of Android applications (APKs) and distinguish between benign and malicious ones. Here are some examples of existing systems and approaches:

DroidAPIMiner: This methodology builds machine learning models for malware detection through the use of API call sequences that are acquired from Android apps. It looks seeking trends in the order in which apps make API calls for the purpose to discover possibly hazardous activity.

Drebin: Drebin is a machine learning system that evaluates apps as benign or dangerous based on metadata taken from Android apps, particularly component features, API calls, and permissions requested.

AndroPyTool: AndroPyTool uses machine learning algorithms coupled with static and dynamic analysis techniques to identify malware for Android devices. It gathers parameters from APKs' Dalvik bytecode and integrates them into machine learning models.

MaMaDroid: The technique retrieves features from Android apps using both static and dynamic analysis. It then organizes apps into divides that are benign or threatening using machine learning techniques like cooperative learning and deep learning.

DroidDet: DroidDet is a machine learning system intended for recognizing families of Android malware. Classifiers for discriminating between various malware families undergo training using features like intent filters, permissions, and API requests.

DeepDroid: DeepDroid analyzes the raw bytecode of Android apps using deep learning models, namely convolutional neural networks (CNNs). It picks up patterns and characteristics straight from the bytecode, making it possible it to recognize malware based on minute temporal variations.

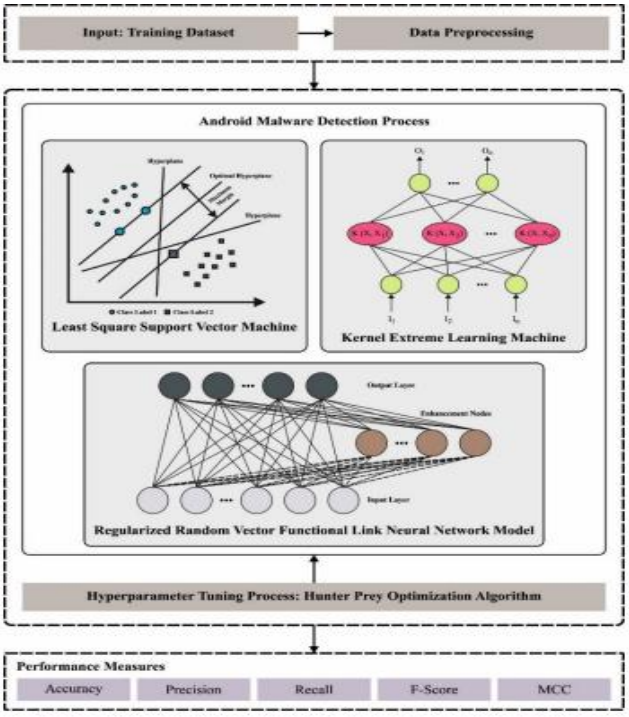
AndroGuard: While not exclusively a machine learning system, AndroGuard is a powerful tool for analyzing Android applications. It provides various features and APIs that can be used in conjunction with machine learning techniques for malware detection.

V. PROPOSED SYSTEM

The design of the AAMDOELAC technique for precise and automated Android malware detection has been the main focus of this study. The AAMD-OELAC technique was designed to automatically identify and classify Android malware. In order to accomplish this, The HPO-based parameter tweaking, ensemble classification, and data preprocessing are all included in the AAMD-OELAC approach.

A. PREPROCESSING OF DATA

Data preparation is done from the beginning to enhance the quality of the real data. Selecting attributes to indicate the class that a new record would pertain to is crucial when it comes to classification [24]. This means that all Android applications can remove the permission and API calls, which are database features.



By deleting the DEX file permissions from each individual APK file, the Androguard utility may be used to analyze APK files. As a result, a data frame consisting of apps (rows) and features (columns); all of the columns indicate a specific permission or binary API call, while the rows indicate either malicious or safe APK files. B. MEMBERNIAL EDUCATION-FUNDED MALWARE CLASSIFICATION Three machine learning models (ML) are used in the ensemble learning process of the AAMD-OELAC technique for Android malware detection: LS-SVM,KELM,andRRVFLN. 1)LS-SVMMODEL

A more advanced version of the SVM technique is called LS-SVM. It also reduces computation expenses and streamlines the computational process [25]. The LS-SVM model, in contrast to the SVM model, trains using a series of linear equations:

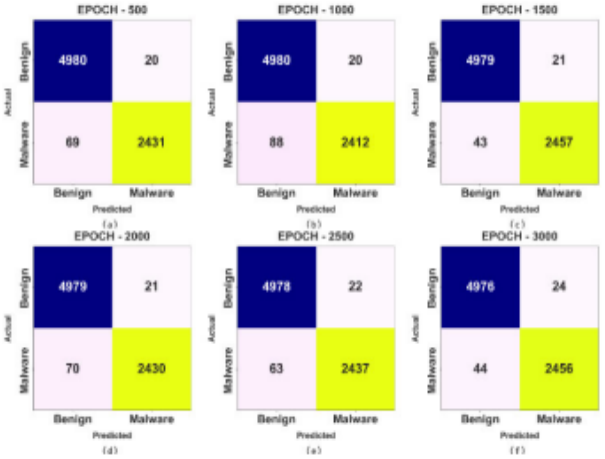
Z equals αT ϕ (x) + b + et (1) Within Equation (1): Z represents the dependent variable, and x

input, E(x) denotes a non-linear mapping function, b displays the bias value, and α represents the weightcoefficient.The function estimation is based on the optimization problem.

Minimize: 1 2 α T α + 1 2 γ X M t=1 e 2 t (2) subjected (to) (f (x)) = α T ϕ (x) + b + et
f (x) = X M t=1 βtK (x, xt) + b

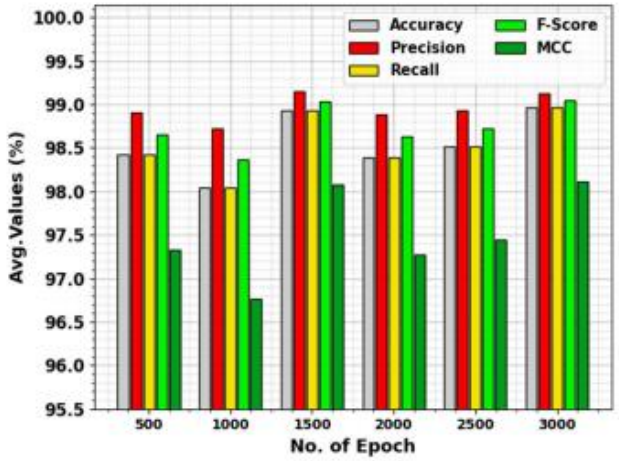
Details of database

Class	No. of Samples
Benign	5000
Malware	2500
Total Samples	7500



Confusion matrices of AAMD-OELAC system (a-f) Epochs 500-3000.

Class	$Accu_y$	$Prec_n$	$Reca_l$	F_{score}	MCC
Epoch - 500					
Benign	99.60	98.63	99.60	99.11	97.33
Malware	97.24	99.18	97.24	98.20	97.33
Average	98.42	98.91	98.42	98.66	97.33
Epoch - 1000					
Benign	99.60	98.26	99.60	98.93	96.76
Malware	96.48	99.18	96.48	97.81	96.76
Average	98.04	98.72	98.04	98.37	96.76
Epoch - 1500					
Benign	99.58	99.14	99.58	99.36	98.08
Malware	98.28	99.15	98.28	98.71	98.08
Average	98.93	99.15	98.93	99.04	98.08
Epoch - 2000					
Benign	99.58	98.61	99.58	99.09	97.27
Malware	97.20	99.14	97.20	98.16	97.27
Average	98.39	98.88	98.39	98.63	97.27
Epoch - 2500					
Benign	99.56	98.75	99.56	99.15	97.45
Malware	97.48	99.11	97.48	98.29	97.45
Average	98.52	98.93	98.52	98.72	97.45
Epoch - 3000					
Benign	99.52	99.12	99.52	99.32	97.96
Malware	98.24	99.03	98.24	98.63	97.96
Average	98.88	99.08	98.88	98.98	97.96
Malware	98.41	99.07	98.41	98.74	98.11
Average	98.97	99.13	98.97	99.05	98.11



VI.METHODOLOGY

DataCollection:Gather a diverse and representative dataset of Android applications, including both malware and benign (non-malicious) samples. You can use sources like VirusTotal, various malware repositories, or create your own dataset.
Extract relevant features from the apps, such as permissions requested, API calls, resource files, manifestdetails,etc.
DataPreprocessing:Clean the dataset by removing duplicates, irrelevant samples, and outliers. Convert the extracted features into a suitable format for machine learning algorithms, such as numericalorbinaryvectors. FeatureSelection:

Use techniques like feature importance, correlation analysis, or dimensionality reduction (e.g., PCA) to select the most informative and discriminative features for training the model. This step helps in reducing computational complexity and improving the model's performance.The process of selecting a machine learning technique or group of algorithms for classification involves selecting a suitable model. Some examples of these include Support Vector Machines (SVM), Random Forest, Gradient Boosting Machines, Neural Networks, and so on.
Think about the trade-offs between accuracy, interpretability, and model complexity.
Instruction:
Divide the preprocessed dataset (e.g., ratio 70-30 or 80-20) into training and testing sets.
Utilizing the training data, train the chosen machine learning model. Avoid overfitting by optimizing hyperparameters using strategies like cross-validation.
To determine how well the model detects malware, examine its performance metrics (such as accuracy, precision, recall, and F1-score) on the testing set.
Following Processing:Utilize post-processing strategies to further improve the model's predictions and lower false positives and negatives, such as threshold tweaking, ensemble methods, or anomaly detection.
Deployment: Combine the trained model with an application or system for Android malware detection.

Put in place the required security measures to safeguard the model and guarantee its integrity while it is being run.
Monitoring and Updating: Using actual data, periodically assess how well the deployed model is doing.
Retrain the model on a regular basis to accommodate new malware samples and keep it updated to reflect growing threats.
input Loop: Over time, improve the system's detection capabilities by incorporating fresh research findings, security reports, and user input.
Work together with researchers and cybersecurity specialists to stay up to date on the newest mitigation techniques and malware trends.

IV.CONCLUSION

In conclusion, developing and launching an Android malware detection system that leverages machine learning techniques has numerous benefits in the fight against the continually shifting mobile security threat landscape. Using this system, we are ableto:
Enhanced Accuracy: By examining massive quantities of data to discover trends and aberrations that can point to the presence of malware, machine learning models can lower false positives and negatives.
Real-time Detection: By using real-time data analysis, the system can straightaway identify and respond to arising malware threats. By doing this, users are certain of prompt security.
Adaptability: Machine learning models are capable of making alterations and improving over time in response to new data and malware

techniques, thereby bolstering the detection system's resistance against future threats.
Intervention: By automating malware detection, less manual intervention is required, enabling ongoing monitoring and detection without a large human resource requirement.
Scalability: Because the system is capable of controlling a large number of Android applications with success, it is suitable for a wider deployment across a range of devices and networks.
It's important to keep in consideration that no detection approach is perfect and that ongoing research and development are necessary to stay up to date with ever-more-devious infection strategies. Researcher, developer, and cybersecurity professional collaboration is still critical to improving the efficacy and stability of machine learning-based Android malware detection systems.

VII.FUTURE WORK

Forthcoming enhancements in machine learning for Android malware detection might involve deep learning improvements specialized for Android app scrutiny, repelling adversarial attacks to reinforce model resilience, and guaranteeing the explicability of AI-generated verdicts for greater confidence. Distributed computing and online learning would enhance real-time detection capabilities, and federated learning and other privacy-preserving strategies might safeguard user data. Runtime behavioral analysis may offer greater insight into the activities of malware.

Extending detection to IoT devices and leveraging mobile edge computing would enhanceAbsolute security. Efforts to make equipment compatible with one another might enhance defensive capabilities, and collaborative techniques would make it easier to share data on threats for precautionary defense tactics. Openness and equal treatment in model deployment are still profoundly ethical considerations.
.

VIII. REFERENCES

[1] H. Rathore, A. Nandanwar, S. K. Sahay, and M. Sewak, ‘‘Adversarial superiority in Android malware detection: Lessons from reinforcement learning based evasion attacks and defenses,’’ Forensic Sci. Int., Digit. Invest., vol. 44, Mar. 2023, Art. no. 301511. [2] H. Wang, W. Zhang, and H. He, ‘‘You are what the permissions told me! Android malware detection based on hybrid tactics,’’ J. Inf. Secur. Appl., vol. 66, May 2022, Art. no. 103159. [3] A. Albakri, F. Alhayan, N. Alturki, S. Ahamed, and S. Shamsudheen, ‘‘Metaheuristics with deep learning model for cybersecurity and Android malware detection and classification,’’ Appl. Sci., vol. 13, no. 4, p. 2172, Feb. 2023. [4] M. Ibrahim, B. Issa, and M. B. Jasser, ‘‘A method for automatic Android malware detection based on static analysis and deep learning,’’ IEEE Access, vol. 10, pp. 117334–117352, 2022. [5] L. Hammood, İ. A. Doğru, and K. Kılıç, ‘‘Machine learning-based adaptive genetic algorithm for Android malware detection in auto-driving vehicles,’’ Appl. Sci., vol. 13, no. 9, p. 5403, Apr. 2023. [6] P. Bhat and K. Dutta, ‘‘A multi-tiered feature selection model for Android malware detection based on feature discrimination and information gain,’’ J. King Saud Univ.-Comput. Inf. Sci., vol. 34, no. 10, pp. 9464–9477, Nov. 2022. [7] D. Wang, T. Chen, Z. Zhang, and N. Zhang, ‘‘A survey of Android malware detection based on deep learning,’’ in Proc. Int. Conf. Mach. Learn. Cyber Secur. Cham, Switzerland: Springer, 2023, pp. 228–242. [8] Y. Zhao, L. Li, H. Wang, H. Cai, T. F. Bissyandé, J. Klein, and J. Grundy, ‘‘On the impact of sample duplication in machine-learning-based Android malware detection,’’ ACM Trans. Softw. Eng. Methodol., vol. 30, no. 3, pp. 1–38, Jul. 2021. [9] E. C. Bayazit, O. K. Sahingoz, and B. Dogan, ‘‘Deep learning based malware detection for Android systems: A comparative analysis,’’ Tehnički vjesnik, vol. 30, no. 3, pp. 787–796, 2023. [10] H.-J. Zhu, W. Gu, L.-M. Wang, Z.-C. Xu, and V. S. Sheng, ‘‘Android malware detection based on multi-head squeeze-and-excitation residual network,’’ Expert Syst. Appl., vol. 212, Feb. 2023, Art. no. 118705.
[11] K. Shaukat, S. Luo, and V. Varadharajan, ‘‘A novel deep learning-based approach for malware detection,’’ Eng. Appl. Artif. Intell., vol. 122, Jun. 2023, Art. no. 106030. [12] J. Geremias, E. K. Viegas, A. O. Santin, A. Britto, and P. Horchulhack, ‘‘Towards multi-view Android malware detection through image-based deep learning,’’ in Proc. Int. Wireless Commun. Mobile Comput. (IWCMC), May 2022, pp. 572–577.13] J. Kim, Y. Ban, E. Ko, H. Cho, and J. H. Yi, ‘‘MAPAS: A practical deep learning-based Android malware detection system,’’ Int. J. Inf. Secur., vol. 21, no. 4, pp. 725–738, Aug. 2022. [14] S. Fallah and A. J. Bidgoly, ‘‘Android malware detection using network traffic based on sequential deep learning models,’’ Softw., Pract. Exper., vol. 52, no. 9, pp. 1987–2004, Sep. 2022. [15] V. Sihag, M. Vardhan, P. Singh, G. Choudhary, and S. Son, ‘‘De-LADY: Deep learning-

based Android malware detection using dynamic features,'' J. Internet Serv. Inf. Secur., vol. 11, no. 2, p. 34, 2021. [16] W. Wang, M. Zhao, and J. Wang, ''Effective Android malware detection with a hybrid model based on deep autoencoder and convolutional neural network,'' J. Ambient Intell. Humanized Comput., vol. 10, no. 8, pp. 3035–3043, Aug. 2019. [17] P. Yadav, N. Menon, V. Ravi, S. Vishvanathan, and T. D. Pham, ''EfficientNet convolutional neural networks-based Android malware detection,'' Comput. Secur., vol. 115, Apr. 2022, Art. no. 102622. [18] M. Masum and H. Shahriar, ''Droid-NNet: Deep learning neural network for Android malware detection,'' in Proc. IEEE Int. Conf. Big Data (Big Data), Dec. 2019, pp. 5789–5793. [19] F. Idrees, M. Rajarajan, M. Conti, T. M. Chen, and Y. Rahulamathavan, ''PIndroid: A novel Android malware detection system using ensemble learning methods,'' Comput. Secur., vol. 68, pp. 36–46, Jul. 2017. [20] A. Guerra-Manzanares, H. Bahsi, and M. Luckner, ''Leveraging the first line of defense: A study on the evolution and usage of Android security permissions for enhanced Android malware detection,'' J. Comput. Virol. Hacking Techn., vol. 19, no. 1, pp. 65–96, Aug. 2022. [21] A. Taha and O. Barukab, ''Android malware classification using optimized ensemble learning based on genetic algorithms,'' Sustainability, vol. 14, no. 21, p. 14406, Nov. 2022. [22] K. Sabanci, M. F. Aslan, E. Ropelewska, and M. F. Unlersen, ''A convolutional neural network-based comparative study for pepper seed classification: Analysis of selected deep features with support vector machine,'' J. Food Process Eng., vol. 45, no. 6, Jun. 2022, Art. no. e13955[23] A. Batouche and H. Jahankhani, ''A comprehensive approach to Android malware detection using machine learning,'' in Information Security Technologies for Controlling Pandemics. USA: Springer, 2021, pp. 171–212. [24] O. N. Elayan and A. M. Mustafa, ''Android malware detection using deep learning,'' Proc. Comput. Sci., vol. 184, pp. 847–852, Jan. 2021. [25] S. S. Sammen, M. Ehteram, Z. Sheikh Khozani, and L. M. Sidek, ''Binary coati optimization algorithm- multi- kernel least square support vector machine-extreme learning machine model (BCOAMKLSSVM-ELM):
[26] M. A. Khan, S. Kadry, P. Parwekar, R. Damaševicius, A. Mehmood, J. A. Khan, and S. R. Naqvi, ''Human gait analysis for osteoarthritis prediction: A framework of deep learning and kernel extreme learning machine,'' Complex Intell. Syst., vol. 2021, pp. 1–19, Jan. 2021.

P

ORIGINALITY REPORT

19%	7%	10%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1 Hayam Alamro, Wafa Mtouaa, Sumayh Aljameel, Ahmed S. Salama, Manar Ahmed Hamza, Aladdin Yahya Othman. "Automated Android Malware Detection Using Optimal Ensemble Learning Approach for Cybersecurity", IEEE Access, 2023 Publication	8%
2 eprints.whiterose.ac.ukInternet Source	5%
3 www.mdpi.comInternet Source	1%
4 Arun M, Ranjana R, Prathipa R, M. PremKumar, G P Dhanalakshmi. "ArduinoBased Robotic Arm for Sludge Cleaning", 2023 9th International Conference on Advanced Computing and Communication Systems (ICACCS), 2023 Publication	1%

5 Saad Sh. Sammen, Mohammad Ehteram,Zohreh Sheikh Khozani, Lariyah Mohd Sidek. 1%

"Binary Coati Optimization Algorithm- Multi-Kernel Least Square Support Vector Machine-Extreme Learning Machine Model (BCOAMKLSSVM-ELM): A New Hybrid Machine Learning Model for Predicting Reservoir Water Level", Water, 2023

Publication

---

6 Submitted to Universiti Teknologi Malaysia Student Paper <1%

---

7 Submitted to De Montfort University Student Paper <1%

---

8 Yash Sharma, Anshul Arora. "A comprehensive review on permissions-based Android malware detection", International Journal of Information Security, 2024 <1%

Publication

---

9 Submitted to Brunel University Student Paper <1%

---

10 Submitted to Northern Arizona University Student Paper <1%

11	Suraj Kumar Prusty, V K Surya, Nijwm Wary. "A High-Speed Charge-Injection based Double Tail Latch for Decision Feedback Equalizer (DFE)", 2023 21st IEEE Interregional NEWCAS Conference (NEWCAS), 2023 Publication	<1%
12	Submitted to University of Malaya Student Paper	<1%
13	link.springer.com Internet Source	<1%
14	Zahraddeen Bala, Fatima Umar Zambuk, Badamasi Ya'u Imam, Abdulsalam Ya'u Gital et al. "Transfer Learning Approach for Malware Images Classification on Android Devices Using Deep Convolutional Neural Network", Procedia Computer Science, 2022 Publication	<1%
15	"Are Your Training Datasets Yet Relevant?", Lecture Notes in Computer Science, 2015. Publication	<1%
16	Advances in Intelligent Systems and Computing, 2016. Publication	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		

REFERENCES

[1] Jayanthi, R. and Florence, L., 2019. Software defect prediction techniques using metrics based on neural network classifiers. Cluster Computing, 22(1), pp.77-88.

[2] Felix, E.A. and Lee, S.P., 2017. Integrated approach to software defect prediction. IEEE Access, 5, pp.21524-21547.

[3] Wang, T., Zhang, Z., Jing, X., Zhang, L.: Multiple kernel ensemble learning for software defect prediction. Autom. Softw. Eng. 23, 569–590 (2015).

[4] Xu, Z., Xuan, J., Liu, J., Cui, X.: MICHAC: defect prediction via feature selection based on maximal information coefficient with hierarchical agglomerative clustering. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Suita, pp. 370–381 (2016).

[5] Ryu, D., Baik, J.: Effective multi-objective naïve Bayes learning for cross- project defect prediction. Appl. Soft Comput. 49, 1062 (2016).

[6] Shan C., Chen B., Hu C., Xue J., Li N.: Software defect prediction model based on LLE and SVM. In: Proceedings of the Communications Security Conference (CSC ’14), pp. 1–5 (2014).

[7] Yang, Z.R.: A novel radial basis function neural network for discriminant analysis. IEEE Trans. Neural Netw. 17(3), 604–612(2006).

[8] K. Han, J.-H. Cao, S.-H. Chen, and W.-W. Liu, “A software reliability prediction method based on software development process,” in Quality, Reliability, Risk, Maintenance, and Safety Engineering (QR2MSE), 2013 International Conference on. IEEE, 2013, pp. 280–283.