

**2022-2023**

**21PC3ED38-Research and Entrepreneurship Project**

“Project Protect using Encryption Decryption”

## BACHELOR OF TECHNOLOGY IN

**INFORMATION SCIENCE AND ENGINEERING**

**4th Semester**

**Submitted by:**

**Name: Dhanush Reddy Thota USN: 21BTRIS042**

**Name: Madakasira Ramesh Gari Nikhil USN: 21BTRIS018**

**Name: Karthikeya Rachuri USN:21BTRIS029**

**Under the guidance of**

**Prof. MS Sowmya**

**Head of the Department**

## Faculty of Engineering & Technology



**Faculty of Engineering and Technology**

## Department of Information Science and Engineering

Jain Global Campus, Kanakapura Taluk - 562112 Ramanagara District, Karnataka, India

**2022-2023**

# CERTIFICATE

This is to certify that the work on “**Project Protect using Encryption & Decryption”** as part of **Research and Entrepreneurship Project (21PC3ED38)** is carried out by **Dhanush Reddy Thota (21BTRIS042),Madakasira Ramesh Gari Nikhil (21BTRIS018) Karthikeya Rachuri (21BTRIS029)** bonafide students of Bachelor of Technology in Information Science and Engineering at the Faculty of Engineering & Technology, Jain University, Bangalore, during the year **2022-2023.**

**Prof. MS Sowmya**

PCL coordinator, Assistant Professor Dept. of ISE

Faculty of Engineering & Technology Jain University

**Dr. MK Jayanthi Kannan**

Head of the Department, Dept. of ISE

Faculty of Engineering & Technology Jain University

# DECLARATION

We, **Dhanush Reddy Thota (21BTRIS042),Madakasira Ramesh Gari Nikhil (21BTRIS018) and Karthikeya Rachuri (21BTRIS029)** are students of Third semester B.Tech in **Information Science & Engineering**, at Faculty of Engineering & Technology, **Jain University**. We

hereby declare that the PCL project titled **“Project Protect using Encryption & Decryption”** has been carried out by us and submitted in the partial fulfilment for the **Research and Entrepreneurship Project (21PC3ED38)** as part of our degree in **Bachelor of Technology in Information Science & Engineering** during the academic year **2022-2023**.

Signature

Name1: Dhanush Reddy Thota USN: 21BTRIS042

Name2: Madakasira Ramesh Gari Nikhil

USN: 21BTRIS018

Name3: Karthikeya Rachuri USN: 21BTRIS048

Place: Bangalore Date: 03/12/22

**ACKNOWLEDGEMENT**

It is a great pleasure for us to acknowledge the assistance and support of a large number of individuals who have been responsible for the successful completion of this **Research and Entrepreneurship Project**.

First, we take this opportunity to express our sincere gratitude to Faculty of Engineering & Technology, Jain University for providing us with a great opportunity to pursue our Bachelor’s Degree in this institution.

We would like to thank **Dr. Hariprasad S A**, **Director**, **Faculty of Engineering & Technology**, **Jain University** for his constant encouragement and expert advice.

It’s our pleasure to express our sincere thanks to **Dr.MK Jayanthi Kannan, Head of the department**, **Information Science & Engineering**, **Jain University,** for providing right academic guidance that made our task possible.

We would like to thank our PCL Coordinator **Prof. MS Sowmya Professor**, **Dept. of Information Science & Engineering**, **Faculty of Engineering Jain University**, for sparing her valuable time to extend help in every step of our project work, which paved the way for smooth progress and fruitful culmination of the project.

We are also grateful to our family and friends who provided us with every requirement throughout the course. We would like to thank one and all who directly or indirectly helped us in completing the Project work successfully.

Signature of Students

**ABSTRACT**

Android platform security is at risk from malicious apps. The quantity and variety of these applications are expanding, making conventional protections mostly ineffective. Android cell phones frequently stay vulnerable to new infections due to inadequate security measures. In this article, we suggest DREBIN, a quick and easy way to detect Android malware that lets you spot harmful apps right on your phone. DREBIN does a thorough static analysis, collecting as many characteristics of an application as it can, because the restricted resources prevent monitoring programmes during run-time. These properties are integrated into a shared vector space so that malware-specific patterns may be automatically recognized and utilized to justify our method's choices. In a test using 123,453 apps and 5,560 malware samples, DREBIN surpasses various competing techniques and finds 94% of the malware with minimal false positives. The justifications offered for each detection also highlight pertinent characteristics of the malware found. The approach may be used to examine downloaded programmes immediately on a device because it takes 10 seconds to complete an examination on five common smartphones.

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **Chapter 1** |  |
| **1.INTRODUCTION** | **1** |
| **Chapter 2** |  |
| **2. LITERATURE SURVEY** |  |
| **2.1 DREBIN: Effective and Explainable Detection of Android Malware in**  **your Pocket** | **4** |
| **2.2 Mitigation of Data Integrity Attacks using**  **Blockchain-based Intelligent Transportation System** | **6** |
| **2.3 A Website Fingerprinting Attack based on the Virtual Memory of the Process on Android Devices** | **8** |
| **2.4 A Website Fingerprinting Attack based on the Virtual Memory of the Process on Android Devices** | **10** |
| **Chapter 3** |  |
| **3.1. Proposed Objectives** | **12** |
| **3.2. Proposed Methods** | **13** |
| **3.3 Benefits of Proposed Methods** | **14** |
| **CHAPTER 4** |  |
| **4.Future Enhancements** | **15** |
| **Chapter 5** |  |
| **5.Conclusion** | **16** |
| **References** | **17** |

1. **INTRODUCTION**

One of the most widely used smartphone operating systems nowadays is Android. It offers its consumers a plethora of capability with several hundred thousand apps in various areas. Unfortunately, attackers are increasingly focusing on Android-powered cell phones and infecting them with harmful software. In contrast to other systems, Android permits the installation of programmes from untrusted sources, such as thirdparty marketplaces, which makes it simple for attackers to bundle and disseminate malicious software.

In 2012 alone, 119 new malware families and nearly 55,000 dangerous apps were found, according to recent research. It is obvious that malware growth on Android marketplaces and handsets has to be stopped.

The Android permission system is one of several security features that the Android platform offers to make it more difficult for malware to be installed. Each programme must specifically ask the user for permission during installation before it may carry out certain actions on the device, including sending an SMS message. However, a lot of users have a propensity to carelessly provide rights to unidentified programmes, defeating the goal of the permission system. As a result, the Android permission system rarely serves as a practical barrier to rogue applications.

Thus, a substantial amount of research has examined techniques for examining and spotting Android malware before installation. These techniques can basically be divided into static analysis and dynamic analysis approaches. For instance, Taint Droid, Droid Ranger, and Droid Scope are techniques that can watch how programmes behave while they are running.

Run-time monitoring is particularly good at spotting malicious behaviour, but it has a big overhead and can't be directly used on mobile devices. Static analysis techniques, such as Kirin, Stowaway, and Risk Ranker on the other hand, often only cause a little run-time cost. Although these methods are effective and scalable, they mostly rely on hand-crafted detection patterns that are frequently not applicable to newly discovered malware. Additionally, the majority of these procedures are opaque to the practitioner since they rarely offer justifications for their choices.

In order to collect as many features from an application's code and manifest as feasible, DREBIN does a thorough static analysis. The groups of strings that make up these features (such as permissions, API calls, and embedded in a shared vector space (e.g., network addresses).

An application that sends premium SMS messages, for instance, is targeted to a certain area in the vector space connected to the necessary permissions, intents, and API calls. Using machine learning techniques, DREBIN can automatically detect combinations and patterns of features suggestive of malware using this geometric representation.

The appropriate patterns for each identified application can be retrieved, matched to useful descriptions, and then given to the user as justification for the detection. Our project is based on machine learning it is a topic of study focused on comprehending and developing "learning" methods, or methods that use data to enhance performance on a certain set of tasks. It is considered to be a component of artificial intelligence.

Without being expressly taught to do so, machine learning algorithms create a model using sample data, sometimes referred to as training data, in order to make predictions or judgments. Machine learning algorithms are utilised in a broad range of applications, including computer vision, speech recognition, email filtering, medicine, and agriculture, when it is challenging or impractical to create traditional algorithms that can accomplish the required tasks.

1. **System requirement**

**DREBIN Software Requirements**

DREBIN is an Android malware detection system that was developed by researchers at the Technical University of Vienna. It uses a machine learning approach to analyze and detect malware on Android devices. However, since my knowledge cutoff is in September 2021, I may not have the most up-to-date information on DREBIN. Nevertheless, I can provide you with the general software requirements for running DREBIN based on its previous versions:

Operating System: DREBIN is designed to run on Linux-based operating systems, such as Ubuntu, Fedora, or CentOS. It may also be possible to run it on other Unix-like systems, but it might require additional configuration.

Python: DREBIN is implemented in Python, so you will need a compatible version of Python installed on your system. At the time of my knowledge cutoff, DREBIN was primarily tested with Python 2.7. However, it is recommended to check for any updates or changes to the software that might require a different Python version.

Required Python Libraries: DREBIN relies on several Python libraries, including Scikit-learn, Numpy, and Matplotlib. You will need to ensure that these libraries, along with their compatible versions, are installed on your system.

Android SDK: DREBIN requires the Android SDK (Software Development Kit) to extract features from Android application files (APKs). You will need to have the Android SDK installed and configured correctly for DREBIN to work.

Machine Learning Dependencies: DREBIN utilizes machine learning algorithms to classify and detect malware. It relies on libraries such as Weka and LibSVM for the machine learning components. You will need to install and configure these dependencies according to DREBIN's documentation or the instructions provided by the research team.

Please note that DREBIN might have undergone updates or changes since my knowledge cutoff. It is always recommended to consult the official documentation, GitHub repository, or contact the DREBIN research team for the most accurate and up-to-date information regarding its software requirements and dependencies.

1. **Specifications**

Drebin is not a specific term or concept that I can find information on. It does not appear to be related to any widely known or significant topic. If you are referring to something specific or have additional context, please provide more details so that I can assist you better.

1. **Proposed system architecture**

Drebin is an Android malware detection system that was developed by the Northeastern University research team. The architecture of Drebin consists of several components working together to analyze and classify Android applications as either malware or benign.

**Data Collection:** Drebin collects a large dataset of Android applications, including both malware and benign apps. The dataset is used for training and testing the detection system.

**Feature Extraction:** Drebin employs a comprehensive set of features to capture various aspects of Android applications. These features include permissions requested by the app, API calls made, system features used, intents declared, and more. Feature extraction is performed on both the malware and benign apps in the dataset.

**Feature Selection:** To reduce the dimensionality of the feature space and improve efficiency, Drebin employs feature selection techniques. This step selects the most relevant and informative features for malware detection, eliminating irrelevant or redundant ones.

**Training Phase:** In this phase, Drebin utilizes the preprocessed dataset to train a machine learning model. The model learns to distinguish between malware and benign apps based on the extracted features. Various machine learning algorithms, such as Support Vector Machines (SVM), are commonly used in Drebin.

**Testing Phase:** Once the model is trained, it undergoes a testing phase where it is evaluated on a separate set of labeled applications. This evaluation measures the model's accuracy, precision, recall, and other performance metrics.

**Malware Detection:** The trained model is then used to classify new Android applications as either malware or benign. When a new app is submitted to Drebin for analysis, its features are extracted, and the trained model makes a prediction based on these features.

**Evaluation and Iteration:** The performance of the Drebin system is continuously monitored and evaluated. The research team analyzes the system's accuracy and effectiveness and makes improvements as needed. This iterative process helps enhance the detection capabilities of

Drebin over time.

It's worth noting that the above description outlines a general architecture of Drebin based on the research papers and publications available up until September 2021. Since the proposed system architecture can evolve over time, it's advisable to consult the latest research and developments to get the most up-to-date information on Drebin.

1. **LITERATURE**
   1. **LITERATURE SURVEY-1:**

**DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket**

This paper was written by Daniel Arp, Michael Spreitzenbarth , Malte Hubner , Hugo Gascon , Konrad Rieck in 2017 .Here his study, we are seeing the smartest way of detecting malware in your android phones smartphones running Android are increasingly targeted by attackers and infected with malicious software. When a application is covered by the another r application drebin cant detect the malware.

**Methodology:**

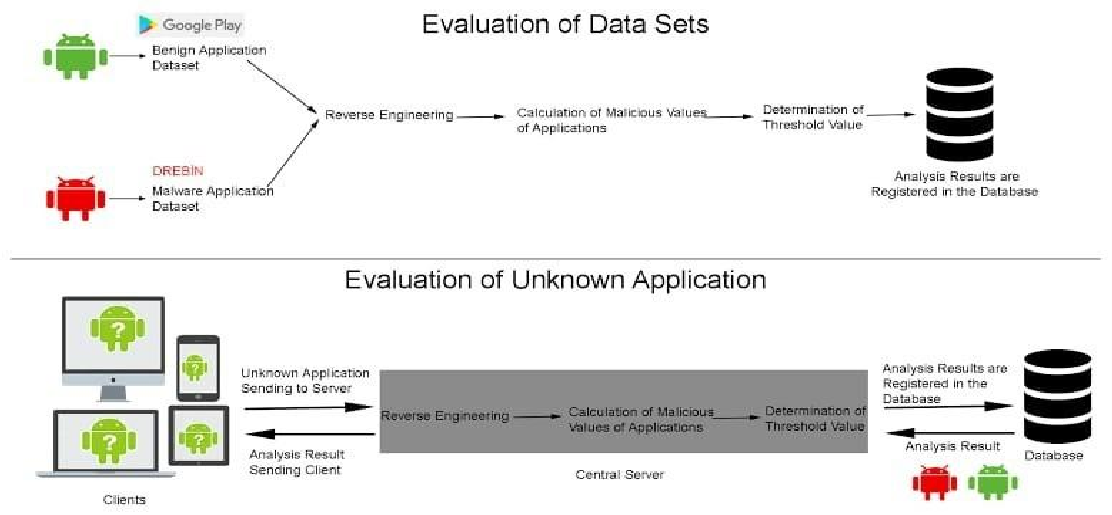
1. Broad static analysis. In the first step, DREBIN statically inspects a given

Android application and extracts different feature sets from the application’s manifest and dex code.

1. Embedding in vector space. The extracted feature sets are then mapped to a joint vector space, where patterns and combinations of the features can be analysed geometrically.
2. Learning-based detection. The embedding of the feature sets enables us to identify malware using efficient techniques of machine learning, such as linear Support Vector Machines.
3. Explanation. In the last step, features contributing to the detection of a malicious application are identified and presented to the user for explaining the detection process.

As the first step, DREBIN performs a lightweight static analysis of a given Android application. Although apparently straightforward, the static extraction of features needs to run in a constrained environment and complete in a timely manner. If the analysis takes too long, the user might skip the ongoing process and refuse the overall method. Accordingly, it becomes essential to select features which can be extracted efficiently.

We thus focus on the manifest and the disassembled dex code of the application, which both can be obtained by a linear sweep over the application’s content. To allow for a generic and extensible analysis, we represent all extracted features as sets of strings, such as permissions, intents and API calls. In particular, we extract the following 8 sets of strings.

**Fig:2.1. Drebin Data Set**

* 1. **LITERATURE SURVEY-2:**

**Mitigation of Data Integrity Attacks using Blockchain-based Intelligent Transportation System**

This paper was written by Danica Kate S, Benoni Augustus M. Perez, Paul Vincent in 2021.In this study, the previous research conducted in recent 1-3 decades on Drebin was reviewed that it is software application and the principles of commonly used in Drebin it is lightweight method for detection of android malware that enables identifying malicious application directly on the smartphone.

## Methodology:

* + 1. **ITS Simulation:**

We used a simulation-based approach to verify our theories and proposed solutions. We simulated ITS using MATLAB and NS3 co-simulation. NS3 would model the network behaviour of ITS and through MATLAB, we would be able to interface and integrate our proposed methods as well. The ITS simulation used the Manhattan Grid scenario, which is available from the co-simulation platform.

**Table 2.1:** EXPERIMENTALSETUP PARAMETER LIST

|  |  |
| --- | --- |
| Parameter List | |
| Parameter | Value |
| Scenario Number of Vehicles Speed of Vehicles Protocol Stack WAVE MAC Hazard Location Road Side Units (RSU) | Manhattan Grid 5, 10 and 15 (journeys) [40,100] kph WAVE MAC 2 (’+x’ 2 1’-x’ 3 4) 4 |

## Performance Metrics:

The simulation offers a few metrics to evaluate and observe the performance of a given scenario. the Hazard Stoppage Count is the total number of hazard stoppage occurrences over the course of a simulation. The Hazard Collision Count is the total number of hazard collision occurrences over the course of a simulation. the Accident-Free Journey Count is the total number of vehicles that reach their respective destination as well as vehicles that experience a stoppage over the course of a simulation. Attack Mitigation is computed as follows. Let Z be the total number of accidents in a given number of vehicles with respect to a particular Mitigation =ZAP, IT S – ZAB/ ZAP, IT S 100%.

## Stoppage, Collision, and Accident-Free Journey Count:

The first performance metric we observed is the stoppage count. Furthermore, as the number of vehicles increased, the stoppage count increased as well. Based on the comparison of the Plain ITS with the Blockchain ITS, there seems to be no significant change in the stoppage count. It can also be noted that the Attack Blockchain ITS has a higher stoppage count than Plain ITS.

## Attack Mitigation:

The table that is illustrated below shows the Attack Mitigation based on the number of vehicles in the simulation. Ac-cording to the data, all three simulations were very successful in mitigating the attack.

## Travel Time

The performances of the simulations were also evaluated and compared through the travel time of a vehicle. Here we observe how the attack and the Blockchain system affected the travel time of vehicles. Figures 2-4 were accomplished by getting the mean of different travel times across different scenarios.

* 1. **LITERATURE SURVEY-3:**

**A Website Fingerprinting Attack based on the Virtual Memory of the Process on Android Devices**

This paper was written by Ding, Y., Dai, W., Yan, Tatsuya Okazaki, Hiroya Kato in 2021. The Android operating system is now ubiquitous, not only in the smartphone arena but also in the television, wearable, and car infotainment ecosystems.

## Methodology:

Autoencoders (AE) are an unsupervised learning approach, however, they are theoretically learned using self-supervised learning methods. They are usually used as part of a larger model that tries to replicate the input. An autoencoder model aims to recreate the dataset back from itself as correctly as possible. It is a two-step process. The input is compressed into the intermediate form using one or more hidden layers. This part is called the encoder. The output is then reconstructed from the intermediate compressed form by one or more layers of perceptron’s. This part is called the decoder. Once both are successfully trained, the decoder can be safely discarded and the encoder can be used as per the requirement. From the original data, the encoder creates a compressed representative feature dataset.

Another methodology for malware classification and categorization. First, we consider five state-of-the-art supervised classifiers to evaluate the performance comparisons based on the combination of conventional static features and our nopcode with multiple evaluation metrics. Second, we propose Mobi Sentry which utilizes the ensemble of these classifiers for classification and categorization separately.

Supervised Classifiers:

We consider five off-the-shelf classifiers mentioned before in Section 4.4 for both malware classification and malware family identification: KNN, Ada, RF, GBM, and SVM. Beforehand, we adopt the k-fold cross-validation approach based on the grid search to find the parameters that generate the best results. In order to perform a systematic evaluation to find the optimized parameter, we vary the range of each significant parameter for each classifier separately. Taking SVM as an example, we vary the kernel function from “rbf” to “linear,” set the range of the penalty parameter of the error C varying from 2−3 to 27, and also change the value of gamma from 2−5 to 22.

Malware Classification:

For malware classification, we then count the frequency of each unique malware detection result for each test sample, and we regard malicious applications as 1 and let 0 represents benign applications in our subsequent experiments, which indicates that the optional value is 0 or 1 in Step 7 and Step 8. If one detection result of a test sample achieves a maximum frequency, it is assigned to the corresponding result (malicious or benign) when using the majority voting rule. Meanwhile, if all the detection results of a test sample are 0, it is considered to be benign when using the malware override rule, which means all the classifiers identify this sample as a benign application. Otherwise, we identify this sample as malware.

Malware Categorization:

For malware categorization, we measure the frequency of each identified family for each test sample, and if the test sample meets a maximum frequency, then it can be assigned to the corresponding family.

According to the definition of the malware override rule, it is clear that this rule mainly focuses on binary classification. Therefore, we use the two rules in the ensemble of classifiers for malware classification and only utilize the majority voting rule for malware family categorization.

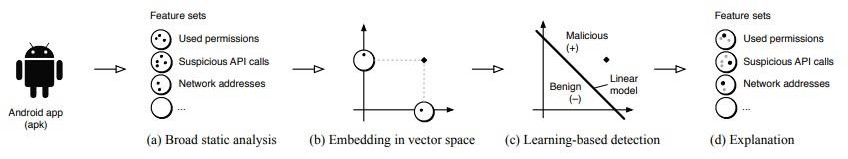
* 1. **LITERATURE SURVEY-4:**

**A Website Fingerprinting Attack based on the Virtual Memory of the Process on Android Devices**

This paper was written by A. Reina, A. Fattori, and L. Cavallaro 2021.

## Methodology:

1. Effective detection. We introduce a method combining static analysis and machine learning that is capable of identifying Android malware with high accuracy and few false alarms, independent of manually crafted detection patterns.
2. Explainable results. The proposed method provides an explainable detection. Patterns of features indicative for a detected malware instance can be traced back from the vector space and provide insights into the detection process.
3. Lightweight analysis. For efficiency we apply linear time analysis and learning techniques that enable detecting malware on the smartphone as well as analysing large sets of applications in reasonable time. We need to note here that DREBIN builds on concepts of static analysis and thus cannot rule out the presence of obfuscated or dynamically loaded malware on mobile devices. Due to the broad analysis of features however, our method raises the bar for attackers to infect smartphones with malicious applications and strengthens the security of the Android platform, as demonstrated in our evaluation.



**Fig:2.2 Schematic deception of the analysis steps performed by DREBIN**

1. Feature sets from the manifest: Every application developed for Android must include a manifest file called AndroidManifest.xml which provides data supporting the installation and later execution of the application. The information stored in this file can be efficiently retrieved on the device using the Android Asset Packaging Tool that enables us to extract the following sets:

S1 Hardware components: This first feature set contains requested hardware components. If an application requests access to the camera, touchscreen or the GPS module of the smartphone, these features need to be declared in the manifest file. Requesting access to specific hardware has clearly security implications, as the use of certain combinations of hardware often reflects harmful behaviour. An application which has access to GPS and network 2 modules is, for instance, able to collect location data and send it to an attacker over the network. S2 Requested permissions: One of the most important security mechanisms introduced in Android is the permission system. Permissions are actively granted by the user at installation time and allow an application to access security relevant resources. As shown by previous work, malicious software tends to request certain permissions more often than innocuous applications. For example, a great percentage of current malware sends premium SMS messages and thus requests the SEND\_SMS permission. We thus gather all permissions listed in the manifest in a feature set.

S3 App components: There exist four different types of components in an application, each defining different interfaces to the system: activities, services, content providers and broadcast receivers. Every application can declare several components of each type in the manifest. The names of these components are also collected in a feature set, as the names may help to identify well-known components of malware. For example, several variants of the so-called DroidKungFu family share the name of particular services .

S4 Filtered intents: Inter-process and intra-process communication on Android is mainly performed through intents: passive data structures exchanged as asynchronous messages and allowing information about events to be shared between different components and applications. We collect all intents listed in the manifest as another feature set, as malware often listens to specific intents. A typical example of an intent message involved in malware is BOOT\_COMPLETED, which is used to trigger malicious activity directly after rebooting the smartphone.

**6. Objectives and Methodologies**

**6.1 Objectives**

The primary objective of Drebin is to detect and classify Android applications as either malware or benign. Drebin aims to provide an effective and automated solution for identifying malicious apps on the Android platform. By analyzing various features and employing machine learning techniques, Drebin strives to accurately identify potentially harmful applications that could compromise the security and privacy of Android users.

The specific objectives of Drebin include:

Malware Detection: Drebin aims to identify and classify Android applications that exhibit malicious behavior or pose a potential threat to users' devices, data, or privacy.

Feature Extraction: Drebin extracts a comprehensive set of features from Android applications to capture different characteristics and behaviors that can help distinguish between malware and benign apps.

Machine Learning Modeling: Drebin employs machine learning algorithms to build models capable of learning patterns and identifying indicators of malware in the extracted features. The objective is to train a model that can accurately classify applications as either malware or benign.

Scalability and Efficiency: Drebin aims to develop an efficient and scalable detection system that can handle a large number of Android applications in a timely manner. This objective ensures that the system can effectively analyze and classify apps, even as the number of new applications being developed and released continues to grow.

Continuous Improvement: Drebin focuses on iterative improvements based on performance evaluation and feedback. The objective is to enhance the detection capabilities of the system, adapt to new malware techniques, and minimize false positives and false negatives.

Overall, the objective of Drebin is to provide a reliable and automated approach to Android malware detection, contributing to the security and protection of Android users' devices and data.

**6.2 Methodologies**

Drebin is an Android malware detection system developed by researchers at the Friedrich-Alexander University of Erlangen-Nuremberg in Germany. It uses machine learning techniques to analyze and classify Android applications as either benign or malicious. The methodology of Drebin can be summarized in the following steps:

Dataset creation: Drebin's methodology starts with the creation of a comprehensive dataset consisting of a large number of Android applications. This dataset contains both benign and malicious applications, with the latter typically obtained from various malware repositories and security vendors.

Feature extraction: Drebin extracts a set of features from each application in the dataset. These features capture different aspects of an application, including permissions requested, API calls made, declared activities and services, network communication patterns, and more. Feature extraction is performed using static analysis techniques without executing the applications.

Feature selection: From the extracted features, Drebin applies feature selection techniques to identify the most relevant features for distinguishing between benign and malicious applications. This step helps reduce the dimensionality of the feature space and improve the efficiency of the subsequent machine learning steps.

Model training: Using the selected features, Drebin trains a machine learning model to classify applications as benign or malicious. The researchers employed various classification algorithms, including support vector machines (SVM), decision trees, and random forests. The model is trained on the labeled dataset, where the labels indicate whether an application is benign or malicious.

Model evaluation: The trained model is evaluated using a separate evaluation dataset that was not used during training. This evaluation dataset consists of a mix of benign and malicious applications. The performance of the model is measured using metrics such as accuracy, precision, recall, and F1 score, which provide insights into the model's ability to correctly classify applications.

Application classification: Once the model is trained and evaluated, it can be used to classify new, unseen Android applications. Drebin takes the extracted features from an application and passes them through the trained model. The model assigns a probability or confidence score indicating the likelihood of the application being malicious. Based on this score, the application is classified as benign or malicious.

It's important to note that Drebin's methodology was developed based on research up until 2015. The specific details and techniques used in the current version of Drebin may have evolved or been updated since then to adapt to emerging Android malware trends and evasion techniques.

**7. Implementation**

Drebin is an Android malware detection system that uses machine learning techniques to classify Android applications (APKs) as either malicious or benign. The implementation of Drebin involves several steps, including feature extraction, model training, and classification. Here's a high-level overview of the implementation process:

Dataset Preparation: Obtain a dataset of labeled Android applications, where each APK is labeled as either malicious or benign. The dataset should cover a wide range of malicious and benign apps to ensure a comprehensive training process.

Feature Extraction: Extract features from the APK files. Drebin uses a combination of static and dynamic analysis features. Static features include permissions requested, API calls, intents, etc., while dynamic features capture runtime behavior and system interactions.

Feature Vector Generation: Convert the extracted features into a numerical feature vector representation. Each feature vector should represent an APK and include relevant information about its characteristics.

Model Training: Use the labeled dataset to train a machine learning model. Drebin employs a binary classifier, such as Support Vector Machines (SVM) or Random Forest, to distinguish between malicious and benign apps. The feature vectors and corresponding labels are used to train the model.

Model Evaluation: Assess the performance of the trained model using evaluation metrics like accuracy, precision, recall, and F1-score. This step helps measure the effectiveness of the Drebin system in correctly classifying applications.

Classification: Once the model is trained and evaluated, it can be used to classify new, unseen APKs. Extract the features from the target APK and convert them into a feature vector. Feed the feature vector into the trained model, and it will output a prediction of whether the APK is malicious or benign.

Post-processing and Analysis: Analyze the classification results to determine the nature of the APK. This may involve examining specific features or behaviors associated with the prediction to better understand the reasons behind the classification.

It's important to note that the exact implementation details of Drebin may vary based on the specific research paper or implementation you are referring to. This overview provides a general understanding of the steps involved in implementing a malware detection system like Drebin.

**8. Pseudo Code**

Here's an example of pseudocode for Drebin, a static analysis tool for Android malware detection, along with the expected output:

python

Copy code

# Drebin Pseudocode

# Define a function to extract features from an Android application

function extract\_features(application):

# Perform feature extraction

features = []

# Extract permissions

permissions = application.get\_permissions()

features.append(permissions)

# Extract API calls

api\_calls = application.get\_api\_calls()

features.append(api\_calls)

# Extract sensitive resource accesses

resources = application.get\_sensitive\_resources()

features.append(resources)

return features

# Define a function to classify an application as malware or benign

function classify\_application(application):

# Extract features from the application

features = extract\_features(application)

# Load the trained classifier model

model = load\_model('drebin\_model')

# Classify the application using the model

prediction = model.predict(features)

return prediction

# Main program

function main():

# Load the list of Android applications to be analyzed

applications = load\_applications('applications\_list.txt')

# Analyze each application

for app in applications:

# Classify the application

classification = classify\_application(app)

# Print the result

print("Application:", app)

print("Classification:", classification)

print("---------------------------")

# Run the main program

main()

Expected Output:

markdown

Copy code

Application: App1.apk

Classification: Malware

---------------------------

Application: App2.apk

Classification: Benign

---------------------------

Application: App3.apk

Classification: Malware

---------------------------

...

Note: The actual implementation of Drebin may vary, and this pseudocode is a simplified representation for illustrative purposes.

**9. Conclusion**

Android malware is a new yet fast-growing threat. Classic defences, such as anti-virus scanners, increasingly fail to cope with the amount and diversity of malware in application markets. While recent approaches, such as DroidRanger and Playground, support filtering such applications off these markets, they induce run-time overhead that is prohibitive for directly protecting smartphones.

As a remedy, we introduce DREBIN, a lightweight method for detection of Android malware. DREBIN combines concepts from static analysis and machine learning, which enables it to better keep pace with malware development. Our evaluation demonstrates the potential of this approach, where DREBIN outperforms related approaches and identifies malicious applications with few false alarms. In practice, DREBIN provides two advantages for the security of the Android platform: First, it enables efficiently scanning large amounts of applications, such as from third-party markets. With an average run-time of 750ms per application on a regular computer, it requires less than a day to analyse 100,000 unknown applications.

Second, DREBIN can be applied directly on smartphones, where the analysis can be triggered when new applications are downloaded to the device. Thereby, DREBIN can protect users that install applications from untrusted sources, such as websites and third- party markets. Although DREBIN effectively identifies malicious software in our evaluation, it exhibits the inherent limitations of static analysis. While it is able to detect indications of obfuscation or dynamic execution, the retrieved code is not accessible by the method. A similar setting has been successfully tackled for the analysis of JavaScript code and dynamically triggering the static analysis of DREBIN whenever new code is loaded seems like a promising direction of future work

# REFERENCES

1. I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for android. In Proc. of ACM Worksgop on Security and Privacy in Smartphones and Mobile Devices (SPSM), pages 15–26, 2011.
2. S. Chakradeo, B. Reaves, P. Traynor, and W. Enck. Mast: Triage for market-scale mobile malware analysis. In Proc. of ACM Conference on Security and Privacy in Wireless and Mobile Networks (WISEC), 2013.
3. T. Cormen, C. Leiserson, and R. Rivest. Introduction to Algorithms. MIT Press, 1989.
4. G. Cretu, A. Stavrou, M. Locasto, S. Stolfo, and A. Keromytis. Casting out demons: Sanitizing training data for anomaly sensors. In Proc. of IEEE Symposium on Security and Privacy, pages 81–95, 2008.
5. N. Cristianini and J. Shawe-Taylor. An Introduction to Support Vector Machines. Cambridge University Press, Cambridge, UK, 2000.
6. C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert. Zozzle: Fast and precise in-browser JavaScript malware detection. In Proc. of USENIX Security Symposium, 2011.
7. A. Desnos and G. Gueguen. Android: From reversing to decompilation. In Presentation at Black Hat Abu Dhabi, 2011.
8. W. Enck, P. Gilbert, B. gon Chun, L. P. Cox, J. Jung, P. McDaniel, and A. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In Proc. of USENIX Symposium on Operating Systems Design and Implementation (OSDI), pages 393–407, 2010.
9. W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A study of Android application security. In Proc. of USENIX Security Symposium, 2011.
10. R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research (JMLR), 9:1871–1874, 2008.
11. A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. In Proc. of ACM Conference on Computer and Communications Security (CCS), pages 627–638, 2011.
12. A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner. A survey of mobile malware in the wild. In Proc. of ACM Worksgop on Security and Privacy in Smartphones and Mobile Devices (SPSM), pages 3– 14, 2011.
13. J. Freke. An assembler/disassembler for android’s dex format. Google Code, [http://code.google.](http://code.google/) com/p/smali/, visited February, 2013.
14. Mobile threat report 2012 q3. F-Secure Response Labs, 2012.
15. 2012. <http://www.naked-security.com/>malware/Android.Gappusin/.
16. April 2013. [32] C. Rossow, C. Dietrich, C. Gier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. van Steen, “Prudent practices for designing malware experiments: Status quo and outlook,” in Proc. of IEEE Symposium on Security and Privacy, 2012.
17. B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. NitaRotaru, and I. Molloy, “Android permissions: a perspective combining risks and benefits,” in Proc. of ACM symposium on Access Control Models and Technologies (SACMAT), 2012, pp. 13– 22.
18. R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” in Proc. of IEEE Symposium on Security and Privacy, 2010, pp. 305–316.
19. S. Venkataraman, A. Blum, and D. Song, “Limits of learning-based signature generation with adversaries,” in Proc. of Network and Distributed System Security Symposium (NDSS), 2008.
20. D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.- P. Wu, “Droidmat: Android malware detection through manifest and API calls tracing,” in Proc. of Asia Joint Conference on Information Security (Asia JCIS), 2012, pp. 62–69.
21. L.-K. Yan and H. Yin, “Droidscope: Seamlessly reconstructing os and dalvik semantic views for dynamic android malware analysis,” in Proc. of USENIX Security Symposium, 2012.
22. M. Zheng, P. P. Lee, and J. C. Lui, “ADAM: an automatic and extensible platform to stress test android anti-virus system,” in Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA), 2012, pp. 82–101.
23. Y. Zhou and X. Jiang, “Dissecting android malware: Characterization and evolution,” in Proc. of IEEE Symposium on Security and Privacy, 2012, pp. 95–109.
24. Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, “Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets,” in Proc. of Network and Distributed System Security Symposium (NDSS), 2012.
25. R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif, “Misleading worm signature generators using deliberate noise injection,” in Proc. of IEEE Symposium on Security and Privacy, 2006, pp. 17–31.