Android Security: Genetic Algorithm-Based Malware Detection System

Divya Kapil
School of Computing
Graphic Era Hill University
Dehradun, India
divya.k.rksh@gmail.com

Varsha Mittal
Computer Science and Engineering
Graphic Era Deemed to be University
Dehradun, India
Var.aadi@gmail.com

Atika Gupta
School of Computing
Graphic Era Hill University
Dehradun, India
atika04591@gmail.com

Mayank Kumar Bamaniya
School of Computing
Graphic Era Hill University
Dehradun, India
mayankbamaniya775@gmail.com

***Abstract*—The expansion of Android devices in today's digital landscape highlights the critical demand for robust security actions to counteract the escalating threat of malware. The influence of Android malware extends beyond smartphones. While Android is mainly utilized on smartphone devices presently, its reach will expand to include Internet of Things (IoT) devices. Notably, an Android-based operating system tailored for IoT, initially named "Android Things" and later rebranded as "Brillo," has already been presented. As an outcome, Android malware will increasingly impact a broader scope of devices beyond just smartphones. This research paper aims to provide a comprehensive learning of the Android architecture, and the general landscape of malware threats and proposes a genetic algorithm-based security framework for malware detection. We employed the random forest and ANN classifiers and got 91.441 % and 97.206 % accuracy respectively. We expect that this research will inspire researchers to work in this direction.**

***Index Terms*—Android, Android Architecture and Threat Model, Machine Learning**

I. Introduction

In the rapidly developing landscape of mobile technology, Android has emerged as a prevailing force, powering a vast array of smartphones, tablets, and other smart devices. Invented by Google, the Android operating system has not only revolutionized the way we interact with our devices but has also paved the way for a successful ecosystem of applications and services. Android, presented in 2008, has developed exponentially to become the most widely used mobile operating system globally. Its open-source nature, coupled with a dynamic community of developers, has fueled invention and permitted the customization of user experiences. As of now, Android's market share reflects its pervasiveness, as it powers a myriad range of devices, from flagship smartphones to budget-friendly options and even smart TVs and wearables.

Malware in Android has become a dominant and evolving threat in the ever-expanding landscape of mobile technology. Android's open nature, while encouraging innovation and manifold application development, has also developed opportunities for malicious actors to exploit vulnerabilities. Malware attacks pose a significant threat to the Android operating system, characterized by the presence of malicious applications containing harmful code. These applications are created with malicious purposes, aiming to achieve unauthorized access and perform actions that compromise the security principles of confidentiality, integrity, and availability.

Static, dynamic, and hybrid approaches are commonly used in the field of Android malware detection [1]. Android malware can penetrate devices through various channels, exploiting both technical vulnerabilities. Common distribution methods include App Markets: Malicious apps may be hidden as legitimate ones and spread through third-party app stores or even Google Play Store, bypassing initial security inspections. Web Browsing: Malicious websites or compromised legitimate sites can host malware, which can be inadvertently downloaded when users see these pages. Emails and Messages: Phishing emails or messages may have links to malicious websites or attachments that, when unlocked, install malware on the device. Networks: Malware can be transmitted through compromised Wi-Fi networks, especially in public spaces, where users connect to unsecured networks. Bluetooth and NFC: Malware can be sent through Bluetooth or Near Field Communication (NFC) connections if devices are not adequately secured.

The existence of malware on Android devices can have severe consequences for both users and their devices. (a) Privacy Breaches: Malware often strives to steal sensitive information, including personal data, login credentials, and monetary information, leading to privacy breaches.

(b) Financial Loss: Specific types of malware, such as ran-

somware, can extort money from users by encrypting their files or terrorizing to publish sensitive information.
(c) Performance Degradation: Malware can consume device resources, resulting in slower performance, decreased battery life, and raised data usage.
(d) Unauthorized Access: Backdoors and other forms of malware can deliver attackers with unauthorized access to the device, permitting them to control it remotely.
(e) Data Loss: Malware can delete, alter, or exfiltrate data from the device, leading to conceivable data loss for the user. In this paper, we discuss the Android Architecture and, the role of Machine Learning in Android security. We propose a genetic algorithm-based malware detection system for Android. Our key contributions are shown below:

- To present the Android architecture with the discussion of its components and present android threat model
- To propose genetic algorithm based malware detection in android.

The research paper is arranged into 5 Sections. The related work is discussed in Section 2 and section 3 presents the android architecture and threat model. In section 4, we present the importance of machine learning in android security and in section 5, we present genetic algorithm base malware detection system , and eventually, we conclude this paper in section 7.

II. Related Work

The Android operating system stands as the most dominant mobile OS, giving rise to a surfeit of third-party Android application (app) markets. The lack of regulatory standards in these markets has produced research efforts toward developing malware detection techniques. However, improvements in malware sophistication and Android system evolution pose challenges in sustaining long-term efficacy in detection methods. Moreover, augmenting feature sets can heighten model complexity and computational overheads. Permissions occur as pivotal components in Android app security. Term Frequency—Inverse Document Frequency (TF-IDF) serves to evaluate word importance within a file set in a corpus. Static analysis methods obviate the demand for app execution, efficiently pulling permissions from apps. Yuan et al. [2] proposed a novel static detection method integrating TF-IDF and Machine Learning. System permissions are extracted from Android application package manifest files. TF-IDF calculates the Permission Value (PV) and Sensitivity Value of APK (SVOA) for each app. Machine learning then leverages SVOA and the number of used permissions for learning and testing. Evaluation involves 6070 benign apps and 9419 malware samples. Results indicate that relying solely on dangerous permissions or their quantity inadequately distinguishes between malicious and benign apps. The proposed approach achieves up to 99.5% accuracy in malware detection, with a learning and training time of just 0.05s. For malware family detection, accuracy reaches 99.6%. Notably, the method attains a 92.71% accuracy in detecting unknown/new samples. Comparative analysis against state-of-the-art approaches demonstrates the superiority of the proposed method in detecting both malware and malware families. Zhao et al. [3] introduced AntiMalDroid for Android malware detection, employing behavior sequences as features. Enck et al. [4] devised the TaintDroid technique, which identifies features such as variable, method, and file tracking utilized by malware. Burguera et al. [5] proposed Crowdroid, a clustering technique aimed at detecting malware. Dini et al. [6] presented MADAM, utilizing dynamic approaches for detection at both kernel and user levels. Wu et al. [7] introduced the DroidDolphin approach, which involves creating log files and extracting information to safeguard against malware. Additionally, a dynamic system call-centric analysis and stimulation technique named CopperDroid is proposed in [8], leveraging AVM-based methods.

III. Android Architecture and Android Threat Model

Understanding the architecture of the Android operating system is fundamental to comprehending its robust capabilities and functionalities. Android's architecture is designed with a layered approach, each layer contributing to the overall functionality of the system.

A. Android Architecture

The key components of the Android architecture include:

(i) Linux Kernel: At the heart of Android is the Linux kernel, delivering the foundational infrastructure for the operating system. This layer handles essential hardware resources, such as memory, device drivers, and the file system, ensuring seamless communication between the hardware and the higher layers of the system.

(ii) Libraries: The libraries layer consists of a set of libraries written in C and C++ that provide elementary functions and services to the Android system. These libraries cover different domains, including graphics rendering, database management, and network connectivity, enhancing the overall efficiency of the operating system.

(iii) Android Runtime: The Android Runtime (ART) is accountable for executing and handling application code. In earlier versions, the Dalvik Virtual Machine was utilized, but ART introduced ahead-of-time (AOT) compilation, resulting in enhanced performance and efficiency.

(iv) Application Framework: Built on top of the libraries and the runtime, the Application Framework provides a comprehensive set of tools and APIs for developers to build feature-rich applications. It contains components such as Activities, Services, Broadcast Receivers, and Content Providers, facilitating the development of interactive and interconnected applications.

(v) Applications: The top layer of the Android architecture is dedicated to applications that users interact with daily. These can range from pre-installed system apps to third-party applications available for download on the Google Play Store. This layer describes the culmination of the underlying layers, delivering a user-friendly and customizable experience.

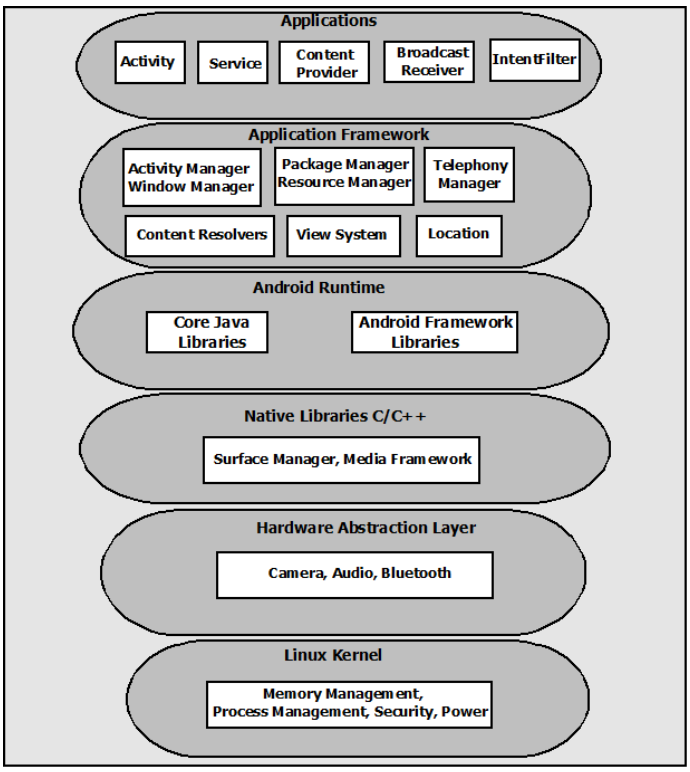

Fig. 1. Android Architecture

B. Android Threat Model

Creating a threat model for Android involves determining potential threats and vulnerabilities in the system to improve security measures. The following is a comprehensive threat model for Android, highlighting potential threats and corresponding mitigation strategies:

1. Malware and Untrusted Apps:

Threat: Installation of malicious apps from third-party sources, leading to data theft, device compromise, or unauthorized access.

Mitigation: Encourage users to download apps only from official app stores. Implement robust app vetting processes to catch and remove malicious applications. Encourage user education on identifying and avoiding potentially harmful apps.

2. Device Theft and Physical Access: Threat: Unauthorized access to the device due to stealing or physical possession.

Mitigation: Enforce strong device passcodes or biometric authentication. Enable device encryption to protect stored data. Implement remote wipe capacities for lost or stolen devices.

3. Network Attacks: Threat: Man-in-the-middle attacks, Wi-Fi spoofing, or eavesdropping on unsecured networks.

Mitigation: Employ encrypted connections (HTTPS) for data communication. Bypass connecting to unsecured Wi-Fi networks. Implement Virtual Private Network (VPN) solutions for secure network access.

4. Phishing Attacks: Threat: Deceptive tries to trick users into revealing sensitive information through fake websites or emails.

Mitigation: Educate users about recognizing phishing attempts. Implement secure browsing practices, including HTTPS usage. Employ email filtering to detect and block phishing emails.

5. Operating System and Software Vulnerabilities: Threat: Exploitation of security vulnerabilities in the Android OS or installed apps.

Mitigation: Regularly update the device's operating system and applications. Employ a secure development lifecycle for app development. Implement timely security patches to address known vulnerabilities.

6. Insecure Data Storage:

Threat: Storage of sensitive information in an insecure manner, making it susceptible to unauthorized access.

Mitigation: Implement strong encryption for sensitive data stored on the device. Utilize secure storage APIs provided by the Android platform. Avoid storing sensitive information in plain text.

7. Social Engineering Attacks:

Threat: Manipulation of users through psychological tactics to disclose confidential information.

Mitigation: Familiarize users about social engineering tactics and understanding. Implement multi-factor authentication for added security. Enable users to verify the legitimacy of requests for sensitive information.

8. Lack of Device Updates:

Threat: Devices running outdated software without security updates are vulnerable to known exploits.

Mitigation: Encourage users to enable automatic updates for both the operating system and applications. Provide timely security patches and updates for supported devices. Promote the use of devices with long-term support and update commitments. 9. Insufficient App Permissions:

Threat: Apps requesting excessive or unnecessary permissions, leading to privacy breaches.
Mitigation: Implement a granular permission system that allows users to control app permissions. Encourage users to review and understand app permissions before installation. Promote the principle of least privilege for app developers. 10. Insufficient User Education:

Threat: Users may lack awareness of security best practices, making them susceptible to various threats.
Mitigation: Provide user-friendly security education and awareness materials. Incorporate security tips and guidelines into the user interface. Conduct regular security awareness campaigns.

IV. Machine Learning for Android Security

Machine learning plays a crucial role in malware detection for Android devices, offering a dynamic and adaptive approach to recognize and mitigate growing threats. The use of machine learning in Android malware detection provides several benefits over conventional signature-based methods, permitting for the detection of both known and unknown malware. As mobile device usage continues to surge, so do malware attacks, specifically targeting Android phones, which command a substantial 72.2% share of the market. Hackers employ various tactics, including credential theft, watch, and malicious advertising, to compromise smartphones. Among the plethora of countermeasures available, machine learning (ML)-based methods have emerged as effective means of detecting such attacks. ML techniques emanate classifiers from sets of training examples, bypassing the need for explicit signatures when developing malware detectors. This adaptability improves the efficacy of malware detection in the dynamic aspect of cybersecurity threats.

In Android malware detection, two primary methodologies are employed: signature-based detection and behavior-based detection. Signature-based detection involves comparing the binary code of an application with known malware signatures kept in a database. While this method is precise, efficient, and produces low false positives, it cannot detect unknown malware. Hence, behavior-based or anomaly-based detection methods are commonly utilized. These methods leverage techniques from machine learning (ML) and data science to analyze the behavior of applications. Numerous research studies have explored the use of both traditional ML-based methods, such as Decision Trees (DT) and Support Vector Machines (SVM), as well as novel Deep Learning (DL)-based models like Deep Convolutional Neural Networks (Deep-CNN) and Generative Adversarial Networks (GANs) for Android malware detection. The researchers leverage datasets such as Drebin, Google Play, AndroZoo, AppChina, Tencent, YingYongBao, Contagio, Genome/MalGenome, VirusShare, IntelSecurity/McAfee, MassVet, Android Malware Dataset (AMD), APKPure, Android Permission Dataset, Andrototal, Wandoujia, Kaggle, CICMaldroid, AZ, and Github for experimentation and model training. This comprehensive use of diverse datasets highlights the robustness and applicability of ML techniques in Android malware detection.

There are key elements of the role of machine learning in Android malware detection: (a) Traditional signature-based detection relies on known patterns, making it useless against new and growing malware. Machine learning models can adapt to new and previously unseen malware patterns by learning from diverse datasets. This adaptability is important in the dynamic aspect of Android malware.

(b) Malware often shows dynamic behavior, making it difficult to detect based on static features alone. Behavioral analysis, facilitated by machine learning, allows for the identification of malicious behavior patterns. Models can learn normal behavior and detect anomalies, signaling potential malware activities.

(c) Traditional methods may struggle with pulling relevant features from large datasets for effective malware detection. Machine learning techniques enable automatic feature extraction and selection, identifying the most relevant aspects of data for accurate malware detection. This improves the efficiency of the detection process.

(d) Signature-based approaches may fail to detect zero-day threats that exploit unknown vulnerabilities. Machine learning models can identify previously unseen threats by learning patterns and behaviors rather than relying on predefined signatures. This is particularly important for early detection of emerging threats.

(e) Analyzing large datasets manually for malware patterns can be time-consuming and impractical. Machine learning algorithms excel at processing large-scale datasets efficiently. They can analyze patterns, relationships, and anomalies across vast amounts of data, enhancing the overall detection ability.

(f) Some malware may execute quickly or change behavior rapidly, requiring real-time detection capabilities. Machine learning models, especially those based on streaming algorithms, can provide real-time analysis, enabling swift detection and response to emerging threats.

(g) Traditional methods may yield false positives, flagging benign applications as malicious. Machine learning models can be trained to reduce false positives by learning from comprehensive datasets, enhancing accuracy in distinguishing between malicious and legitimate behaviors.

(h) Detecting collaborative malicious activities across multiple applications (app collusion) can be intricate. Machine learning enables the monitoring of inter-app communication and monitoring app permissions to identify patterns meaningful of app collusion. This enhances the ability to detect sophisticated attacks involving multiple applications.

(i) The threat landscape is continually evolving, necessitating constant updates to detection methods. Machine learning models can undergo continuous learning and adaptation, ensuring that they remain effective against new and arising Android malware threats without the need for manual intervention.

V. Genetic Algorithm Based Malware Detection in Android

In this section, we propose Genetic Algorithm based Malware Detection system for Android. Genetic algorithms can be effectively employed for feature selection in machine learning by evolving subsets of features that contribute most to the predictive performance of a model. The process involves representing potential feature subsets as individuals in a population, using a fitness function to evaluate their performance, and applying genetic operations such as crossover and mutation to evolve better subsets over successive generations. There are many benefits using genetic algorithm for feature selection.
(a) Genetic algorithms efficiently explore the vast combinatorial space of possible feature subsets, making them suitable for high-dimensional datasets.
(b) Genetic algorithms can capture non-linear relationships between features, allowing them to discover complex interactions that may enhance predictive performance.
(c) The algorithm tends to select subsets that minimize redundancy and exclude irrelevant features, improving the efficiency of the model. (d) Genetic algorithms aim for global optimization by considering diverse solutions, reducing the risk of getting stuck in local optima.
(e) Genetic algorithms are model-agnostic and can be applied to feature selection for different machine learning models.

A. Dataset

We employ Drebin dataset [9] that is a well-known dataset used in the field of Android malware detection and classification. It is specifically designed for evaluating the performance of machine learning models in detecting malicious Android applications (malware). The dataset was created to address the need for a standardized benchmark to assess the effectiveness of various detection methods.
The Drebin dataset consists of a collection of Android applications, including both benign (non-malicious) and malicious samples. The dataset is labeled, with each sample annotated as either benign or belonging to a specific malware family. The dataset is relatively large, containing thousands of samples. The exact number of samples may vary, as the dataset has been expanded and updated over time to include new instances of Android malware. Drebin covers a diverse set of Android malware families, representing various types of malicious behavior and attack strategies. This diversity is crucial for evaluating the robustness of malware detection models across different threats. Each sample in the Drebin dataset is associated with a set of features extracted from the Android application's manifest and code. These features include permissions requested, API calls made, and other relevant characteristics that can be used to train machine learning models for malware detection. he dataset provides ground truth labels, indicating whether each application is benign or belongs to a particular malware family. These labels are essential for supervised learning approaches, where machine learning models are trained on labeled data.
For data preprocessing, we handle missing values by replacing them with zeros for each column and converts categorical variables into numerical values. We drop rows where the value of 'TelephonyManager.getSimCountryIso' column is '?' and 'TelephonyManager.getSimCountryIso' and 'class' columns to integers are converted. Then, we maps 'class' column values to binary integers (1 for 'S', which presumably stands for malicious, and 0 for 'B', which stands for benign).

B. Genetic Algorithm for Feature Selection

We employ a genetic algorithm (GA) to select features based on a defined fitness function. This define the fitness function and runs the GA to select features with the highest fitness value.
Before fitting the Isolation Forest model, the code selects features based on the binary representation of the chromosome (selected_features). This selection process is determined by the genes in the chromosome; if a gene is set to 1, the corresponding feature is included, otherwise it's excluded. Selected Features: [0 3 4 7 8 9 10 11 13 15 16 18 19 20 22 23 24 28 29 31 32 35 36 40 44 45 47 48 50 51 52 53 54 56 57 58 60 61 62 63 64 69 71 75 78 80 86 88 93 96 99 100 101 102 103 104 106 107 111 112 115 118 119 123 128 129 132 133 134 136 138 140 141 142 150 151 152 155 156 157 158 160 161 163 167 171 173 174 175 177 179 181 184 185 186 187 189 190 192 194 199 200 201 202 205 207 208 210 211]
Once the features are selected, the Isolation Forest model is fitted using the training data with only the selected features. The Isolation Forest algorithm then constructs an ensemble of decision trees. Each tree is built recursively by randomly selecting features and split points to isolate data points. After fitting the Isolation Forest model, anomaly scores are computed for the training data that returns the anomaly score for each data point, which indicates how much of an outlier it is. Negative scores typically indicate anomalies, with lower scores suggesting a higher likelihood of being an anomaly.
Finally, the fitness function computes the average anomaly score based on the scores obtained from the Isolation Forest model. This average anomaly score serves as the fitness value for the chromosome, reflecting how well the selected features perform in detecting anomalies. The Objective function is 0.06360752464585982 that is shown in figure 2.

C. Train Model with ANN

After feature selection, we train with random forest and ANN. Random forest is an ensemble learning method that consists of a collection of decision trees. Each tree in the forest is trained independently and generates a prediction. The final prediction is typically made by aggregating the predictions of all individual trees. In training, RF uses a bagging technique and random feature selection to train multiple decision trees independently. Each tree is trained on a bootstrap sample of the training data, and a random subset of features is considered at each split point.

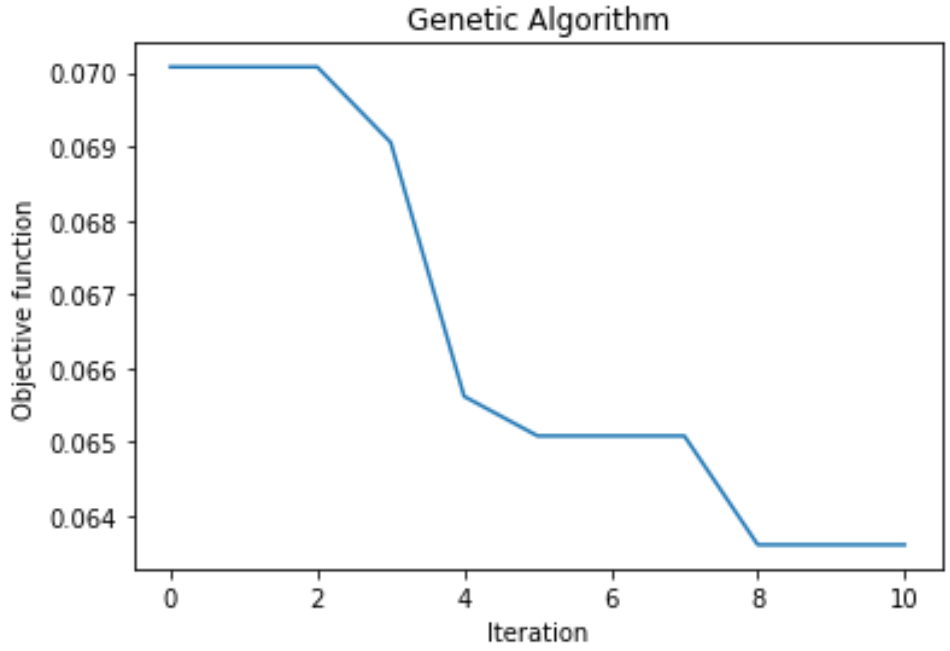

Fig. 2. Genetic Algorithm

ANN is a computational model inspired by the biological neural networks of the human brain. It consists of interconnected layers of artificial neurons (nodes). Neurons in each layer receive input, apply an activation function, and pass the output to neurons in the next layer. ANNs can have multiple hidden layers between the input and output layers, giving them the ability to learn complex patterns in data. For training, ANN typically uses gradient-based optimization algorithms (e.g., backpropagation) to iteratively adjust the weights and biases of the network to minimize a loss function. During training, the network learns to map input data to output labels by adjusting the parameters based on the error between predicted and true labels.

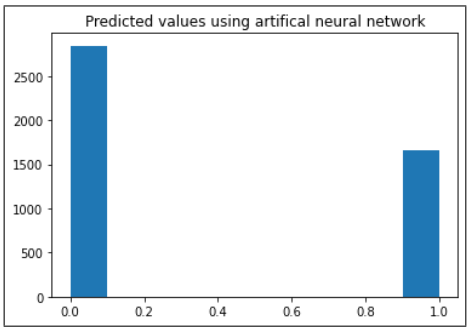

Fig. 3. Predicted Values

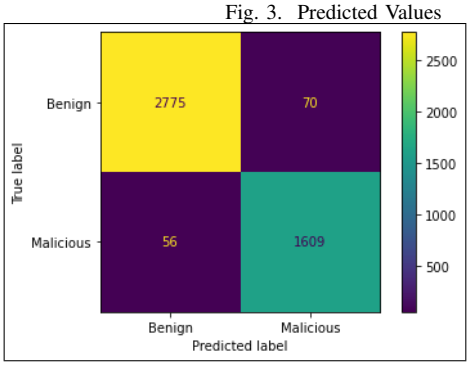

Fig. 4. Confusion Matric

VI. Experiment And Results

In this section, we perform the experiments and discuss the results. We employ drebin dataset and we employ genetic algorithm for feature selection approach. We use and then build model with random forest and ANN. We use the accuracy as the performance parameter and we also compare our work with existing work. For our experiment, we have physical machine that has 16 GB RAM, 500 GB SSD, and with intel core i7 2.80 GHz CPU. We use Python as a programming language. After perform experiment, we obtained the accuracy 91.44 % and 97.20% with random foreat and ANN respectively. We show the predicted values with ANN in figure 3 and confusion matrix in figure 4. With confusion matrix, we show the performance of a classification model on a set of test data for which the true values are known. Accuracy of the model is shown in table 1.

VII. Conclusion and future Scope

This research addresses the growing concern of Android malware and its potential impact on a wide range of devices beyond smartphone. It shows the need for robust security measures to combat the escalating threat of malware in the

Android system. The research paper provides valuable insights into the evolving landscape of Android malware and proposes innovative approaches for enhancing security measures through the application of machine learning techniques. This paper provides the android architecture to understand the each components of the system that helps to understand the attack vectors that can be compromised. We also present the threat model so that potential threat can be identified. Then we propose a genetic algorithm with ANN based security system that shows promising results.

REFERENCES

[1] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan, "Android security: a survey of issues, malware penetration, and defenses," *IEEE communications surveys & tutorials*, vol. 17, no. 2, pp. 998–1022, 2014.

[2] H. Yuan, Y. Tang, W. Sun, and L. Liu, "A detection method for android application security based on tf-idf and machine learning," *Plos one*, vol. 15, no. 9, p. e0238694, 2020.

[3] M. Zhao, F. Ge, T. Zhang, and Z. Yuan, "Antimaldroid: An efficient svm-based malware detection framework for android," in *Information Computing and Applications: Second International Conference, ICICA 2011, Qinhuangdao, China, October 28-31, 2011. Proceedings, Part I 2*, pp. 158–166, Springer, 2011.

[4] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, pp. 1–29, 2014.

[5] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pp. 15–26, 2011.

[6] G. Dini, F. Martinelli, A. Saracino, and D. Sgandurra, "Madam: a multi-level anomaly detector for android malware," in *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*, pp. 240–253, Springer, 2012.

[7] W.-C. Wu and S.-H. Hung, "Droiddolphin: a dynamic android malware detection framework using big data and machine learning," in *Proceedings of the 2014 conference on research in adaptive and convergent systems*, pp. 247–252, 2014.

[8] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "Copperdroid: Automatic reconstruction of android malware behaviors.," in *Ndss*, pp. 1–15, 2015.

[9] M. H. H. G. Daniel Arp, Michael Spreitzenbarth and K. Rieck, "Drebin: Efficient and explainable detection of android malware in your pocket," *21th Annual Network and Distributed System Security Symposium (NDSS),*, 2014.