IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

PermGuard: A Scalable Framework for Android Malware Detection Using Permission-to-Exploitation Mapping

ARVIND PRASAD[1], SHALINI CHANDRA [2], MUEEN UDDIN [3], TAHER AL-SHEHARI [4], NASSER A ALSADHAN [5], SYED SAJID ULLAH [6]

[1]Department of Computer Engineering and Applications, GLA University, Mathura, India
[2]Department of Computer Science, BBA University, Lucknow India
[3]College of Computing and Information Technology, University of Doha for Science and Technology, Doha, Qatar
[4]Computer Skills, Department of Self-Development Skill, Common First Year Deanship, King Saud University, Riyadh, Saudi Arabia
[5]Computer Science Department, College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia
[6]Department of Information and Communication Technology, University of Agder, Grimstad, Norway

Corresponding author: Mueen Uddin(mueen.uddin@udst.edu.qa)

This work was supported by the Researchers Supporting Project number (RSPD2024R846), King Saud University, Riyadh, Saudi Arabia.
The Open Access funding was provided by the Qatar National Library.

ABSTRACT Android, the world's most widely used mobile operating system, is increasingly targeted by malware due to its open-source nature, high customizability, and integration with Google services. The increasing reliance on mobile devices significantly raises the risk of malware attacks, especially for non-technical users who often grant permissions without thorough evaluation, leading to potentially devastating effects. This paper introduces PermGuard, a scalable framework for Android malware detection that map permissions into exploitation techniques and employs incremental learning to detect malicious apps. It presents a novel technique for constructing the PermGuard dataset by mapping Android permissions to exploitation techniques, providing a comprehensive understanding of how permissions can be misused by malware. The dataset consists of 55,911 benign and 55,911 malware apps, providing a balanced and comprehensive foundation for analysis. Additionally, a new strategy using similarity-based selective training reduces the amount of data required for the training of an incremental learning-based model, focusing on the most relevant data to improve efficiency. To ensure robustness and accuracy, the model adopts a test-then-train approach, initially testing on application data to identify weaknesses and refine the training process. The framework's resilience is tested against adversarial attacks, demonstrating its ability to withstand attempts to bypass or deceive detection mechanisms and enhance overall security. Designed for scalability, PermGuard can handle large and continuously growing datasets, making it suitable for real-world applications. Empirical results indicate that the model achieved an accuracy of 0.9933 on real datasets and 0.9828 on synthetic datasets, demonstrating strong resilience against both real and adversarial attacks.

INDEX TERMS Android Malware Detection, Machine Learning, Permissions Exploitation, Cybersecurity, Mobile Security

I. INTRODUCTION

Android OS (Operating System) continues to dominate the global mobile operating system market, holding a 71.67% share [1], as of August 2024. This dominance is even more evident when looking at all operating systems worldwide, where Android leads with 45.38%, far ahead of Windows at 25.61% and iOS (iPhone Operating System) at 18.39%. Because of its widespread use, Android has become a prime target for cyberattacks. In recent times, millions of malware attacks have been detected and blocked on Android mobile devices by various security vendors [2], [3].

With over three billion active users globally, Android is the most popular mobile operating system [4]. Its success comes from several key factors: its open-source nature, high customizability, and seamless integration with Google services. However, these same features make it attractive to cybercriminals to develop malicious apps. The open nature and support for wide range of devices mean there are more chances

for hackers to find and exploit weaknesses, and to spread malware. This makes Android a big target for malicious activities. This subsection explores the factors contributing to Android's popularity and how these same factors make it an attractive target for cybercriminals to create and spread malware:

Open Source Nature: The open-source nature of the Android OS encourages a vast developer community to create a wide variety of applications, further increasing its popularity. Because it is open source, manufacturers and developers can modify the source code to create unique user experiences and features, attracting many manufacturers and boosting its adoption. However, open-source code is accessible to everyone, including hackers. They can study the code to find and exploit weaknesses, creating security risks. Additionally, The platform's support for third-party app (application) stores, which often have low security checks, further aids in the distribution of malicious apps [5], [6].

Security Gaps: The rapid and widespread adoption of Android sometimes outpaces the development of robust security measures. Despite this, users flock to Android because of its features and affordability. Many Android devices run different versions of the OS, and not all receive timely updates. This fragmentation leaves many devices with known vulnerabilities unpatched, making them easy targets for malware. Security patches often reach devices slowly, if at all, giving hackers a larger window of opportunity to exploit these security gaps.

Diverse Manufacturers: A multitude of manufacturers use Android, offering a wide range of devices from budget-friendly to high-end models. This variety helps Android reach a broad audience. The presence of numerous manufacturers means Android can cater to different markets and preferences, contributing to its widespread use. However, different manufacturers have varying levels of commitment to security. Some may not prioritize timely updates or robust security measures, increasing the risk for users. Customizations and modifications by different manufacturers can introduce unique vulnerabilities that may not be present in the base Android OS.

Google Play Store and Beyond: The Google Play Store offers a wide selection of apps, enhancing the appeal of Android devices. This vast ecosystem attracts users seeking diverse applications. The Google Play Store's integration with Android devices makes it easy for users to find and download apps, contributing to user satisfaction and retention. Despite Google's efforts to secure the Play Store, malicious apps occasionally bypass security checks and infiltrate the platform. Once in the store, these apps can be downloaded by many users before they are detected and removed. Moreover, Android allows the installation of apps from third-party stores, which often lack the security measures of the Google Play Store. These sources can be rife with malware, posing significant risks to users who download from these sources.

Malware apps exploit permissions granted by the Android OS to perform malicious activities [7], [9]. For example, the Joker malware pretends to be real apps and uses SMS (Short Message Service) permissions to secretly sign users up for various online services without their knowledge [8]. Similarly, the BankBot trojan steals financial information by using accessibility services to display fake login screens on top of real banking apps, tricking users into entering their passwords [10]. The Ghimob trojan abuses permissions like SMS, call log, and accessibility services to intercept and forward authentication codes, allowing attackers to bypass two-factor authentication and access victims' accounts.

These examples illustrate how Android malware leverages permissions granted to apps to carry out various malicious activities, ranging from subscription fraud to credential theft [11]. As Android permissions grant apps access to sensitive device resources and user data, it is crucial for users to exercise caution when granting permissions and to regularly review and uninstall suspicious apps.

Addressing this challenge requires innovative approaches that transcend traditional signature-based detection methods. Machine learning (ML) emerges as a promising solution, leveraging advanced algorithms to analyze vast datasets and identify intricate patterns indicative of malicious behavior [12]–[18]. ML-based Android malware detection frameworks offer several advantages, including adaptability to evolving threats, scalability to handle large datasets, and efficiency in detecting previously unseen malware variants [19]–[24].

Android permissions serve as an important line of defense against malware by providing insights into app behavior and enabling the detection of suspicious activities. Security solutions leverage permission analysis techniques to identify potential indicators of malware and protect users from malicious apps. By carefully analyzing the permissions requested by an app, security researchers and antivirus solutions can identify potential indicators of malicious behavior.

The research addresses a critical problem in Android security: the increasing prevalence of malware attacks that exploit app permissions. With over three billion active users globally, Android's open-source nature makes it a prime target for cybercriminals. Traditional malware detection methods, such as signature-based approaches, are proving ineffective against evolving malware threats, especially those that manipulate permissions for malicious purposes. This research proposes a scalable, machine-learning-based framework that dynamically analyzes Android permissions and maps them to exploitation techniques [25]. By focusing on subtle indicators of malicious behavior, this research enhances malware detection efficiency, addressing the urgent need for better mobile security in real-world scenarios.

Here are the major contributions of this research, highlighting its innovative aspects and practical significance:

- A dataset construction technique is proposed that collects 55,911 benign apps and 55,911 malware apps. It further extracts Android permissions from these apps and maps them into exploitation techniques. This method enables a comprehensive understanding of how

permissions can be misused by malware while significantly reducing the number of features.
- The proposed framework includes a similarity-based selective training approach to minimize the data required for training, focusing on the most relevant information and reducing computational overhead.
- PermGuard applies a test-first approach to validate its robustness and accuracy by evaluating multiple datasets before refining the learning process.
- The framework is evaluated for its resilience against adversarial attacks, ensuring that it can withstand attempts to deceive or bypass its detection mechanisms, thereby enhancing security.
- PermGuard is scalable, capable of managing large and expanding datasets.

The remainder of the paper is organized as follows: Section 2 discusses existing work on Android malware detection. Section 3 presents the proposed PermGuard framework, followed by Section 4, which explains the dataset construction technique. Section 5 describes the malware detection techniques, while Section 6 provides the evaluation results. Finally, Section 7 concludes the paper and outlines directions for future work.

II. RELATED WORK

In Android malware detection, recent research has introduced various machine learning approaches aimed at mitigating the ever-evolving landscape of threats. Prior works have contributed significantly to the field, laying the groundwork for advancements in feature selection, model interpretability, and detection accuracy. Researchers aim to create effective solutions for identifying malicious applications by delving into the intricacies of Android malware behaviour. This section offers an in-depth exploration of some of the state-of-the-art work within this domain.

Wajahat et al. [26] introduced a lightweight Android malware detection system employing explainable machine learning. Through recursive feature elimination, they pinpointed critical features, reducing computational overhead while preserving high accuracy. The utilization of Shapley additive explanation (SHAP) values enhances interpretability. Diverging from traditional methods, the approach emphasizes static feature analysis, enhancing efficiency and obviating the need for app execution. Experimental results demonstrate accuracy (>0.99 F1-score) with minimal resource consumption. Furthermore, the study presents a condensed dataset of 40 high-impact features, facilitating future research. In sum, it presents a practical solution for IoT security, making notable strides in feature selection, transparency, and efficiency.

Almarshad et al. [27] introduce a pioneering approach to Android malware detection by integrating Siamese one-shot learning with machine learning algorithms. Utilizing the Drebin dataset comprising 9476 benign and 5560 malware apps, the proposed model achieves an impressive accuracy of 98.9%. Through the amalgamation of Siamese one-shot learning and machine learning algorithms, the authors effectively address the challenge of limited training samples. Moreover, they employ visualization analysis for feature comprehension, enabling the model to surpass traditional methods. The Siamese network architecture demonstrates robustness, generalization, and the ability to avoid overfitting. The study underscores its superiority over LSTM (Long Short-Term Memory), random forest classifier, and SVM (Support Vector Machine), marking significant advancements in malware detection efficacy.

Aurangzeb and Aleem [28] addressed the pressing issue of detecting obfuscated Android malware, which poses a significant security threat. They introduced an ensemble voting-based classification approach and a novel feature-based obfuscation mechanism to defend against Android malware. Leveraging the Kronodroid dataset, the study conducts comprehensive evaluations on both real devices and emulators. It demonstrates the effectiveness of the proposed deep learning model in accurately detecting obfuscated malware while identifying key features crucial for detection. Notably, the study highlights the importance of a small subset of features for detection, emphasizing the significance of intelligent tools in combating evolving malware threats.

Odat and Yaseen [29] utilized co-existing static features and machine learning techniques to detect Android malware. The authors established that malware demonstrates abnormal co-existence patterns of permissions and APIs (Application Programming Interface) compared to benign apps. They constructed a dataset from Drebin, Malgenome, and MalDroid2020, and extracted co-existence features at varying levels, which were then analyzed using the FP-growth algorithm. Several conventional machine learning algorithms were employed for evaluation, with Random Forest achieving the highest accuracy of 98%. This approach outperformed existing models, notably surpassing PermPair, and achieved up to 98% accuracy. The research work highlights the efficacy of feature co-existence in Android malware detection, providing valuable insights for future research.

Alani and Awad [30] proposed PAIRED, a novel approach to Android malware detection, addressing the security challenges posed by the widespread adoption of the Android operating system. Leveraging explainable machine learning, the system achieved over 98% accuracy while maintaining a small footprint on devices. It employed feature selection via recursive feature elimination, reducing the number of critical features necessary for accurate detection. The use of SHAP (Shapley Additive Explanation) values enhanced model explainability. PAIRED introduced a reduced version of the dataset, enhancing future research. With high accuracy and explainability, PAIRED stood out as a significant advancement in Android malware detection.

Alkahtani and Aldhyani [31] have addressed the escalating threat of malware targeting Android devices. By employing machine learning and deep learning techniques, such as SVM (Support Vector Machine), KNN (K-Nearest Neighbors), LDA (Linear Discriminant Analysis), LSTM (Long Short-Term Memory), CNN-LSTM (Convolutional Neural

Network, Long Short-Term Memory), and autoencoder algorithms, they have made significant contributions. The proposed approach was experimented with on the CICAndMal2017 [32] and Drebin [33] datasets, where SVM reached 100% accuracy, while LSTM achieved 99.40%. The research emphasizes feature selection through correlation analysis, highlighting the efficiency of SVM, LSTM, and CNN-LSTM in malware detection. The study underscores the importance of AI-based security systems in mitigating evolving threats to Android devices, showcasing the superior performance of SVM and LSTM models.

Gupta et al. [34], the authors proposed a novel method for detecting Android malware by applying rough set theory. The main contribution lies in prioritizing features using discernibility matrices and ranking scores, leading to enhanced detection accuracy. Various machine learning models, including Support Vector Machines, K-Nearest Neighbor, Random Forest, and Logistic Regression, were utilized. Feature selection was performed through a combination of data preprocessing and rough set theory, improving overall detection effectiveness while minimizing complexity. The proposed model achieved a 97

Vu and Jung [35] introduced AdMat, a novel framework for Android malware detection. It employs adjacency matrices as input images for Convolutional Neural Networks (CNNs). AdMat achieves an average detection rate of 98.26% across various malware datasets and successfully categorizes over 97.00% of different malware families with limited training data. The framework proposes adjacency matrices for effective feature engineering, which is validated by CNN models, showcasing the approach's efficacy. AdMat demonstrates promising results in malware classification, signifying a significant advancement in mobile security research.

Surendran et al. [36] proposed GSDroid, a novel approach to Android malware detection using graph signal processing. It addressed the limitations of existing high-dimensional feature representations by proposing a low-dimensional representation based on system call sequences. Through modeling system call sequences as directed graphs and defining informative signals on graph vertices, the method captured system call dependencies effectively. Using machine learning classifiers like random forest and decision tree, it achieved a remarkable accuracy of 0.99 with a 16-dimensional feature vector. The main contributions included graph-based representation of Android apps, defining informative signals, and demonstrating superior performance in malware detection. Overall, GSDroid offered a promising solution for combating sophisticated Android malware.

Mehtab et al. [37] introduced AdDroid, a rule-based machine learning framework for Android malware analysis. AdDroid used 63 meticulously crafted rules, derived from diverse application artefacts, to pinpoint malicious behavior accurately. Employing an ensemble-based strategy incorporating Adaboost and traditional classifiers, AdDroid delivered an outstanding 99.11% accuracy on a robust dataset of 1420 Android applications. Feature selection techniques further refined the model's efficacy. Noteworthy for its high true positive and true negative rates, AdDroid demonstrated versatility across varying application types. Its streamlined computational complexity facilitated real-time analysis, effectively combating the challenge of detecting Android malware. AdDroid stood out as an indispensable tool for analysts, empowering them in the precise identification and categorization of potential malware applications.

Mahindru and Sangal [38] proposed MLDroid, a web-based framework for Android malware detection using machine learning. The authors applied feature selection techniques to reduce feature sets and enhance model performance. The experiment, conducted on a dataset of over 500,000 Android apps, reveals a 98.8% detection rate using four machine learning algorithms: deep learning, farthest first clustering, Y-MLP, and nonlinear ensemble decision tree forest. MLDroid's innovation lies in its utilization of permissions, API calls, app ratings, and downloads as features, enabling efficient detection even of unknown malware families. This research underscores the effectiveness of feature selection in improving detection accuracy and reducing misclassification errors, with notable contributions from various machine learning techniques.

Li et al. [39] introduced an Android malware detection method that uses multimodal fusion of fine-grained features extracted from both source code (PM) and binary code (MM). Their method employs sensitive API calls and RGB image processing for feature extraction. Experimented on a dataset of 10,000 benign and 10,000 malicious apps, the method demonstrated high accuracy of 98.28% and robust generalization of 92.86%.

Liu et al. [40] proposed SeGDroid, a novel Android malware detection method based on sensitive function call graph learning. Using datasets like CICMal2020 and MalRadar, SeGDroid achieved 98% accuracy in malware detection and 96% in family classification. The model employs graph pruning to reduce training complexity by focusing on sensitive APIs, though it may face challenges when dealing with smaller graphs. Despite these limitations, SeGDroid provides efficient and accurate detection through advanced graph-based learning mechanisms.

After reviewing the existing state-of-the-art work in Android malware detection using machine learning, it is evident that researchers have made significant strides in addressing security challenges. Their dedication has led to the development of various effective detection methodologies. However, the following are a few limitations that are lacking in the existing studies:

- Existing studies lack large and up-to-date datasets, potentially limiting the effectiveness of their models.
- Some studies may not effectively leverage Android permissions as features, resulting in larger dataset sizes or less efficient feature representation.
- The scalability of existing frameworks may be limited, particularly in handling large and continuously growing

datasets.

- Classical machine learning techniques employed in prior studies need complete retraining when data changes, potentially leading to inefficiencies in model maintenance and adaptation to evolving threats.

III. PROPOSED FRAMEWORK

With mobile devices becoming integral to daily life, users frequently grant app permissions without proper evaluation or understanding of the potential risks, thereby increasing their vulnerability to malicious attacks. Android malware can exploit these permissions to access sensitive data, monitor user activities, or take control of the device.

PermGuard addresses this escalating threat by providing a robust solution specifically designed to combat Android malware. By mapping permissions into exploitation techniques, PermGuard effectively identifies and mitigates these threats, ensuring greater security for Android users.

This section presents the flowchart of the proposed PermGuard framework in Figure 1, offering a comprehensive visual representation of the Android malware detection process. The flowchart illustrates each stage of the framework, which includes the following steps:

- Starts with app collection from various sources
- Permission extraction from the collected apps.
- Construction of a dataset of both benign and malicious applications.
- Mapping permissions to exploitation techniques commonly used by malware.
- Application of similarity-based selective training to reduce computational overhead.
- Utilization of incremental learning to improve the efficiency of malware detection.
- Testing the model on the PermGuard dataset for validation.
- Testing on a synthetic dataset to evaluate resilience against adversarial attacks.

The subsequent sections provide detailed overview of each stages.

IV. ANDROID MALWARE DATASET CONSTRUCTION TECHNIQUE

Machine learning models trained on large datasets typically exhibit better generalization performance [41]. By exposing models to a wide range of malware instances, researchers can mitigate overfitting and improve the model's ability to accurately classify previously unseen samples [42]–[45]. This section outlines the process of constructing a comprehensive and up-to-date dataset of Android malware, a crucial step in improving the efficacy of machine learning algorithms for threat detection.

A. ANDROID APPLICATION COLLECTION

This subsection outlines an automated approach for downloading Android applications from Androzoo [46]. It has two

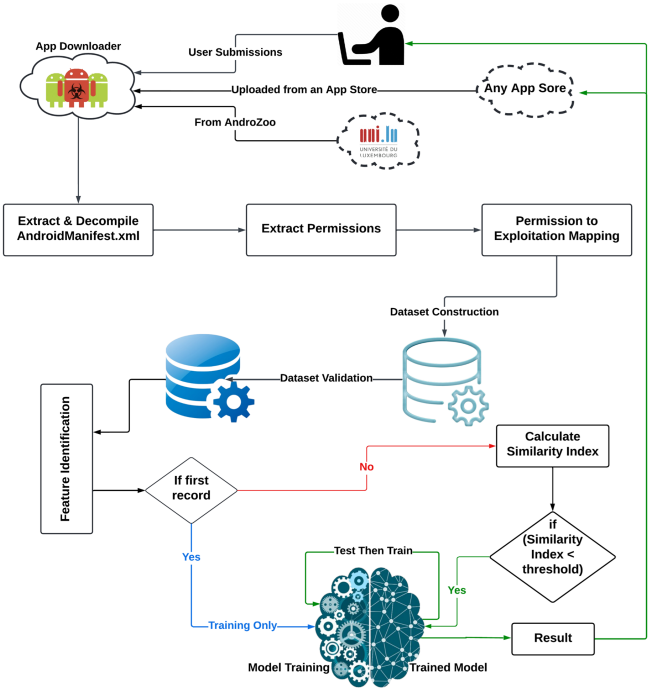

FIGURE 1. Flowchart of the PermGuard Framework for Android Malware Detection

steps, **Prerequisite Application Data** and **Android application download from Androzoo**. The **Prerequisite Application Data** step is a Python script that filters the extensive 'latest.csv' dataset downloaded from Androzoo website to include only Android application records scanned between January 1, 2019, and July 1, 2024. It separates benign and malware applications based on detection values, streamlining analysis by focusing on relevant data subsets. Algorithm 1 presents the pseudo-code for acquiring Android application data for further processing.

Algorithm 1 Pseudo code for Prerequisite Application Data

Require: CSV (Comma Separated Values) file path containing dataset ('latest.csv').

Ensure: Processed dataframes *dfBenign* and *dfMalware*.

1: Load CSV file into a dataframe:
$df \leftarrow$ pd.read_csv("latest.csv")
2: Filter dataframe to select records scanned after '2019-01-01':
$dfNew \leftarrow df.loc[(df['vt\_scan\_date'] >' 2019-01-01')]$
3: Extract records identified as benign:
$dfBenign \leftarrow dfNew.loc[dfNew['vt\_detection'] == 0]$
4: Extract records identified as malware:
$dfMalware \leftarrow dfNew.loc[dfNew['vt\_detection'] > 10]$
5: Concatenate benign and malware dataframes:
$df \leftarrow pd.concat([dfBenign, dfMalware])$

The **Android application download from Androzoo** step is a Python script that facilitates this automated downloading approach. This script uses sha256 (Secure Hash Algorithm 256-bit) values extracted from a local Excel file named 'latest.csv', downloaded directly from the Androzoo website.

The automation process further requires an API key obtained from the Androzoo platform. The script constructs unique download URLs using this key by combining the API key with each sha256 hash. The script then sends requests to Androzoo to download the Android applications.

The script saves the downloaded applications locally after successful retrieval. Each file is named after its corresponding sha256 value, ensuring clear identification and organization. This streamlined approach empowers researchers with efficient access to Android applications for analysis and research endeavours. The researchers downloaded 55911 benign and 55911 malware apps using the above discussed technique. The pseudo-code to download applications is given in Algorithm 2.

Algorithm 2 Pseudo code for Android application download from Androzoo

Require: Obtain dataframe df from Algorithm 1

Ensure: Downloaded Android application files saved locally with their corresponding sha256 values as filenames.

Define function *download_apk(api_key, sha256):*
Construct the URL for downloading the APK (Android Application Package) using the provided API key and sha256 value.
Send a GET request to the constructed URL.
if the response status code is 200 **then**
    Open a new file with the name as the sha256 value and ".apk" extension in binary write mode.
    Write the content of the response to the newly created file.
    Print a success message indicating the download.
else
    Print a failure message indicating the download failure.
end if
Define function *main():*
Load sha256 values from df (obtained from prerequisite data processing)
Include 'Your_API_Key' into obtained from Androzoo.
for each sha256 value in the list **do**
    Call the *download_apk* function with the API key and current sha256 value as arguments.
end for
Call the *main* **function.**

B. DECOMPILATION AND PERMISSION EXTRACTION

Following the download of the applications, the next crucial step involved the extraction of permissions requested by the app. This was achieved by extracting the AndroidManifest.xml file from each application. Subsequently, the axmldec tool was used to decompile the XML (Extensible Markup Language) file and decompile these files to make it human-readable [47]. This process extracts the permissions requested by each application.

Initially, two lists were compiled: one comprising the top 100 permissions frequently utilized in benign applications and the other containing the top 100 permissions prevalent in malware applications. These lists were then combined, consolidating the permission landscape across benign and malicious applications. Subsequently, redundant permissions were removed, resulting in a refined list which includes 123 distinct permissions.

C. MAPPING OF PERMISSIONS INTO TECHNIQUES FOR MALWARE DEVELOPMENT

To better understand the behavior of malicious Android applications, the researchers map Android permissions to specific malware development techniques. This approach maps individual permissions into broader exploitation categories, providing a more comprehensive view of how malware can misuse permissions.

The 123 permissions identified in previous subsection, were mapped onto 23 techniques that could potentially be used in developing Android malware. These techniques are identified based on the working approach of various Android malware. Table 1 presents the mapping of permissions into techniques for malware development. By categorizing permissions into broader techniques such as "App Permission Exploitation" or "Network Surveillance," the researchers gain a detailed understanding of the potential threats posed by app permissions. This mapping allows for a detailed analysis of app behavior, enabling the identification of patterns and trends associated with various types of malicious activity.

Table 2 categorizes Android permissions into broader exploitation techniques commonly used by malware. It highlights how malicious actors can exploit these permissions for various attacks, such as data exfiltration, credential theft, or device control. Understanding these techniques is crucial for enhancing malware detection systems and improving Android security.

The approach of mapping Android permissions into techniques for malware development offers several advantages:

- Aggregating permissions into higher-level techniques makes the dataset more compact and manageable. This condensed representation simplifies analysis and visualization.
- Grouping permissions based on their potential for exploitation enables a focused analysis of the techniques commonly employed by malware developers.
- Many permissions may contribute to multiple malware development techniques. By consolidating these permissions, redundancy in the dataset is reduced, leading to a more efficient use of resources for analysis and mitigation.

TABLE 1. List of mapped Android permissions into techniques for malware development

Malware Development Techniques	Android Permissions
Remote Command Execution	INTERNET, BIND_DEVICE_ADMIN, PROCESS_OUTGOING_CALLS, READ_SMS, SEND_SMS, READ_CALL_LOG, READ_LOGS, RECORD_AUDIO, CAMERA, MODIFY_PHONE_STATE, WRITE_EXTERNAL_STORAGE
Rootkit Installation	INTERNET, BIND_DEVICE_ADMIN, READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE, READ_LOGS, MODIFY_PHONE_STATE, RECORD_AUDIO, CAMERA, SYSTEM_OVERLAY_WINDOW, ACCESS_SUPERUSER
Exploit Delivery	INTERNET, WRITE_EXTERNAL_STORAGE, READ_EXTERNAL_STORAGE, INSTALL_PACKAGES, REQUEST_INSTALL_PACKAGES
Data Exfiltration	INTERNET, READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE, ACCESS_NETWORK_STATE
Credential Theft	INTERNET, READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE, RECORD_AUDIO, CAMERA, READ_CONTACTS, GET_ACCOUNTS
Screen Logging	INTERNET, READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE, RECORD_AUDIO, CAMERA
Keylogging	INTERNET, READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE, RECORD_AUDIO, CAMERA, READ_LOGS, READ_SMS, RECEIVE_SMS, READ_CONTACTS
Audio Surveillance	RECORD_AUDIO, INTERNET, READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE
Social Engineering Attack	READ_CONTACTS, SEND_SMS, READ_SMS, RECEIVE_SMS, WRITE_EXTERNAL_STORAGE
GPS Spoofing	ACCESS_ASSISTED_GPS, ACCESS_GPS
Device Bricking	DEVICE_POWER
Call Interception	READ_PHONE_STATE, READ_CALL_LOG, PROCESS_OUTGOING_CALLS
Network Traffic Interception	INTERNET, ACCESS_NETWORK_STATE, CHANGE_NETWORK_STATE
Device Lockout	MODIFY_PHONE_STATE, WRITE_SECURE_SETTINGS, SHUTDOWN
Browser Hijacking	INTERNET, WRITE_HISTORY_BOOKMARKS, WRITE_SECURE_SETTINGS
System Settings Modification	WRITE_SETTINGS, WRITE_SECURE_SETTINGS, CHANGE_CONFIGURATION
File System Manipulation	READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE, ACCESS_CACHE_FILESYSTEM, WRITE_INTERNAL_STORAGE
Camera Hijacking	CAMERA, RECORD_AUDIO, RECORD_VIDEO
App Installation without User Consent	INSTALL_PACKAGES, REQUEST_INSTALL_PACKAGES, INSTALL_SHORTCUT, DOWNLOAD_WITHOUT_NOTIFICATION
Location Tracking	ACCESS_LOCATION
Contact Information Theft	READ_CONTACTS, WRITE_CONTACTS
Browser History Theft	READ_HISTORY_BOOKMARKS, ACCESS_DOWNLOAD_MANAGER
Package Management Manipulation	INSTALL_PACKAGES, REQUEST_INSTALL_PACKAGES, DELETE_PACKAGES, PACKAGE_USAGE_STATS
Notification Manipulation	BIND_ACCESSIBILITY_SERVICE
System Log Manipulation	READ_LOGS
Process Management Manipulation	KILL_BACKGROUND_PROCESSES, BIND_JOB_SERVICE, UPDATE_APP_OPS_STATS
Alarm Hijacking	SET_ALARM, RECEIVE_BOOT_COMPLETE, RECEIVE_USER_PRESENT
Calendar Event Manipulation	READ_CALENDAR, WRITE_CALENDAR
Task Manipulation	REORDER_TASKS, GET_TASKS
Fake App Installation	INSTALL_PACKAGES, REQUEST_INSTALL_PACKAGES
Bluetooth Hijacking	BLUETOOTH, BLUETOOTH_ADMIN
WiFi Network Hijacking	ACCESS_WIFI_STATE, CHANGE_WIFI_STATE
USB Debugging Exploitation	WRITE_SECURE_SETTINGS
Screen Overlay Attack	SYSTEM_ALERT_WINDOW
Sim Card Manipulation	READ_PHONE_STATE, WRITE_SETTINGS
Battery Drain Attack	MODIFY_PHONE_STATE
SMS Spamming	SEND_SMS, BROADCAST_SMS, SEND_RESPOND_VIA_MESSAGE, RECEIVE_MMS, MESSAGE
Ad Fraud	ACCESS_NETWORK_STATE, INTERNET
Account Information Theft	GET_ACCOUNTS, MANAGE_ACCOUNTS, AUTHENTICATE_ACCOUNTS
Certificate Manipulation	READ_EXTERNAL_STORAGE, WRITE_EXTERNAL_STORAGE
Runtime Environment Manipulation	MODIFY_AUDIO_SETTINGS, MODIFY_PHONE_STATE
Call Log Manipulation	READ_CALL_LOG, WRITE_CALL_LOG

- Quantifying the occurrence of each technique allows the researchers to prioritize defense strategies based on the prevalence of specific threats. High-occurrence techniques may warrant immediate attention for mitigation efforts, while less common techniques can be addressed as secondary concerns.
- As new permissions are introduced in Android releases and malware developers exploits these new permissions, this approach remains adaptable by updating the mapping of permissions to techniques.

V. EMPLOYING DATA VALIDATION STRATEGIES FOR EFFECTIVE FEATURE SELECTION

In this section, the researchers explore the application of data validation strategies as a precursor to effective feature selection. The construction of this extensive Android malware dataset enables researchers to observe evolving trends

TABLE 2. Description of each of the mapped Android permissions focusing on their role in malware development

Malware Development Technique	Description focusing on their role in malware development
Remote Command Execution	Allows malware to remotely execute commands, enabling control over the device, launching malicious actions, or sending/receiving data covertly.
Rootkit Installation	Enables malware to gain elevated privileges, bypassing security mechanisms and granting persistent, undetectable control over the device for malicious purposes.
Exploit Delivery	Facilitates the downloading and installation of additional malicious payloads or exploits, expanding the malware's impact and capabilities without user intervention.
Data Exfiltration	Allows the transmission of sensitive user data, including personal, financial, or system information, to external servers controlled by attackers.
Credential Theft	Enables unauthorized access to user credentials, such as login information or account details, which can be exploited for fraud or identity theft.
Screen Logging	Captures sensitive on-screen information, such as user inputs or displayed data, for the purpose of stealing personal or financial details.
Keylogging	Monitors user input, including keystrokes or other forms of input, allowing malware to record sensitive information such as passwords or messages.
Audio Surveillance	Enables the recording of audio from the device's microphone, allowing attackers to eavesdrop on private conversations or gather sensitive information.
Social Engineering Attack	Facilitates the sending and receiving of deceptive messages designed to trick users into divulging personal information or installing further malware.
GPS Spoofing	Alters the device's reported location, misleading users or location-based services and potentially enabling fraud or misrepresentation.
Device Bricking	Maliciously disables key device functionalities, rendering the device inoperable, often as part of ransomware or destructive attacks.
Call Interception	Enables the interception or monitoring of phone calls, allowing attackers to eavesdrop on conversations or steal call-related information.
Network Traffic Interception	Allows the interception or manipulation of network communications, enabling attackers to monitor, alter, or redirect data for malicious purposes.
Device Lockout	Prevents the user from accessing the device, often used in ransomware attacks where attackers demand payment to restore access.
Browser Hijacking	Redirects browser traffic or alters browser settings, potentially leading users to malicious websites or altering browsing history for tracking purposes.
System Settings Modification	Alters critical system settings, disabling security features or allowing the malware to maintain persistent control over the device.
File System Manipulation	Enables the reading, modification, or deletion of files, allowing malware to tamper with user data or hide its presence within the file system.
Camera Hijacking	Gains unauthorized access to the device's camera, allowing malware to capture photos or videos for surveillance or blackmail.
App Installation without User Consent	Installs malicious applications without the user's consent, escalating malware activity or gaining further control over the device.
Location Tracking	Tracks the device's real-time location, allowing attackers to monitor user movements or target location-based attacks.
Contact Information Theft	Steals or manipulates contact data, which can be used to propagate malware or gather information for social engineering attacks.
Browser History Theft	Accesses browsing history to track user behavior, potentially launching targeted phishing attacks or selling user data to third parties.
Package Management Manipulation	Maliciously installs or uninstalls applications, including removing security apps and installing harmful software to further compromise the device.

and patterns in malware behaviour over time. Additionally, the balanced distribution of benign and malware instances ensures robustness and reliability in model training and evaluation, mitigating biases commonly associated with imbalanced datasets.

Dataset validation is crucial in ensuring the quality, integrity, and relevance of the data used for analysis. Density plots, heatmaps, and histograms are powerful visualization techniques that serve as effective approaches for dataset validation. Density plots provide insights into the distributional characteristics of data, allowing for identifying patterns and anomalies. Heatmaps visualize correlations between variables, highlighting relationships and trends within the dataset. Histograms offer a detailed depiction of data distribution, facilitating the assessment of central tendency, variability, and shape. These visualization tools enable researchers to validate the dataset's quality, integrity, and relevance, enhancing the robustness and reliability of subsequent analyses and modelling endeavours.

Effective feature selection is essential for streamlining the modeling process, enhancing interpretability, and boosting predictive performance. This approach reduces dimensionality, mitigates overfitting, and optimizes model efficiency by identifying and prioritizing the most informative and discriminative features [48]–[53]. Furthermore, feature selection facilitates the extraction of actionable insights from complex datasets, enabling researchers to focus on factors that significantly influence the outcome of interest.

In the following subsections, the researchers delve into the applications of visualization techniques—density plots,

heatmaps, and histograms—as powerful tools for dataset validation. Further, the researchers explore their role in identifying and prioritizing informative features to ensure robust and reliable modelling outcomes.

A. DENSITY PLOTS FOR DATASET VALIDATION AND FEATURE SELECTION

Density plots visualize the distribution of data by estimating the probability density function of a continuous variable. These plots provide insights into the shape, spread, and peaks of the data distribution, allowing for the identification of patterns and anomalies. The probability density function (PDF) represents the probability distribution of a continuous random variable. It describes the likelihood of the variable taking on a particular value within a given range.

Density plots are a visual representation of the probability density function of a continuous variable. The kernel density estimate (KDE) is commonly used to construct density plots. The mathematical formula for the kernel density estimate at a point x is given by:

$$\hat{f}(x) = \frac{1}{n \cdot h} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right) \tag{1}$$

Where:

$\hat{f}(x)$ is the kernel density estimate at point x.

n is the number of observations in the dataset.

h is the bandwidth parameter, controlling the smoothness of the estimate.

X_i are the observed data points.

K(u) is the kernel function.

Commonly used kernel functions include the Gaussian kernel K(u)

$$K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} \tag{2}$$

and the Epanechnikov kernel K(u)

$$K(u) = \frac{3}{4}(1 - u^2) \tag{3}$$

This above formula calculates the kernel density estimate by summing up the contributions of each data point, weighted by the chosen kernel function and bandwidth parameter. The resulting estimate represents the probability density of observing a data point at the specified value x.

Figure 2 shows the distribution of each feature's values for both malware and benign samples.

The density Plots provide insights into the probability density of each feature for the two classes, helping to identify patterns or differences between malware and benign data. The researchers considered following aspect while analysing the density graph in Figure 2:

- A significant overlap suggests that the feature alone may not be a strong indicator of malware or benign behavior.
- If the distributions are well-separated, it indicates that the feature can potentially discriminate between malware and benign samples effectively.
- A skewed distribution may indicate the presence of outliers or a non-normal distribution of data, which could impact the effectiveness of the feature in distinguishing between malware and benign samples.
- Features with distinct peaks or narrower spreads for malware or benign samples may provide valuable discriminatory information.
- Features that consistently exhibit differences in distributions between malware and benign samples are likely more informative for classification purposes.

Table 3 shows the ranking of the features based on the mean density value. The researchers calculated the absolute difference between normalized density value of benign and malware applications. The researchers removed the feature which has highest absolute difference between normalized density value as they cannot potentially discriminate an application between malware and benign application effectively.

B. HEATMAP FOR DATASET VALIDATION AND FEATURE SELECTION

The heatmap illustrates the correlation matrix of features in the dataset. Features with values close to 1 or -1 indicate strong positive or negative correlations respectively, while values close to 0 suggest weak or no correlation. Strong correlations between features might indicate redundancy, potentially allowing for feature reduction. Conversely, low correlations may imply independence or lack of relationship between features.

The formula to calculate the Pearson correlation coefficient (ρ) is given in Equation 2

$$\rho = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum(X_i - \bar{X})^2 \sum(Y_i - \bar{Y})^2}} \tag{4}$$

Where:

X_i and Y_i are the individual data points of variables X and Y respectively.

$\bar{X}$ and $\bar{Y}$ are the means of variables X and Y respectively.

Figure 3 shows the heatmap plotted using the Pearson correlation coefficient. Most of the correlation values in the heatmap are low. It indicates that all features are diverse and independent. Low correlations between variables are desirable, especially in predictive modelling tasks, as highly correlated features can introduce multicollinearity and potentially degrade model performance. To remove highly correlated features, the researchers first identified a list of features with correlation values higher than 0.85. The researchers then removed features that appeared more than once, suggesting they have high correlations with multiple features. These redundant features were then removed to avoid potential multicollinearity issues, which can affect model performance. Table 4 presents the correlations between various features with values above 0.85. Only the highlighted features from this list were retained for further analysis, as they demonstrated lower redundancy compared to others.

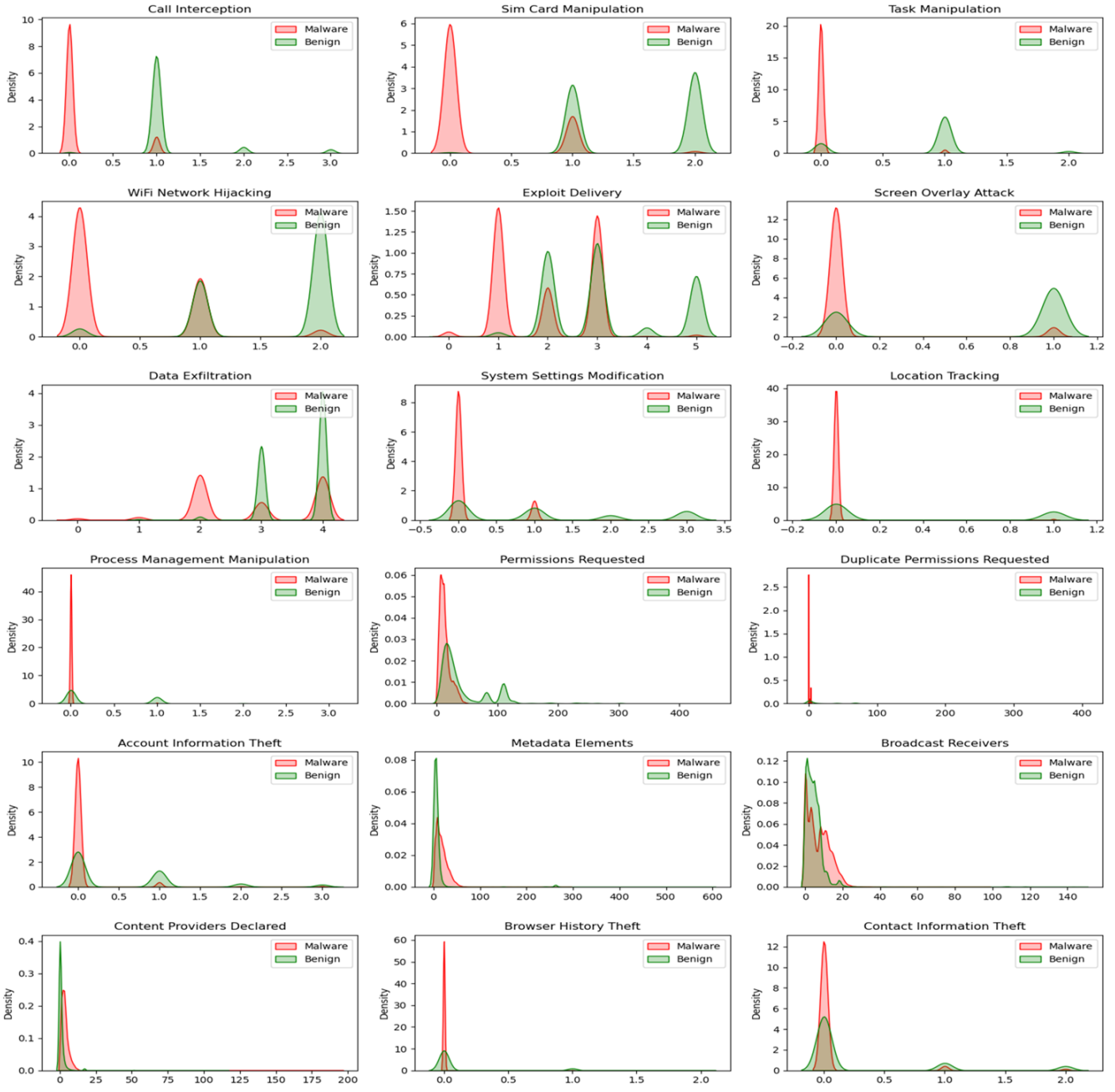

FIGURE 2. Density plot for visualising the distribution of features in the dataset

C. HISTOGRAM PLOTS FOR DATASET VALIDATION AND FEATURE SELECTION

Histograms provide a visual representation of the distribution of data within each feature. By comparing the distributions of different features, one can identify patterns or irregularities that may require further investigation. It visually represents the data distribution by dividing it into intervals and counting the frequency of data points falling into each interval, which is then normalized to represent a probability distribution. The process of generating histograms involves two primary steps: binning and normalization.

- Binning involves dividing the range of the data into intervals, called bins or buckets. The number of bins and their width can vary depending on the data and the desired level of detail in the histogram.
- After binning, the counts of data points falling into each bin are typically normalized to represent a probability distribution. This normalization ensures that the area under the histogram's curve sums up to 1, making it a probability density function. This step involves dividing the count of data points in each bin by the total number of data points and the bin width.

TABLE 3. Ranking of features by absolute difference in normalized density between benign and malware applications

Features	Normalized Benign Density	Normalized Malware Density	Absolute Difference (High to Low)
Duplicate Permissions Requested	0.125000	0.820769	0.695769
Content Providers Declared	0.125000	0.749692	0.624692
Account Information Theft	0.766379	0.181077	0.585302
Remote Command Execution	0.125000	0.678615	0.553615
Permissions Requested	0.285345	0.820769	0.535424
Screen Overlay Attack	0.766379	0.252154	0.514225
Screen Logging	0.125000	0.607538	0.482538
Metadata Elements	0.285345	0.749692	0.464347
Ad Fraud	0.606034	0.181077	0.424957
Device Bricking	0.125000	0.536462	0.411462
Rootkit Installation	0.285345	0.678615	0.393270
Activities Declared	0.445690	0.820769	0.375079
System Log Manipulation	0.766379	0.394308	0.372071
USB Debugging Exploitation	0.606034	0.252154	0.353880
System Settings Modification	0.125000	0.465385	0.340385
Call Log Manipulation	0.445690	0.110000	0.335690
Keylogging	0.285345	0.607538	0.322193
Version Code	0.445690	0.749692	0.304002
Location Tracking	0.766379	0.465385	0.300994
Task Manipulation	0.606034	0.323231	0.282803
Contact Information Theft	0.125000	0.394308	0.269308
SMS Spamming	0.445690	0.181077	0.264613
Call Interception	0.285345	0.536462	0.251117
Exploit Delivery	0.445690	0.678615	0.232925
Browser Hijacking	0.766379	0.536462	0.229917
Services Declared	0.606034	0.820769	0.214735
Notification Manipulation	0.606034	0.394308	0.211726
Process Management Manipulation	0.125000	0.323231	0.198231
WiFi Network Hijacking	0.445690	0.252154	0.193536
File System Manipulation	0.285345	0.465385	0.180040
Runtime Environment Manipulation	0.285345	0.110000	0.175345
Audio Surveillance	0.445690	0.607538	0.161848
GPS Spoofing	0.766379	0.607538	0.158841
Target SDK Version	0.606034	0.749692	0.143658
App Installation without User Consent	0.606034	0.465385	0.140649
Fake App Installation	0.125000	0.252154	0.127154
Calendar Event Manipulation	0.445690	0.323231	0.122459
Browser History Theft	0.285345	0.394308	0.108963
Battery Drain Attack	0.285345	0.181077	0.104268
Network Traffic Interception	0.44569	0.536462	0.090772
Credential Theft	0.766379	0.678615	0.087764
Data Exfiltration	0.606034	0.678615	0.072581
Device Lockout	0.606034	0.536462	0.069572
Sim Card Manipulation	0.125000	0.181077	0.056077
Broadcast Receivers	0.766379	0.820769	0.054389
Package Management Manipulation	0.445690	0.394308	0.051382
Alarm Hijacking	0.285345	0.323231	0.037886
Bluetooth Hijacking	0.285345	0.252154	0.033191
Camera Hijacking	0.445690	0.465385	0.019695
Is App Taking Backup	0.766379	0.749692	0.016687
Certificate Manipulation	0.125000	0.110000	0.015000
Social Engineering Attack	0.606034	0.607538	0.001504

Figure 4 shows the plotted histogram comparing feature distributions between malware and benign applications. The researchers considered the following points while validating the dataset.

- A significant difference in the distribution of a feature between classes may indicate its importance in distinguishing between benign and malware applications.
- A data point significantly different from the rest of the dataset can be identified in histograms as values that fall outside the expected range or have a shallow frequency.
- Features that exhibit clear separation between classes are likely to be more informative and useful for classification tasks.
- Skewed distributions or features with different scales between classes may indicate the need for such preprocessing techniques to improve model performance.

Considering the aforementioned criteria, the researchers examined Figure 4. This figure depicts a subset of features with minimal absolute differences between the normalized frequencies of benign and malware applications. Table 5

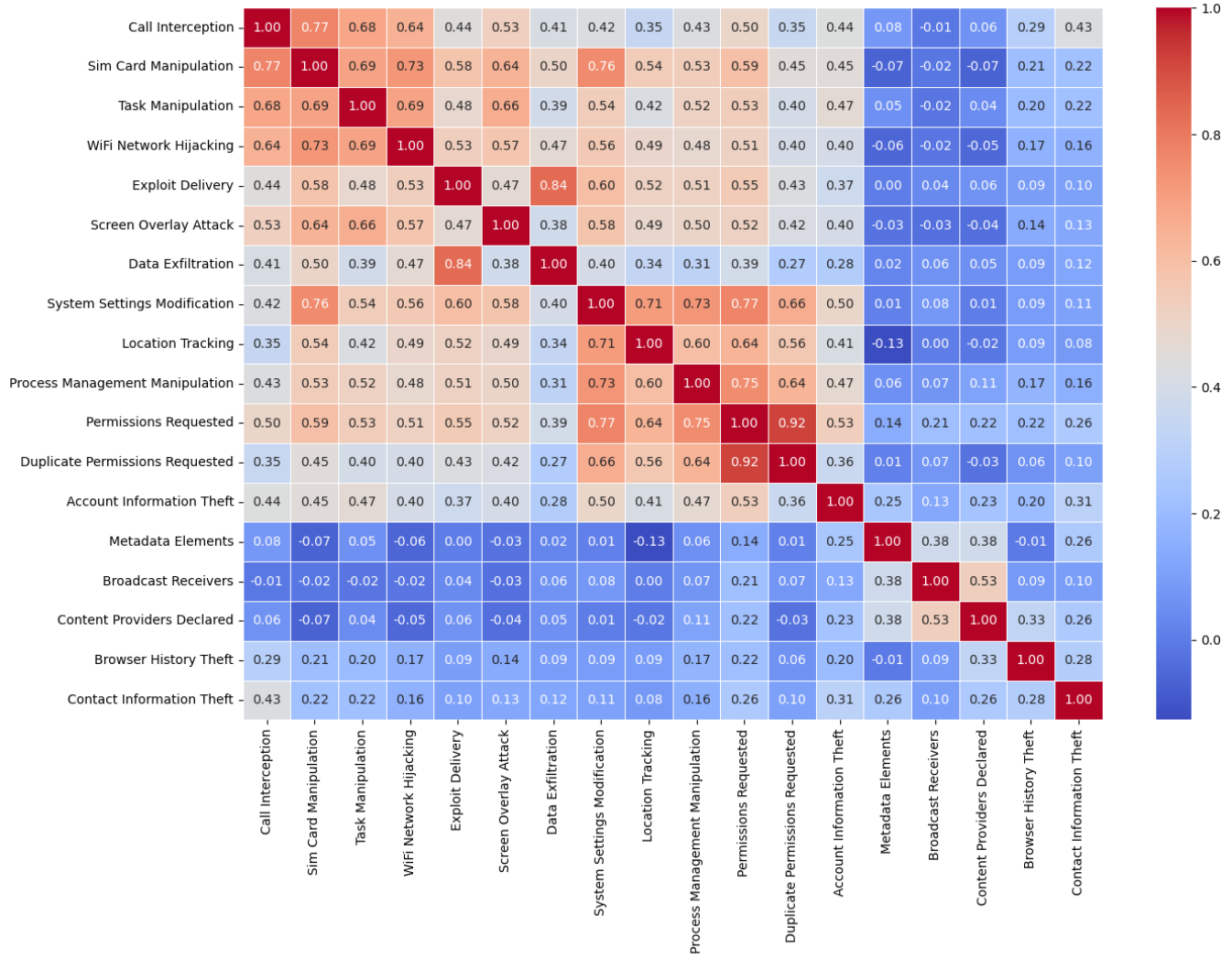

FIGURE 3. Correlation Heatmap to visualizing relationships between features

TABLE 4. Features with high correlation Values (>0.85)

First Feature	Second Feature	Correlation (>0.85)
Rootkit Installation	Screen Logging	0.94960
Keylogging	**Remote Command Execution**	0.94553
Credential Theft	Screen Logging	0.94451
Device Lockout	USB Debugging Exploitation	0.94346
Package Management Manipulation	Fake App Installation	0.94078
Audio Surveillance	Screen Logging	0.93652
Credential Theft	Keylogging	0.92738
Permissions Requested	**Duplicate Permissions Requested**	0.92346
Data Exfiltration	Audio Surveillance	0.91767
Credential Theft	Rootkit Installation	0.91414
File System Manipulation	**Exploit Delivery**	0.90425
Audio Surveillance	Rootkit Installation	0.90082
Keylogging	Rootkit Installation	0.89494
Browser Hijacking	USB Debugging Exploitation	0.89357
Fake App Installation	App Installation without User Consent	0.88291
Credential Theft	Audio Surveillance	0.87308
Keylogging	Screen Logging	0.86605
Device Lockout	Browser Hijacking	0.86112
Package Management Manipulation	App Installation without User Consent	0.85305

Highlighted features were retained for further analysis, as they appeared only once in correlations.

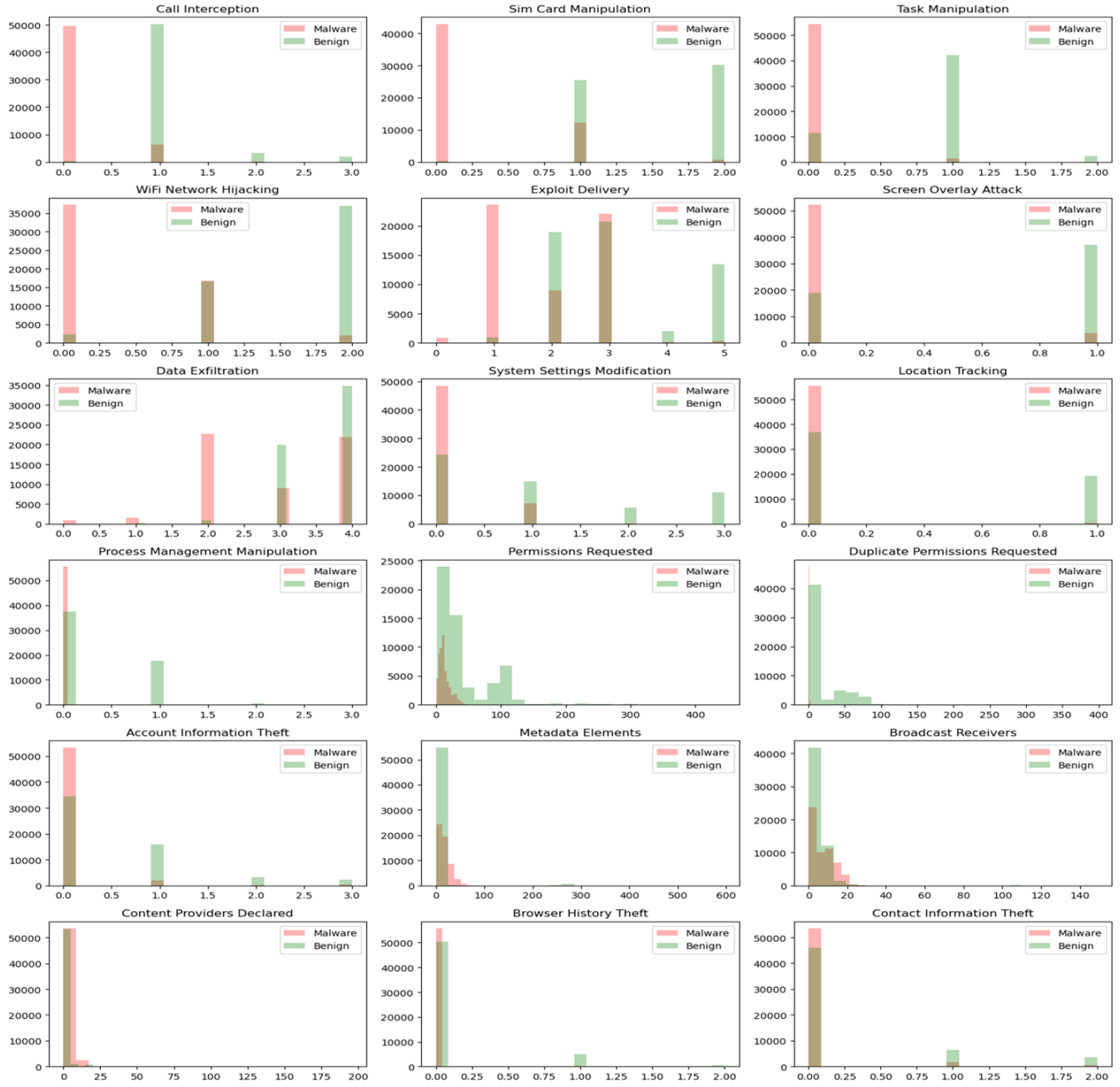

FIGURE 4. Comparison of feature distributions between malware and benign applications

presents the feature ranking based on these normalized frequencies. The researchers eliminated the feature with the lowest absolute difference.

VI. MALWARE DETECTION TECHNIQUE

In this section, the researchers present an overview of the malware detection technique employed in this research, delineating its methodology, innovation, and advantages. This approach embodies an ensemble model utilizing incremental learning classifiers, namely MLPClassifier, BernoulliNB, and PassiveAggressiveClassifier, meticulously chosen through empirical evaluation from a pool of seven classifiers.

A. IDENTIFYING ALGORITHM AND BUILDING ENSEMBLE MODEL

To determine the top-performing incremental learning algorithm for the proposed ensemble model, the researchers evaluated seven different incremental learning algorithms: MLPClassifier, GaussianNB, MultinomialNB, BernoulliNB, PassiveAggressiveClassifier, Perceptron, and SGDClassifier. The selection of these seven algorithms was based on their

TABLE 5. Feature ranking based on normalized frequencies between malware and benign applications

Features	Normalized Benign Frequency	Normalized Malware Frequency	Absolute Difference (High to Low)
Duplicate Permissions Requested	0.125000	0.807358	0.682358
Broadcast Receivers	0.731522	0.110000	0.621522
Permissions Requested	0.125000	0.720189	0.595189
Battery Drain Attack	0.731522	0.197170	0.534352
Ad Fraud	0.125000	0.633019	0.508019
Content Providers Declared	0.327174	0.807358	0.480184
GPS Spoofing	0.731522	0.284340	0.447182
System Settings Modification	0.125000	0.545849	0.420849
Sim Card Manipulation	0.529348	0.110000	0.419348
Screen Overlay Attack	0.327174	0.720189	0.393015
Process Management Manipulation	0.731522	0.371509	0.360013
Task Manipulation	0.125000	0.458679	0.333679
Browser History Theft	0.529348	0.197170	0.332178
Device Bricking	0.327174	0.633019	0.305845
Account Information Theft	0.529348	0.807358	0.278010
Call Interception	0.731522	0.458679	0.272843
Exploit Delivery	0.125000	0.371509	0.246509
Runtime Environment Manipulation	0.529348	0.284340	0.245008
Call Log Manipulation	0.327174	0.545849	0.218675
Data Exfiltration	0.327174	0.110000	0.217174
Metadata Elements	0.529348	0.720189	0.190841
Location Tracking	0.731522	0.545849	0.185673
WiFi Network Hijacking	0.125000	0.284340	0.159340
Notification Manipulation	0.529348	0.371509	0.157839
Contact Information Theft	0.327174	0.458679	0.131505
Calendar Event Manipulation	0.327174	0.197170	0.130004
Activities Declared	0.529348	0.633019	0.103671
System Log Manipulation	0.731522	0.633019	0.098503
Remote Command Execution	0.731522	0.807358	0.075836
Target SDK Version	0.125000	0.197170	0.072170
SMS Spamming	0.529348	0.458679	0.070669
Services Declared	0.327174	0.371509	0.044335
File System Manipulation	0.327174	0.284340	0.042834
Version Code	0.529348	0.545849	0.016501
Network Traffic Interception	0.125000	0.110000	0.015000

The highlighted features indicate the lowest absolute difference and were eliminated from further analysis.

ability to support incremental learning through the **partial_fit** function, which enables the models to continue learning as new data becomes available. The experiments were conducted on a dataset constructed for this study. Each algorithm was evaluated on its performance in terms of accuracy, precision, recall, and overall detection efficacy. Table 6 shows the experimental result.

The experimental results revealed that three algorithms—MLPClassifier, BernoulliNB, and PassiveAggressiveClassifier—consistently outperformed the others across various performance metrics.

MLPClassifier: Demonstrated high accuracy and robustness in handling diverse malware patterns, owing to its multilayer perceptron architecture.

BernoulliNB: Exhibited strong performance particularly in binary/boolean feature spaces, making it well-suited for the feature set derived from Android permissions.

PassiveAggressiveClassifier: Showed excellent adaptability and efficiency, quickly learning from misclassifications and adjusting its parameters accordingly.

Based on these findings, the researchers selected MLPClassifier, BernoulliNB, and PassiveAggressiveClassifier for the ensemble model. These algorithms' superior performance in the incremental learning context ensures that the framework can effectively detect and adapt to new and evolving Android malware threats.

B. MODEL TRAINING APPROACH

In this subsection, we detail the methodology used to train the PermGuard framework. This approach focuses on efficiency and adaptability, leveraging novel techniques to ensure that the model remains robust and effective against a wide array of malware threats.

1) The Sequentially Prediction-Then-Training Approach

Fundamental to the proposed approach is the incremental learning paradigm, a dynamic framework that allows the model to adjust to changing datasets without the need for extensive retraining. Unlike conventional machine learning models, incremental learning enables smooth adaptation to emerging malware variants by continuously updating the model with new data. This model building and training approach are grounded in a sequential process aimed at optimizing model adaptability while minimizing data exposure bias.

TABLE 6. Feature ranking based on normalized frequencies between malware and benign applications

Incremental learning algorithms	Accuracy	Precision	Recall	F1Score
BernoulliNB	**0.96566**	0.97505	0.95589	0.96537
MLP Classifier	**0.96259**	0.94595	0.98137	0.96334
PassiveAggressive Classifier	**0.91636**	0.85853	0.99732	0.92273
MultinomialNB	0.61681	0.90346	0.26292	0.40730
GaussianNB	0.51218	0.69973	0.04536	0.08519
SGD Classifier	0.51116	0.50650	0.92935	0.65566
Perceptron	0.50043	1	0.00244	0.00487

The bold values highlight outperforming incremental learning algorithms.

The following steps delineate this sequentially prediction-then-training approach:

- **Initialization:** The model is initialized using the first record of the dataset. This step is crucial as it establishes the baseline for subsequent predictions and training iterations.
- **Prediction:** Upon initialization, the model predicts the label for the second record in the dataset. This prediction serves as the basis for subsequent model training.
- **Training:** Following prediction, the model is trained using the features and label of the second record. This training process incrementally updates the model's parameters based on the newly encountered data.
- **Iterative Prediction and Training:** The aforementioned steps of prediction followed by training are iteratively repeated for each subsequent record in the dataset. This iterative approach ensures the model adapts to evolving data patterns while reducing the risk of overfitting.

The rationale behind the prediction-then-training approach lies in mitigating the bias introduced by exposing the model to the entire dataset at once. By sequentially predicting and training on each record, the researchers ensure that the model learns incrementally without being unduly influenced by the entirety of the data. This mitigates the risk of data exposure bias, wherein the model may inadvertently memorize the dataset rather than learn generalized patterns. Advantages:

- **Adaptability:** The incremental learning approach enables the model to adapt seamlessly to evolving datasets, ensuring robust performance against emerging malware variants and changing threat landscapes.
- **Bias Mitigation:** By sequentially predicting and training on individual records, the researchers mitigate the risk of data exposure bias, thereby enhancing the model's generalization capabilities.
- **Efficiency:** The iterative nature of the proposed approach optimizes computational efficiency by incrementally updating the model parameters, obviating the need for wholesale retraining with each new data arrival.

2) Similarity-Based Selective Training Strategy

Similarity-Based Selective Training Strategy: In this research, the researchers present a novel Similarity-Based Selective Training Strategy to make model training more efficient and reduce unnecessary computations and biases from nearly identical data. The key element of this method is the use of cosine similarity to measure the similarity between new data and previously seen data. Before starting the training process, the model evaluates the novelty of the new data by comparing it with the last 100 records. By setting a threshold for data dissimilarity, the model only updates itself when the new data is sufficiently different, preventing overfitting and saving computational resources. Here's how it works:

- Before starting model training, each new data record is checked for similarity to existing records. This helps identify data that is very similar to what the researchers already have.
- The researchers use cosine similarity to compare each new data record with the last 100 records in the dataset. This helps us evaluate how similar the new data is, allowing us to distinguish between new and redundant records.
- To balance model adaptation with computational efficiency, the researchers set a selective training threshold using cosine similarity scores. The researchers test different thresholds, ranging from 80% to 95%, to evaluate how data diversity affects model performance.
- If the cosine similarity between the features of a new app and any of the last 100 trained apps is low, the researchers will train the model on the new app's features.
- Checking the similarity with only the last 100 apps speeds up the process by reducing computational time compared to checking the similarity with all apps.

The formula to calculate cosine similarity between features of app **A** and features of app **B** is given in equation 5.

$$\text{CosineSimilarity}(A, B) = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}} \quad (5)$$

C. HYPERPARAMETER TUNING

VII. RESULT AND DISCUSSION

In this section, the researchers present and analyze the outcomes of the experiments with the incremental learning model for Android malware detection. The results from various classifiers, both individually and as part of an ensemble, are scrutinized to determine their effectiveness in identifying malware. The researchers also examine the impact of the

model training approaches on adaptability, computational efficiency, and bias mitigation. By comparing performance metrics and evaluating real-world applicability, it provides a comprehensive discussion of the findings, highlighting the strengths and potential areas for improvement in this approach.

A. EXPERIMENTAL SETUP

The experiments were conducted on a 64-bit Windows 11 Home (version 23H2) system, equipped with an AMD Ryzen 7 7730U processor with Radeon Graphics running at 2.00 GHz. Model development and testing were carried out using Python 3.11 in a Jupyter Notebook environment, while Java version 21 was employed for additional implementation tasks.

B. PERFORMANCE METRICS

Performance metrics are crucial for assessing the accuracy and effectiveness of a machine learning model's predictions. They offer quantitative measures that facilitate the comparison of different models. In this study, these metrics were extensively employed to identify the best incremental learning algorithms for the proposed ensemble model. They also played a key role in hyperparameter tuning, providing insights into the model's strengths and weaknesses, and benchmarking its performance against various advanced machine learning models. The following metrics were used to evaluate PermGuard's performance:

Accuracy: Accuracy measures the proportion of correctly classified instances out of the total instances. It provides a general sense of how often the model makes correct predictions.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{6}$$

Precision: Precision, also known as Positive Predictive Value, indicates the proportion of true positive results among all positive results predicted by the model. It is crucial for assessing the relevance of the positive predictions.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{7}$$

Recall (Sensitivity): Recall, or Sensitivity, measures the proportion of true positives that were correctly identified by the model. It is essential for evaluating the model's ability to capture all relevant instances.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{8}$$

F1-Score: The F1-Score is the harmonic mean of Precision and Recall, providing a single metric that balances both concerns. The F1-Score combines precision and recall into a single metric by taking their harmonic mean, useful for datasets with imbalanced classes.

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{9}$$

Matthews Correlation Coefficient (MCC): The MCC is a balanced measure that takes into account true and false positives and negatives. It is regarded as a more informative metric than Accuracy in the presence of imbalanced classes.

$$\text{MCC} = \frac{\text{TP} \cdot \text{TN} - \text{FP} \cdot \text{FN}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})(\text{TN} + \text{FP})(\text{TN} + \text{FN})}} \tag{10}$$

Markedness Measure: Markedness assesses the quality of the predictions, focusing on the positive predictive value and negative predictive value. It ranges from -1 to 1, where 1 indicates perfect prediction.

$$\text{Markedness} = \frac{TP}{TP + FP} + \frac{TN}{TN + FN} - 1 \tag{11}$$

Youden's J Statistic: Informedness, also known as Youden's J Statistic, measures the probability of an informed decision. It ranges from -1 to 1, where 1 indicates perfect decision making.

$$J = \frac{TP}{TP + FN} + \frac{TN}{TN + FP} - 1 \tag{12}$$

Fowlkes–Mallows Index: The FM Index measures the similarity between the true positive rate and the positive predictive value. It ranges from 0 to 1, where 1 indicates perfect similarity.

$$\text{FM Index} = \sqrt{\left(\frac{TP}{TP + FP}\right) \times \left(\frac{TP}{TP + FN}\right)} \tag{13}$$

Where,

TP (True Positives): Total correctly predicted benign apps.

TN (True Negatives): Total correctly predicted malwares.

FP (False Positives): Total incorrectly predicted benign apps.

FN (False Negatives): Total incorrectly predicted malwares.

By utilizing these performance metrics, the researchers can thoroughly evaluate the model's prediction capabilities, ensuring a comprehensive understanding of its strengths and limitations.

C. COMPARATIVE ANALYSIS

In this section, the researchers compare the performance of PermGuard with benchmark Android malware datasets. This comparative analysis helps to highlight the strengths and weaknesses of the approach and provides insights into its relative efficacy in detecting Android malware. To evaluate the effectiveness of the model, the researchers use several performance metrics including Accuracy, Precision, Recall, F1-Score, and MCC. The comparison of these metrics across different models is presented in Table 7.

TABLE 7. PermGuard performance on various Android malware datasets

Dataset	Accuracy	Precision	Sensitivity	F1Score	MCC	Markedness Measure	Youden's J Statistic	Fowlkes Mallows Index
Naticusdroid [54]	0.9992	1	0.9984	0.9992	0.9984	0.9984	0.9984	0.9992
Malgenome [55]	0.9573	0.9707	0.9653	0.9680	0.9041	0.9016	0.9066	0.9680
CICMalDroid2020 Syscallsbinders [56]	0.9969	0.9963	1	0.9982	0.9881	0.9963	0.9799	0.9982
CICMalDroid2020 Syscalls [56]	0.9957	0.9950	0.9999	0.9975	0.9835	0.9945	0.9726	0.9975
Android Malware [57]	0.9994	1	0.9914	0.9957	0.9954	0.9994	0.9914	0.9957
PermGuard Dataset	0.9933	0.9934	0.9932	0.9933	0.9866	0.9866	0.9866	0.9933

D. EVALUATION ON OTHER CYBERSECURITY DATASETS

In addition to Android malware detection, it is crucial to assess the versatility and robustness of PermGuard on a variety of cybersecurity datasets. Therefore, the researchers extend the evaluation process to include several prominent cybersecurity datasets, namely Intrusion Detection, DDoS Attack, Phishing Attack, and Botnet Attack datasets. This broader analysis aims to demonstrate the generalizability and efficacy of PermGuard across different types of cyber threats.

The results of the model on the different cybersecurity datasets are summarized in Table 8. Through these comparative analyses, the researchers highlight the strengths and weaknesses of the model in various cyber threat scenarios. This evaluation helps in understanding how well this ensemble incremental learning model adapts to different types of cyber-attacks and its overall effectiveness in a broader cyber-attack context.

E. SIMILARITY-BASED SELECTIVE TRAINING STRATEGY EFFECTIVENESS

In this section, the researchers evaluated the effectiveness of the Similarity-Based Selective Training Strategy by experimenting with various threshold values on a dataset of Android applications. The researchers tested thresholds of 70%, 75%, 80%, 85%, 90%, and 95%, which represents the degree of resemblance between two records of Android applications. The higher the threshold, the more similar the new data must be to the existing data for it to be skipped. Conversely, a lower threshold means the model will train on less similar new data more frequently. The results show a clear relationship between the similarity threshold and model performance metrics, including accuracy, precision, sensitivity, F1 Score, MCC, Markedness Measure, Youden's J Statistic, and Fowlkes–Mallows Index. The findings indicate that as the similarity threshold increases, accuracy and other performance metrics also improve, but at the cost of increased computational load due to more frequent model training.

Table 9 summarizes the experimental results for different threshold values:

Table 10 compares the performance of PermGuard with state-of-the-art Android malware detection approaches across key metrics. PermGuard, evaluated on a significantly larger dataset (111,822 apps), achieves a high accuracy of 99.33%, matching or surpassing other methods. It demonstrates competitive precision (99.34%), recall (99.32%), and F1-measure (99.33%), outperforming several studies. PermGuard's large-scale testing and consistently high performance across all metrics indicate its robustness in detecting malware, proving its effectiveness compared to other models, especially when handling larger datasets, making it a reliable solution for Android malware detection.

The experimental results clearly demonstrate a trade-off between accuracy and computational efficiency. As the similarity threshold increases, the model's accuracy improves, reaching up to 99.31% at a 95% threshold. However, this comes at the cost of needing to train on more records, as evidenced by the decrease in the number of skipped records. On the other hand, lowering the threshold to 70% leads to a higher number of skipped records, thus requiring less training, but results in a slightly lower accuracy of 98.13%.

This trade-off highlights the importance of selecting an appropriate threshold based on the specific needs of the application. For scenarios where high accuracy is paramount, a higher threshold should be chosen, while for scenarios where computational resources are limited, a lower threshold might be more suitable.

The line graph plotted in Figure 5 reveals significant insights into the performance of the model as the similarity threshold increases from 70% to 95%. Increasing the similarity threshold results in better accuracy, precision, sensitivity, and MCC, indicating fewer false positives and negatives and a more reliable model. These metrics collectively show that the model's performance improves as it trains on more distinct data, affirming the efficacy of the selective training strategy.

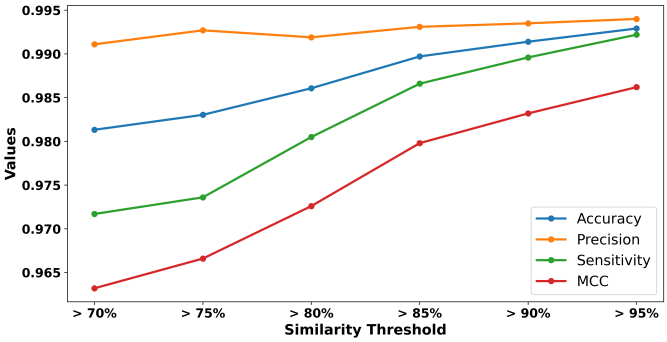

FIGURE 5. Accuracy, Precision, Sensitivity, and MCC vs. Similarity Threshold

The graph plotted in Figure 6 illustrates the number of

TABLE 8. PermGuard performance on various cybersecurity datasets

Dataset	Accuracy	Precision	Sensitivity	F1Score	MCC	Markedness Measure	Youden's J Statistic	Fowlkes Mallows Index
CSECICIDS2018 - Botnet attack [58]	0.9981	0.9994	0.9979	0.9987	0.9951	0.9939	0.9964	0.9987
CICIDS2017 - Botnet attack [58]	0.9897	0.9905	0.9992	0.9948	0.1864	0.4738	0.0733	0.9948
PhiUSIIL - Phishing attack [59]	0.9949	0.9938	0.9974	0.9956	0.9897	0.9903	0.9891	0.9956
CICIDS2017 - DDoS attack [58]	0.9981	0.9987	0.9969	0.9978	0.9962	0.9964	0.9960	0.9978
CICIDS2018 - DDoS attack [58]	0.9977	0.9960	0.9974	0.9967	0.9950	0.9946	0.9953	0.9967
PermGuard Dataset	0.9933	0.9934	0.9932	0.9933	0.9866	0.9866	0.9866	0.9933

TABLE 9. PermGuard performance on various similarity index

Performance Metric	Similarity>70%	Similarity > 75%	Similarity>80%	Similarity>85%	Similarity>90%	Similarity>95%
Accuracy	0.9815	0.9832	0.9863	0.9899	0.9916	0.9931
Precision	0.9911	0.9927	0.9919	0.9931	0.9935	0.994
Sensitivity	0.9717	0.9736	0.9805	0.9866	0.9896	0.9922
F1Score	0.9813	0.9831	0.9862	0.9899	0.9916	0.9931
MCC	0.9632	0.9666	0.9726	0.9798	0.9832	0.9862
Markedness_Measure	0.9634	0.9668	0.9726	0.9798	0.9832	0.9862
Youden's_J_Statistic	0.963	0.9664	0.9725	0.9798	0.9831	0.9862
Fowlkes–Mallows_Index	0.9814	0.9831	0.9862	0.9899	0.9916	0.9931

TABLE 10. PermGuard performance comparison with related works

Android Malware Detection Technique	Benign Apps	Malicious Apps	Total Apps	Accuracy	Precision	Recall	F1-measure
Wajahat et al. [26]	9476	5560	15036	0.979	0.982	0.976	0.979
Almarshad et al. [27]	9476	5560	15036	0.994	0.989	0.988	0.987
Alani and Awad [30]	9476	5560	15036	0.9798	0.981	0.9759	0.9783
Mehtab et al. [37]	510	910	1420	0.9911	0.9933	0.9936	0.9934
Li et al. [39]	10000	10000	20000	0.9828	0.9961	0.9773	0.9866
Liu et al. [40]	4043	12552	16595	0.9807	0.9931	0.9812	0.9871
PermGuard	55911	55911	111822	0.9933	0.9934	0.9932	0.9933

records skipped by the model at each similarity threshold level, ranging from >70% to >95%.

At a 70% similarity threshold, the highest number of records is skipped, totaling 58,123. This indicates that when the threshold for considering new data is lower, the model identifies a larger amount of incoming data as redundant, resulting in more skipped records. This behavior helps conserve computational resources but may come at the cost of missing subtle, yet important, new variations in malware data.

As the similarity threshold increases, the number of skipped records progressively decreases. For instance, at a 75% threshold, the total skipped records drop to 47,174, and further to 42,739 at an 80% threshold. This pattern continues with 33,782 records skipped at an 85% threshold, 26,102 at a 90% threshold, and finally, only 14,059 records are skipped at the highest threshold of 95%.

Lower thresholds result in more skipped records, conserving resources but potentially missing important data, while higher thresholds reduce the number of skipped records, enhancing model performance at the cost of increased computational effort.

In this training strategy, the researchers compare new data records with only the last 100 records to balance computational efficiency and model accuracy. Calculating cosine

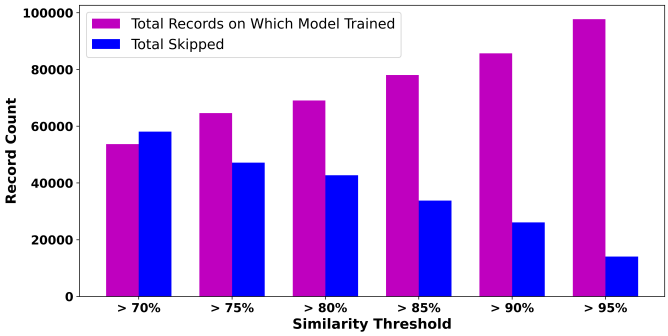

FIGURE 6. Total Records Skipped vs. Similarity Threshold

similarity is computationally intensive. Comparing each new record with all previous records would significantly increase processing time and resource consumption. By limiting the comparison to the last 100 records, the researchers reduce the computational burden, making the model more scalable and efficient. In practical scenarios, such as malware detection in Android applications, real-time performance is crucial. Rapidly evaluating and updating the model ensures timely detection of new threats without delaying system performance. The chosen approach helps maintain this balance. The most recent 100 records are likely to reflect the latest

trends and patterns in malware behavior. Focusing on these records ensures that the model stays up-to-date with the most current threat landscape, which is essential for effective malware detection.

F. EVALUATING RESILIENT AGAINST ADVERSARIAL MANIPULATIONS

In this section, we evaluate the performance of PermGuard on artificially generated synthetic dataset. The primary aim is to assess the model's robustness and its resilient against adversarial manipulations. This helps in identifying potential weaknesses that adversarial attacks could exploit. The authors generated 20000 synthetic data from gretel.ai and evaluated the trained model on this synthetic data.

Here's the line graph comparing the performance metrics of PermGuard on the real and synthetic datasets. The metrics include Accuracy, Precision, Sensitivity, F1 Score, and MCC.

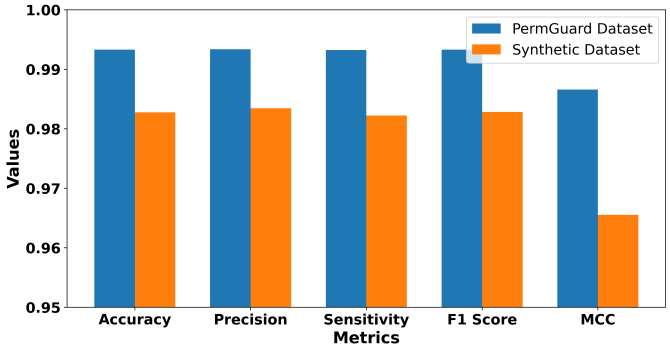

FIGURE 7. Comparison of Model Performance on PermGuard and Synthetic Datasets

The model achieved an accuracy of 0.9828 on the synthetic dataset while accuracy on the real dataset is 0.9933. The graph plotted in Figure 7 shows a slight decrease when tested on the synthetic dataset, indicating robust generalization capabilities and minimal performance loss, thereby validating the model's resilient against adversarial attacks.

By periodically testing the model against AI generated synthetic data, we can monitor its robustness over time. This continuous evaluation helps in proactively identifying and mitigating new vulnerabilities that adversarial attackers might exploit. This ongoing evaluation process is key to building resilient machine learning models capable of withstanding adversarial attacks while maintaining high performance.

VIII. CONCLUSION

In conclusion, this research presents a scalable Android malware detection framework, PermGuard, which leverages machine learning to effectively combat the rising threat of Android malware. The framework includes a dataset construction technique that collects 55,911 benign apps and 55,911 malware apps, mapping Android permissions into exploitation techniques. This mapping significantly reduces the feature size, enabling a comprehensive understanding of permission misuse by malware.

PermGuard introduces a similarity-based selective training approach to reduce the data required for incremental model development, enhancing overall training efficiency. It employs a test-then-train approach to ensure robustness and accuracy, initially testing the model on various datasets to identify weaknesses and refine the training process. Evaluated for its resilience against adversarial attacks, PermGuard can withstand attempts to deceive or bypass its detection mechanisms, enhancing security. Designed to be scalable, it handles large and continuously growing datasets, essential for real-world applications. Empirical results demonstrate that PermGuard achieved an accuracy of 0.9933 on real datasets and 0.9828 on synthetic datasets, showing strong resilience against both real and adversarial attacks.

Despite its strengths, PermGuard has limitations. Its reliance on permission-based features may not capture the complexity of advanced malware that exploits vulnerabilities without explicit permission misuse. It lacks behavioral analysis, which could provide deeper insights into malware actions, and does not use emerging techniques such as generative AI to analyze the entire source code, potentially missing subtle patterns and threats. Future research can address these limitations by incorporating behavioral analysis using sandboxing techniques to understand malware actions beyond permission misuse. Integrating generative AI models to analyze the entire source code with large language models (LLMs) could enhance detection capabilities, improving accuracy and resilience against sophisticated malware threats.

DATASET AVAILABILITY

The PermGuard data set for Android malware detection is available for download at IEEE DataPort (https://dx.doi.org/10.21227/d744-tb96).

REFERENCES

[1] StatCounter, "Mobile Operating System Market Share Worldwide," StatCounter. Accessed: Sep. 25, 2024. [Online]. Available: https://gs.statcounter.com/os-market-share/mobile/worldwide

[2] Kaspersky, "Turkey, Russia, Southeast Asia and Latin America hit by Android threats, Kaspersky identifies," Kaspersky. Accessed: Sep. 26, 2024. [Online]. Available: https://www.kaspersky.com/about/press-releases/turkey-russia-southeast-asia-and-latin-america-hit-by-android-threats-kaspersky-identifies

[3] S. Fadilpašić. "Dangerous new Android malware infects 11 million devices — here's what we know," TechRadar. Accessed: Sep. 26, 2024. [Online]. Available: https://www.techradar.com/pro/security/dangerous-new-android-malware-infects-11-million-devices-here-s-what-we-know

[4] D. Curry, "Business of Apps, Android statistics," Business of Apps. Accessed: Sep. 26, 2024. [Online]. Available: https://www.businessofapps.com/data/android-statistics/.

[5] ST. Sutter, T. Kehrer, M. Rennhard, B. Tellenbach, and J. Klein, "Dynamic Security Analysis on Android: A systematic literature review," IEEE Access, vol. 12, pp. 57261–57287, 2024. doi: https://doi.org/10.1109/ACCESS.2024.3390612

[6] H. Zhu, H. Wei, L. Wang, Z. Xu, and V. S. Sheng, "An effective end-to-end android malware detection method," Expert Systems With Applications, vol. 218, p. 119593, Jan. 2023. doi: https://doi.org/10.1016/j.eswa.2023.119593

[7] C. E. Rubio-Medrano et al., "DyPolDroid: Protecting against permission-abuse attacks in Android," Information Systems Frontiers, Oct. 2022. doi: https://doi.org/10.1007/s10796-022-10328-8

[8] P. Arntz, "New variant of Android SpyJoker malware removed from Play Store after 3 million installs," Malwarebytes Labs. Accessed: Sep. 26, 2024. [Online]. Available: https://www.malwarebytes.com/blog/news/2022/07/new-variant-of-android-spyjoker-malware-removed-from-play-store-after-3-million-installs. Accessed: Jun. 22, 2024.

[9] H. H. R. Manzil and S. M. Naik, "Detection approaches for android malware: Taxonomy and review analysis," Expert Systems With Applications, vol. 238, p. 122255, Oct. 2023. doi: https://doi.org/10.1016/j.eswa.2023.122255

[10] K. Sun, "BankBot found on Google Play, targets ten new UAE banking apps," Trend Micro Security Intelligence. Accessed: Sep. 26, 2024. [Online]. Available: https://www.trendmicro.com/en$_i$n/research/17/i/bankbot-found-google-play-targets-ten-new-uae-banking-apps.html. Accessed: Jun. 24, 2024.

[11] D. Bisson, "What is Ghimob malware?." Security Intelligence. Accessed: Sep. 26, 2024. [Online]. Available: https://securityintelligence.com/articles/what-is-ghimob-malware/. Accessed: Jun. 24, 2024.

[12] J. Kaur, U. Garg, and G. Bathla, "Detection of cross-site scripting (XSS) attacks using machine learning techniques: a review," Artificial Intelligence Review, vol. 56, no. 11, pp. 12725–12769, Mar. 2023. doi: https://doi.org/10.1007/s10462-023-10433-3

[13] A. Prasad and S. Chandra, "VMFCVD: An Optimized Framework to Combat Volumetric DDoS Attacks using Machine Learning," Arabian Journal for Science and Engineering, Jan. 2022. doi: https://doi.org/10.1007/s13369-021-06484-9

[14] J. Kaur, A. Agrawal, and R. A. Khan, "P2ADF: a privacy-preserving attack detection framework in fog-IoT environment," International Journal of Information Security, vol. 22, no. 4, pp. 749–762, Jan. 2023. doi: https://doi.org/10.1007/s10207-023-00661-7

[15] A. Prasad and S. Chandra, "BotDefender: A Collaborative Defense Framework Against Botnet Attacks using Network Traffic Analysis and Machine Learning," Arabian Journal for Science and Engineering, vol. 49, no. 3, pp. 3313–3329, Jun. 2023. doi: https://doi.org/10.1007/s13369-023-08016-z

[16] F. Mohsen, U. Rauf, V. Lavric, A. Kokushkin, Z. Wei, and A. Martinez, "On identification of intrusive applications: A step towards heuristics-based adaptive security policy," IEEE Access, vol. 12, pp. 37586–37599, Jan. 2024, doi: https://doi.org/10.1109/ACCESS.2024.3373202.

[17] R. Vaish, U. D. Dwivedi, S. Tewari, and S. M. Tripathi, "Machine learning applications in power system fault diagnosis: Research advancements and perspectives," Engineering Applications of Artificial Intelligence, vol. 106, p. 104504, Oct. 2021. doi: https://doi.org/10.1016/j.engappai.2021.104504

[18] P. Kumari, A. K. Jain, A. Seth, and N. Raghav, "Leveraging blockchain and machine learning to counter DDoS attacks over IoT network," Multimedia Tools and Applications, Mar. 2024, doi: https://doi.org/10.1007/s11042-024-18842-4

[19] A. Dahiya, S. Singh, and G. Shrivastava, "Android malware analysis and detection: A systematic review," Expert Systems, Oct. 2023. doi: https://doi.org/10.1111/exsy.13488

[20] S. Tewari, A. Prasad, H. Patel, M. Uddin, T. Al-Shehari and N. A. Alsadhan, "A novel multiagent collaborative learning architecture for automatic recognition of mudstone rock facies," IEEE Access, pp. 1–1, 2024. doi: https://doi.org/10.1109/ACCESS.2024.3507569

[21] F. Nawshin, D. Unal, M. Hammoudeh, and P. N. Suganthan, "AI-powered malware detection with Differential Privacy for zero trust security in Internet of Things networks," Ad Hoc Networks, vol. 161, p. 103523, Apr. 2024. doi: https://doi.org/10.1016/j.adhoc.2024.103523

[22] A. Manikandaraja, P. Aaby, and N. Pitropakis, "Rapidrift: Elementary Techniques to Improve Machine Learning-Based Malware Detection," Computers, vol. 12, no. 10, p. 195, Sep. 2023. doi: https://doi.org/10.3390/computers12100195

[23] A. Guerra-Manzanares, "Machine learning for Android malware detection: mission accomplished? A comprehensive review of open challenges and future perspectives," Computers & Security, vol. 138, p. 103654, Mar. 2024. doi: https://doi.org/10.1016/j.cose.2023.103654

[24] T. Sutter, T. Kehrer, M. Rennhard, B. Tellenbach, and J. Klein, "Dynamic security analysis on Android: A systematic literature review," IEEE Access, vol. 12, pp. 57261–57287, Jan. 2024. doi: https://doi.org/10.1109/ACCESS.2024.3390612

[25] 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, doi: https://doi.org/10.3390/su142114406. doi: https://doi.org/10.3390/su142114406

[26] A. Wajahat, J. He, N. Zhu, T. Mahmood, A. Nazir, F. Ullah, et al., "Securing Android IoT devices with GuardDroid transparent and lightweight malware detection", Ain Shams Eng. J., vol. 15, no. 5, May 2024. doi: https://doi.org/10.1016/j.asej.2024.102642

[27] F. A. Almarshad, M. Zakariah, G. A. Gashgari, E. A. Aldakheel, and A. I. A. Alzahrani, "Detection of Android Malware Using Machine Learning and Siamese Shot Learning Technique for Security," IEEE Access, vol. 11, pp. 127697–127714, Jan. 2023. doi: https://doi.org/10.1109/ACCESS.2023.3331739

[28] S. Aurangzeb and M. Aleem, "Evaluation and classification of obfuscated Android malware through deep learning using ensemble voting mechanism," Scientific Reports, vol. 13, no. 1, Feb. 2023. doi: https://doi.org/10.1038/s41598-023-30028-w

[29] E. Odat and Q. M. Yaseen, "A Novel Machine Learning Approach for Android Malware Detection Based on the Co-Existence of Features," IEEE Access, vol. 11, pp. 15471–15484, Jan. 2023. doi: https://doi.org/10.1109/ACCESS.2023.3244656

[30] M. M. Alani and A. I. Awad, "PAIRED: An explainable lightweight Android malware detection system," IEEE Access, vol. 10, pp. 73214–73228, Jan. 2022. doi: https://doi.org/10.1109/ACCESS.2022.3189645

[31] H. Alkahtani and T. H. H. Aldhyani, "Artificial Intelligence Algorithms for Malware Detection in Android-Operated Mobile Devices," Sensors, vol. 22, no. 6, p. 2268, Mar. 2022. doi: https://doi.org/10.3390/s22062268

[32] I. Sharafaldin, A. H. Lashkari and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization", In 2018 International Carnahan conference on security technology (ICCST) (pp. 1-7). IEEE.

[33] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," in Proc. Network and Distributed System Security Symposium (NDSS), 2014, pp. 1–15.

[34] R. Gupta, K. Sharma, and R. K. Garg, "Innovative Approach to Android Malware Detection: Prioritizing Critical Features Using Rough Set Theory," Electronics, vol. 13, no. 3, p. 482, Jan. 2024. doi: https://doi.org/10.3390/electronics13030482

[35] L. N. Vu and S. Jung, "AdMat: A CNN-on-Matrix Approach to Android Malware Detection and Classification," IEEE Access, vol. 9, pp. 39680–39694, 2021. doi: https://doi.org/10.1109/ACCESS.2021.3063748

[36] R. Surendran, T. Thomas, and S. Emmanuel, "GSDroid: Graph Signal Based Compact Feature Representation for Android Malware Detection," Expert Systems with Applications, vol. 159, p. 113581, Nov. 2020. doi: https://doi.org/10.1016/j.eswa.2020.113581

[37] A. Mehtab, W.B. Shahid, T. Yaqoob, M.F. Amjad, H. Abbas, H. Afzal, et al., "AdDroid: Rule-Based Machine Learning Framework for Android Malware Analysis", Mobile Netw Appl, vol. 25, pp. 180-192, 2020. doi: https://doi.org/10.1007/s11036-019-01248-0

[38] A. Mahindru and A. L. Sangal, "MLDroid—framework for Android malware detection using machine learning techniques," Neural Computing and Applications, Sep. 2020. doi: https://doi.org/10.1007/s00521-020-05309-4

[39] X. Li, L. Liu, Y. Liu, and H. Liu, "Detecting Android malware: A multimodal fusion method with fine-grained feature," Information Fusion, vol. 114, pp. 102662–102662, Sep. 2024. doi: https://doi.org/10.1016/j.inffus.2024.102662

[40] Z. Liu, R. Wang, N. Japkowicz, Heitor Murilo Gomes, B. Peng, and W. Zhang, "SeGDroid: An Android malware detection method based on sensitive function call graph learning," Expert systems with applications, vol. 235, pp. 121125–121125, Jan. 2024. doi: https://doi.org/10.1016/j.eswa.2023.1211254.

[41] A. Prasad and S. Chandra, "Machine learning to combat cyberattack: a survey of datasets and challenges," The Journal of Defense Modeling and Simulation Applications Methodology Technology, vol. 20, no. 4, pp. 577–588, May 2022. doi: https://doi.org/10.1177/15485129221094881

[42] M. M. Alani and A. I. Awad, "AdStop: Efficient flow-based mobile adware detection using machine learning," Computers & Security, vol. 117, p. 102718, Jun. 2022. doi: https://doi.org/10.1016/j.cose.2022.102718

[43] M. Injadat, A. Moubayed, A. B. Nassif, and A. Shami, "Multi-Stage Optimized Machine Learning Framework for Network Intrusion Detection," IEEE Transactions on Network and Service Management, pp. 1–1, 2020. doi: https://doi.org/10.1109/TNSM.2020.3014929

[44] M. M. Rahman, M. S. Chowdhury, M. Shorfuzzaman, L. Karim, Md Shafiullah, and Farag Azzedin, "Enhancing Septic Shock Detection through Interpretable Machine Learning," Computer Modeling in Engineering & Sciences, vol. 0, no. 0, pp. 1–10, Jan. 2024. doi: https://doi.org/10.32604/cmes.2024.055065

[45] I. Almomani, T. Almashat, and W. El-Shafai, "Maloid-DS: Labeled Dataset for Android Malware Forensics," IEEE Access, vol. 12, pp. 73481–73546, 2024, doi: https://doi.org/10.1109/access.2024.3400211. doi: https://doi.org/10.1109/ACCESS.2024.3400211

[46] K. Allix, T. F. Bissyandé, J. Klein and Y. L. Traon, "AndroZoo: Collecting millions of Android apps for the research community", Proc. IEEE/ACM 13th Work. Conf. Mining Softw. Repositories (MSR), pp. 468-471, May 2016. doi: https://doi.org/10.1145/2901739.2903508

[47] Y. Tsutano, "axmldec." GitHub. Accessed: Sep. 26, 2024. [Online]. Available: https://github.com/ytsutano/axmldec

[48] A. Prasad, S. Chandra, Ibrahim Atoum, N. Ahmad, and Yazeed Alqahhas, "A collaborative prediction approach to defend against amplified reflection and exploitation attacks," Electronic Research Archive, vol. 31, no. 10, pp. 6045–6070, Jan. 2023. doi: https://doi.org/10.3934/era.2023308

[49] R. Verma and S. Chandra, "RepuTE: A soft voting ensemble learning framework for reputation-based attack detection in fog-IoT milieu," Engineering Applications of Artificial Intelligence, vol. 118, p. 105670, Feb. 2023. doi: https://doi.org/10.1016/j.engappai.2022.105670

[50] G. Xu, H. Shao, J. Cui, H. Bai, J. Li, G. Bai, S. Liu, W. Meng, and X. Zheng, "Gendroid: A query-efficient black-box android adversarial attack framework," Computers & Security, p. 103359, 2023. doi: https://doi.org/10.1016/j.cose.2023.103359

[51] C. S. Yadav, A. Yadav, H. S. Pattanayak, R. Kumar, A. A. Khan, M. A. Haq, et al., "Malware Analysis in IoT & Android Systems with Defensive Mechanism", Electronics, vol. 11, no. 15, pp. 2354, 2022. doi: https://doi.org/10.3390/electronics11152354

[52] A. Daniel, R. Deebalakshmi, R. Thilagavathy, T. Kohilakanagalakshmi, S. Janakiraman, and Balamurugan Balusamy, "Optimal feature selection for malware detection in cyber physical systems using graph convolutional network," Computers & Electrical Engineering, vol. 108, pp. 108689–108689, Apr. 2023. doi: https://doi.org/10.1016/j.compeleceng.2023.108689

[53] M. Mudassir, D. Unal, M. Hammoudeh, and F. Azzedin, "Detection of Botnet Attacks against Industrial IoT Systems by Multilayer Deep Learning Approaches," Wireless Communications and Mobile Computing, vol. 2022, pp. 1–12, May 2022. doi: https://doi.org/10.1155/2022/2845446

[54] A. Mathur, L. M. Podila, K. Kulkarni, Q. Niyaz, and A. Y. Javaid, "NATICUSdroid: A malware detection framework for Android using native and custom permissions," Journal of Information Security and Applications, vol. 58, p. 102696, May 2021. doi: https://doi.org/10.1016/j.jisa.2020.102696

[55] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," 2012 IEEE Symposium on Security and Privacy, May 2012. doi: https://doi.org/10.1109/SP.2012.16

[56] S. Mahdavifar, A. F. Abdul Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning," In 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech) (pp. 515-522). IEEE. doi: https://doi.org/10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094

[57] C. Cop, "Android malware detection," Kaggle. Accessed: Sep. 27, 2024. [Online]. Available:https://www.kaggle.com/datasets/subhajournal/android-malware-detection

[58] I. Sharafaldin, A. Habibi Lashkari and A. A. Ghorbani, "Toward generating a new intrusion detection dataset and intrusion traffic characterization," 4th International Conference on Information Systems Security and Privacy (ICISSP), Portugal, 1, 108-116.

[59] A. Prasad and S. Chandra, "PhiUSIIL: A diverse security profile empowered phishing URL detection framework based on similarity index and incremental learning," Computers & Security, p. 103545, Oct. 2023. doi: https://doi.org/10.1016/j.cose.2023.103545

ARVIND PREASAD received the Ph.D. degree in Computer Science from Babasaheb Bhimrao Ambedkar University (a central university), Lucknow, with a focus on cybersecurity and machine learning. He is currently serving as an Assistant Professor in the Department of CEA at GLA University, Mathura. Prior to this role, he contributed significantly to academia during his tenure as a lecturer at King Saud University, Riyadh, from 2010 to 2019. His research interests include Cybersecurity, Reverse Engineering, Malware Analysis, Network Traffic Analysis, and Machine Learning. His work is dedicated to developing innovative solutions addressing the dynamic challenges of securing cyberspace.

SHALINI CHANDRA is working as Assistant Professor in the Department of Computer Science, Babasaheb Bhimrao Ambedkar University (A Central University), Lucknow, UP. Her research areas are Fog Computing and Cyber Security, Software Security, and Software Quality.

MUEEN UDDIN received his Ph.D. degree from the Universiti Teknologi Malaysia (UTM), in 2013. He is currently working as an Associate Professor of Cybersecurity and Data Sciences at the University of Doha for Science and Technology Qatar. He has published over 170 international journals and conference papers in highly reputed journals with a cumulative impact factor of over 300. His research interests include Blockchain, Cybersecurity, IoT, Network Security and Cloud Computing.

TAHER AL-SHEHARI received the B.S. degree in Computer Science from King Khalid University, Saudi Arabia, in 2007 and the M.S. degree in Computer Science from King Fahd University of Petroleum and Minerals (KFUPM), in 2014. From 2011 to 2014, he was a Research Assistant at King Fahd University of Petroleum and Minerals. Since 2015, he is working as a senior lecturer and researcher at King Saud University. His research interests include information security and privacy, insider threat detection and prevention systems, machine learning models and data analysis. His awards and honors include an Honor Award from King Khalid University's Rector, and Best Designed Curriculum Award from CFY's Dean, KSU. He is the author of several papers that are published in prestige journals.

NASSER ALSADHAN is a distinguished member of the IEEE community, known for his significant contributions to the field of computer and information sciences. With an affiliation to the College of Computer and Information Sciences at King Saud University in Riyadh, Saudi Arabia. Dr. Alsadhan has been instrumental in advancing research in various areas of technology.

His work has spanned a range of topics, including prediction accuracy, affective states, and big data analytics. Dr. Alsadhan's expertise in convolutional neural networks and language models has led to innovative developments in understanding emotion words and personality traits through text analysis.

SYED SAJID ULLAH Syed Sajid Ullah attained his master's degree in computer science from Hazara University, Pakistan, in 2020. Currently, he is pursuing his Ph.D. at the Department of Information and Communication Technology, University of Agder (UiA), located in Grimstad, Norway. He actively contributes as a reviewer for over 30 esteemed journals and serves on the editorial boards of multiple reputable publications. With a prolific track record, he has authored more than 90 articles across various high-impact journals. Additionally, he plays a pivotal role as a researcher in the NIST project focusing on quantum cryptography and Named Data Networking. His primary research interests span cryptography, blockchain, access control, post-quantum cryptography, network security, information-centric networking, named data networking, and the Internet of Things.