Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.048914

ARTICLE

MoBShield: A Novel XML Approach for Securing Mobile Banking

Saeed Seraj[1], **Ali Safaa Sadiq**[1,*], **Omprakash Kaiwartya**[1], **Mohammad Aljaidi**[2], **Alexandros Konios**[1], **Mohammed Ali**[3] **and Mohammed Abazeed**[3]

[1]Department of Computer Science, Nottingham Trent University, Clifton Lane, Nottingham, NG11 8NS, UK

[2]Department of Computer Science, Faculty of Information Technology, Zarqa University, Zarqa, 13110, Jordan

[3]Department of Computer Science, King Khalid University, Abha, 61421, Saudi Arabia

*Corresponding Author: Ali Safaa Sadiq. Email: ali.sadiq@ntu.ac.uk

Received: 21 December 2023 Accepted: 20 March 2024 Published: 15 May 2024

ABSTRACT

Mobile banking security has witnessed significant R&D attention from both financial institutions and academia. This is due to the growing number of mobile baking applications and their reachability and usefulness to society. However, these applications are also attractive prey for cybercriminals, who use a variety of malware to steal personal banking information. Related literature in mobile banking security requires many permissions that are not necessary for the application's intended security functionality. In this context, this paper presents a novel efficient permission identification approach for securing mobile banking (MoBShield) to detect and prevent malware. A permission-based dataset is generated for mobile banking malware detection that consists large number of malicious adware apps and benign apps to use as training datasets. The dataset is generated from 1650 malicious banking apps of the Canadian Institute of Cybersecurity, University of New Brunswick and benign apps from Google Play. A machine learning algorithm is used to determine whether a mobile banking application is malicious based on its permission requests. Further, an eXplainable machine learning (XML) approach is developed to improve trust by explaining the reasoning behind the algorithm's behaviour. Performance evaluation tests that the approach can effectively and practically identify mobile banking malware with high precision and reduced false positives. Specifically, the adapted artificial neural networks (ANN), convolutional neural networks (CNN) and XML approaches achieve a higher accuracy of 99.7% and the adapted deep neural networks (DNN) approach achieves 99.6% accuracy in comparison with the state-of-the-art approaches. These promising results position the proposed approach as a potential tool for real-world scenarios, offering a robust means of identifying and thwarting malware in mobile-based banking applications. Consequently, MoBShield has the potential to significantly enhance the security and trustworthiness of mobile banking platforms, mitigating the risks posed by cyber threats and ensuring a safer user experience.

KEYWORDS

Security; malware detection; deep learning; convolutional neural networks; deep neural networks

1 Introduction

Mobile banking has witnessed significant popularity in recent years, due to the growing mobile internet availability [1]. The malware attacks in mobile banking applications have increased considerably exploiting mobile device constraints. This banking malware can lead to huge financial losses for mobile users and financial institutions such as banks [2]. To better understand the mobile banking transaction, Fig. 1 illustrates the process of a secure transaction between a user and a banking server within the context of a mobile banking application. The diagram highlights the critical stages of this interaction, including user-initiated transaction requests and the exchange of data with the banking and cloud servers. Additionally, it showcases the mobile banking application's Android Packet (APK) permission files that could be extracted, which govern the access and authorization levels required for the application to execute various transaction-related operations securely. Fig. 1 provides a visual representation of the underlying mechanism for secure mobile banking transactions, emphasizing the importance of proper permissions and access control in safeguarding financial transactions.

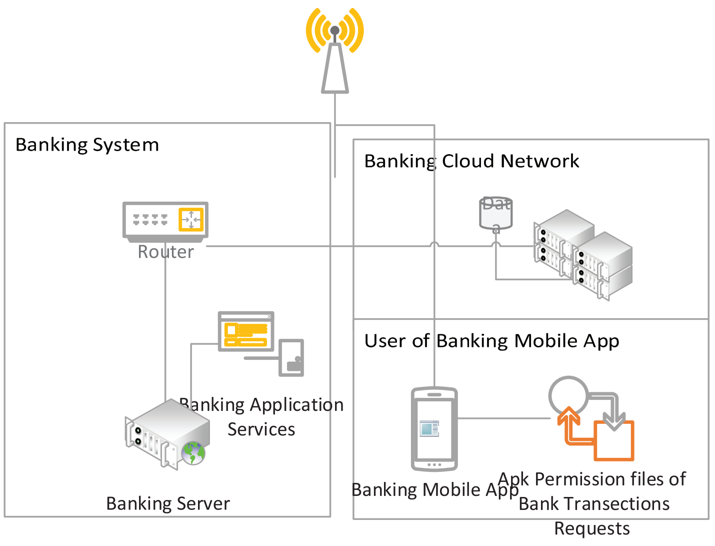

Banking System
Router
Banking Application Services
Banking Server
Banking Cloud Network
Dat a
User of Banking Mobile App
Banking Mobile App
Apk Permission files of Bank Transections Requests

Figure 1: User-banking server interaction and APK permission files in mobile banking

Towards going deep in different types of mobile banking malware-related concerns, trojans are specific malware pieces that appear to be legitimate applications but contain harmful malicious code that, when executed, causes severe damage to the device [3]. Trojanised mobile banking applications can control the browser and steal bank account details such as bank login information [4]. By rooting the device, the mobile viruses can access sensitive files and memory in an unauthorized manner. Banking Trojans are also spread through email attachments, drive-by downloads, pirated software, and corrupted USB drives. Banking Trojans can also be used for theft, identity theft, monitoring mobile users, and activities, deleting data, and communicating with an attacker's command-and-control server.

In order to combat mobile banking malware attacks via trojans, researchers have developed several methods for identifying malware that mainly targets banking applications. In this regard, machine learning-based methods, like neural networks, have shown promise in results in determining malicious code and analyzing permission requests from mobile applications [5]. However, the lack of transparency and interpretability of machine learning models are some major concerns, especially when dealing with critical banking applications [6]. This drawback has led to an interesting shift towards eXplainable ML (XML) techniques for mobile malware detection [7]. XML provides human-understandable explanations for the decisions made by ML models. By incorporating XML techniques into the detection of banking malware in Android apps, the transparency and interpretability of the malware detection process in banking applications can be improved [8]. This not only improves trust in the banking system but also allows security analysts to understand and validate the decisions made by the models. The main reason behind such improvement with the use of XML is that the use of permissions as a feature in classifying malware behaviour in mobile applications could be generated automatically, where explainability will play a crucial role.

In this context, this paper presents a neural network-based technique for detecting banking malware in Android apps. Our method involves analyzing permission requests from an application and using the data to train a neural network to detect malware. By incorporating XML techniques, the aim is to provide explanations for the decisions made by our model, thus enhancing the transparency and interpretability of the detection process. The effectiveness of the proposed method and presented the results were presented based on a set of Android malware samples. The utilised neural network algorithms were trained by analyzing Android malware samples and benign applications. The dataset contains features with 458 permissions, which they extracted as features for our neural networks. To train the model, deep neural network architectures with many layers of neurons are employed. Our Convolutional and dense layers are combined to learn complex representations of permission features. To avoid overfitting and enhance generalization performance, a variety of regularization techniques are employed. The effectiveness of the proposed method is assessed using a set of common metrics, such as accuracy, precision, recall, and F1 score. Overall, the proposed approach can be used to improve mobile banking application security and shield users from financial harm, while providing insights into the decision-making process of the detection system through the incorporation of XML techniques. The three main contributions of the paper are as follows:

- A realistic dataset with 458 mobile banking permissions is developed to use in detecting banking malware on mobile devices.
- We adapted and designed a new model based on convolutional neural networks (CNN), deep neural networks (DNN), and artificial neural networks (ANN) to detect banking malware.
- We evaluate the performance of the model using relevant metrics, with a critical analysis of results.

It is also important to mention that the primary motivation of this work is to tackle the increasing threat of malware attacks in mobile banking applications, specifically focusing on trojans that pose a significant risk to user and financial institution security. Besides, the proposed use of XML has motivated the enhancement in transparency and interpretability in mobile banking security.

The rest of this paper is organized as follows. Section 2 provides a critical review of related work in banking applications malware detection. Section 3 presents the details of the proposed framework MoBShield including the developed dataset, data generation approach, pre-processing of dataset, and feature reduction. Section 4 discusses the performance evaluation including metrics and experimental result analysis. Finally, Section 5 concludes the paper and highlights potential future research directions.

2 Related Works

It is important to highlight the need for an approach that combines effective malware detection with XML in the context of mobile banking applications. While previous research has focused on using machine learning techniques, such as neural networks, for detecting banking malware, the lack of transparency and interpretability has been a significant limitation.

Several studies have emphasized the importance of incorporating XML techniques to enhance the transparency and interpretability of AI systems in various domains. In [9], authors have proposed a framework for explaining the predictions of any classifier, addressing the question of "Why Should I Trust You?" Their work highlights the significance of providing explanations for the decisions made by AI models. In the context of XML, Barredo Arrieta et al. [10] conducted a comprehensive survey on Explainable Artificial Intelligence (XML), outlining concepts, taxonomies, opportunities, and challenges towards responsible AI. Their study emphasizes the need for transparent and interpretable AI models to build trust and facilitate decision-making. In another attempt, the authors in [11] have surveyed eXplainable Artificial Intelligence AI (XAI) and XML, delving into techniques and methods for peeking inside the black box of AI models. They discussed various approaches, such as rule-based methods, and model-agnostic methods that provide interpretability and transparency in AI systems. On the other hand, Miller, T. in [12] has explored the insights from the social sciences on the explanation of artificial intelligence. Their work highlights the interdisciplinary nature of XML, drawing from fields like cognitive science and psychology to understand how humans perceive and interact with explanations generated by AI models. By incorporating XML techniques into the detection process, we address the limitations of transparency and interpretability in previous research on banking malware detection. These references support the need for transparent and interpretable AI models, particularly in sensitive domains such as mobile banking, where the security and privacy of users' financial information are paramount.

Users frequently use mobile banking applications to manage their finances, but they are also susceptible to attacks from malware such as banking trojans. Numerous investigations have been made into the security risks connected with mobile banking applications, as well as methods for identifying and reducing these risks. A framework for the detection of Android Banking Trojans (ABT) was put forth by [13]. The framework analyses the characteristics of network traffic produced by the Trojans and identifies its patterns using machine learning algorithms. Similarly, the authors in [14] looked into the use of machine learning methods to identify network communication traffic from banking malware. The decision tree and random forest algorithms were successful in identifying the malware traffic after they compared various classification algorithms. References [15,16] conducted a study to identify patterns in the behaviours of banking malware. Keylogging and network communication with command-and-control servers were identified as common behaviours after an analysis of the traits of different banking malware types.

The early ZeuS trojan and the more recent Zitmo malware were both studied for trends in banking malware. In [17], the authors have compared the security of user data in native and cross-platform Android mobile banking applications. They compared the security attributes of a number of well-known mobile banking apps, and they discovered that cross-platform apps were more prone to attacks than native ones. Similarly, Chen et al. [18] empirically evaluated the security risks of international Android banking apps and discovered that numerous apps had issues with SSL pinning, insufficient encryption, and unsafe data storage. The repackaging attack on Android banking applications was examined by [19], who also suggested countermeasures to stop such attacks. They discovered that hackers could alter banking applications' source code to steal user data and put forth strategies for preventing and countering these attacks. Additionally, Koala et al. [20] examined the security flaws in Android-based mobile banking and payment applications used in African nations and suggested solutions.

Authors in [21] have researched Android banking application attacks and suggested defences against them. They examined the characteristics of various attack types, including keylogging and phishing, and they suggested techniques for identifying and counteracting them. The Man-in-the-Middle vulnerability in mobile banking applications for Android devices was also investigated by [22], who also suggested methods for identifying and addressing this vulnerability. The formal method for the detection of banking Trojans in Android was lastly put forth by [23,24]. To find the trojans in the applications' source code, their method combines static and dynamic analysis. Similarly, authors in [25] proposed formal methods for analysing and detecting Android banking malware. To find weaknesses and potential threats in the applications' code, their techniques employ model checking and symbolic execution. To evaluate vulnerabilities and conduct penetration tests on Android and iOS mobile banking apps, Abusharekh et al. [26] proposed APTAi, a threat model and security of mobile banking applications (SMB) in [27]. Their attack tree-based model, which aims to identify and reduce the security risks connected to mobile banking applications, was developed by the researchers.

To summarise the main research gaps with the current state of the arts along with the potential contribution of the proposed work, below are the main points to highlight:

- Despite extensive research on mobile banking security, critical gaps persist in integrating effective malware detection with eXplainable Machine Learning (XML).
- Previous studies focused on less transparent machine learning techniques, highlighting the need for transparency, especially in sensitive domains like mobile banking.
- The research aims to address these gaps by incorporating XML techniques, bridging traditional machine learning approaches with the demand for transparent AI models in safeguarding users' financial information.
- The developed dataset enhances the research's robustness, facilitating effective training and evaluation of the proposed malware detection approach.

Table 1 provides a comprehensive comparison of related works, highlighting their types, detection methods, detection targets, features, and limitations.

Table 1: Comparative analysis of the related works

Reference	Type	Detection method	Detection target	Features	Limitations
[18]	Hybrid	Comparative analysis	User data security	Comparative study of user data security in native and cross-platform Android mobile banking applications Utilising XML	Limited to analysing security measures in mobile banking applications only

(Continued)

Table 1 (continued)

Reference	Type	Detection method	Detection target	Features	Limitations
[19]	Dynamic	Dynamic analysis	Android banking malware	Analysed behaviour of Android banking malware on emulated Android devices	Limited to only 5 malware samples. Lack of XML utilisation
[20]	Hybrid	Static and dynamic analysis	Android banking applications	Analysed 20 popular Android banking applications. Utilising XML	Did not analyse malware samples
[21]	Static	Machine learning	Mobile banking malware	Used machine learning to detect malware on Android devices	Limited dataset used for training and testing. Lack of XML utilisation
[22]	Hybrid	Static and dynamic analysis	Mobile banking applications	Analysed 10 popular mobile banking applications	Did not analyse malware samples. Lack of XML utilization
[23]	Hybrid	Static and dynamic analysis	Android banking malware	Analysed a large dataset of malicious APK files	Did not include an analysis of benign apps. Lack of XML utilisation
[24]	Dynamic	Dynamic analysis	Android banking trojans	Analysed behaviour of banking trojans on Android devices	Limited to only 14 samples. Lack of XML utilisation
[25]	Dynamic	Dynamic analysis	Android banking malware	Analysed behaviour of Android banking malware on emulated Android devices	Limited to only 6 malware samples. Lack of XML utilisation

(Continued)

Table 1 (continued)

Reference	Type	Detection method	Detection target	Features	Limitations
[26]	Dynamic	Dynamic analysis	Android banking malware	Analysed behaviour of Android banking malware	Limited to only 10 malware samples. Lack of XML utilisation
[27]	Dynamic	Dynamic analysis	Android banking malware	Analysed behaviour of Android banking malware on emulated Android devices	Limited to only 10 malware samples. Lack of XML utilization
[28]	Hybrid	Comparative analysis	Mobile banking and payment applications	Vulnerability analysis in mobile banking and payment applications on Android in African countries. Utilising XML	Limited to analysing security measures in mobile banking applications only

3 MoBShield: A Novel Permission-Based XML Approach for Securing Mobile Banking

3.1 Dataset Development

We present a new dataset to identify and detect banking malware based on Android apps' permissions in the Android platform. As a result, we created a dataset with 3300 entries based on Android banking malware. To do this, we downloaded 1650 infected Android banking from third-party websites and 1650 benign apps of which 116 are legitimate banking apps and 1534 are from different categories from Google Play. To examine all apk files and extract app permissions, we used VirusTotal online scanner. In addition, we classified the apk files using over 70 trusted anti-malware detection engines. The dataset includes several banking malware families, some of which mainly are BankBot, Bankun, FakeBank, Marcher, SandRoid, SMSspy, SpitMo, Tebak, Wroba, Zitmo, Fakeinst, Misosms, Svpeng, Gepew, SMSkey, AVpass and WannaLocker. The list of Android banking malware families and the number of samples are listed in Table 2. We put all the information in a file to make the dataset usable in CSV file format, which is simple to open and process. The dataset contains 459 columns of which 458 are specific permissions and the label which is the last column. The initial row of the dataset describes column titles, and the remaining rows contain features from 3300 banking malware and benign applications. All values are in binary format, which means they are either '0' or '1'. When an app requires permission, the value in the corresponding dataset entry is '1', and when an app does not require permission, the value is '0'. Based on the VirusTotal report, an Android app recognised as malware by most antimalware companies is possibly risky. The respective value in the label column is set to ‘1’, indicating banking malware. Table 3 showcases a representative subset of the dataset.

Table 2: Banking malware families

Banking malware families	Year of discovery	Number of samples
BankBot	2017	160
Bankun	2014	47
FakeBank	2013	193
Marcher	2013	177
SMSspy	2012	76
SpitMo	2019	190
SVpeng	2013	58
Tebak	2013	63
Wannalocker	2016	7
Wroba	2021	269
Zitmo	2010	142
AVpass	2011	11
Fakeapp	2008	11
Fakeinst	2014	70
Gepew	2019	40
Misosms	2012	50
SMSkey	2000	43
Other families	From 2000 to 2021	45

Table 3: A sample of the developed banking malware dataset

INTERNET	ACCESS_COARSE_ LOCATION	ACCESS_FINE_ LOCATION	GET_TASKS	CHANE_WIFI _STATE
1	1	1	0	1
1	0	1	0	0
1	1	1	0	0
1	1	1	0	0
1	0	1	0	0

Banking Trojans represent a major cybersecurity threat, designed to steal financial information from infected devices. These malicious programs target online banking credentials, credit card data, and other sensitive information that can lead to financial fraud and identity theft. Banking Trojans are typically spread through phishing campaigns and drive-by downloads. Once installed, they monitor host device activity and siphon data to attackers. The following bullet list which is listed in Table 2 describes some of the most prevalent banking Trojan malware variants that infiltrate systems and stealthily steal financial data:

- BankBot: Trojan. Bankbot is a Trojan malware that is designed to steal sensitive financial data such as online banking credentials, credit card numbers, and other personal information.
- Bankun: Trojan. Bankun is a Trojan malware that steals sensitive financial information from infected devices. It typically infects devices via phishing emails, drive-by downloads, or other social engineering methods.
- Fakebank: Trojan. Fakebank is a Trojan malware that impersonates legitimate banking applications or websites to steal financial information from infected devices. It typically infects devices via phishing emails or other social engineering techniques, after which it installs itself.
- Marcher: Trojan. Marcher is a type of Android Trojan malware that steals financial information from infected devices. The malware typically infects devices via phishing emails or SMS messages, or it masquerades as a legitimate application.
- SMSSpy: Trojan. SMSSpy is a Trojan malware that monitors and steals SMS messages from mobile devices infected with it. The malware typically in-stalls itself after being downloaded via malicious apps or phishing emails.
- Spitmo: This Trojan malware steals two-factor authentication (2FA) codes from infected mobile devices. It usually spreads through phishing emails or SMS messages, or by masquerading as a legitimate app.
- SVpeng: This Android Trojan malware steals sensitive information from in-fected mobile devices. Devices are usually infected through malicious apps or phishing emails.
- Tebak: The Trojan. Tebak virus is a type of malware that steals sensitive data from infected computers. It is also known as the "Windows Tebak Trojan."
- Wannalocker: The Trojan. Wannalocker ransomware, also known as Wan-naLocker, is a type of Android ransomware. In most cases, it is spread by malicious apps or phishing emails.
- Wroba: Trojan. Wroba is an Android Trojan that steals sensitive information from infected mobile devices. Phishing emails and malicious apps are com-mon ways for it to spread.
- Zitmo: Mobile malware called Zitmo, or Zeus in the Mobile, is a type of malware that infects mobile devices. The malware intercepts SMS messages containing one-time passwords (OTPs) or other authentication codes sent by banks to customers to steal banking information.
- AVpass: The Trojan AVpass is a type of malware designed to avoid detection by antivirus software through various obfuscation techniques. Most commonly it is spread using malicious email attachments or software downloaded from infected websites.
- Fakeapp: It is an example of malware that imitates legitimate apps in an at-tempt to trick users into installing them. Typically, malware is distributed through bogus app stores or phishing emails containing links to download it.
- Fakeinst: The Trojan.FakeInst malware, also called FakeInst, pretends to be legitimate software installers to trick users into installing it. Typically, malware spreads through malicious websites or spam emails containing download links.
- Gepew: Gepew is Trojan malware that steals sensitive data from infected computers. Typically, it infiltrates a computer via a malicious email attachment or software download from a compromised website.
- Misosms: Misosms are a type of Trojan horse.MisoSMS is a type of malware that infects Android devices and sends unauthorised text messages to premium-rate phone numbers without the user's knowledge or consent.
- SMSkey: SMSkey is a Trojan.SMSKey is a type of malware that infects Android devices and steals sensitive data such as login credentials and financial information. Trojan.SMSKey, once installed on an Android device, may employ a variety of methods to steal sensitive information.

3.2 Dataset Generation

The proposed method is divided into stages, and Fig. 2 demonstrates an overview of our proposed schematic representation of the methodology architecture. Furthermore, our proposed methodology is detailed below. The process of creating a dataset of malicious banking apps is an important step in training machine learning models to detect and prevent malware. Let us examine each of the four stages of this process in greater detail.

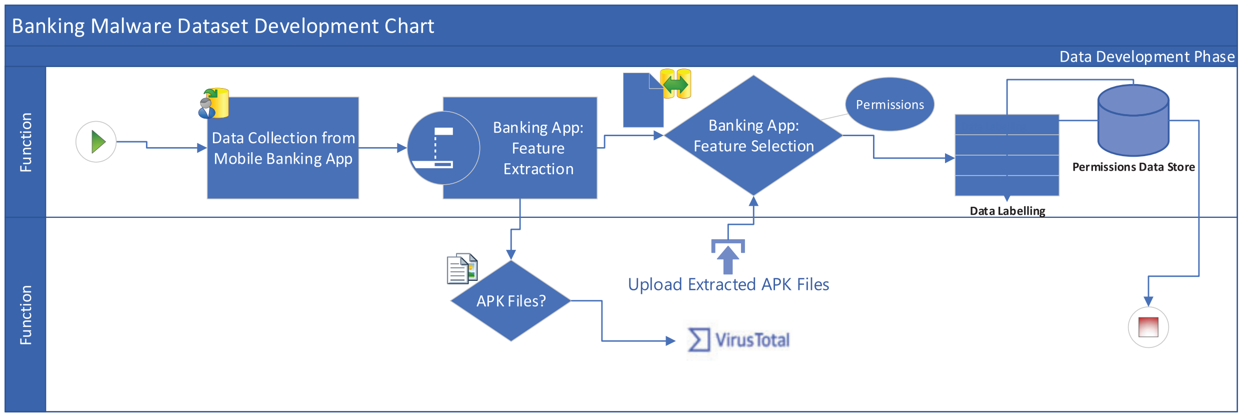

Banking Malware Dataset Development Chart
Data Development Phase
Function
Function
Data Collection from Mobile Banking App
Banking App: Feature Extraction
Banking App: Feature Selection
Permissions
Permissions Data Store
Data Labelling
APK Files?
Upload Extracted APK Files
VirusTotal

Figure 2: Schematic representation of the methodology for constructing the advanced banking Malware dataset

3.2.1 Data Collection

This stage involves collecting a large number of malicious adware apps and benign apps to use as a training dataset. In this particular case, we collected 1650 malicious banking apps from the Canadian Institute of Cybersecurity, University of New Brunswick and 1650 benign apps from Google Play. Using a large dataset is important because it allows the machine learning model to learn from various examples and make more accurate predictions. The emphasis on a large dataset is crucial as it provides diverse examples for the machine learning model, enabling it to learn more effectively and make more accurate predictions. This diversity is essential for training a robust model capable of recognising the nuances between malicious and benign applications in the context of mobile banking security.

3.2.2 Feature Extraction

Once the apps have been collected, the next step is to extract features that can be used to train the machine learning model. This stage collects static features without executing the code. Static features include things like the app's name, version, and permissions requested. In this research, we used the VirusTotal online scanner to extract permissions by uploading an apk file to the website and adding every permission to their dataset. This involved uploading the apk files to the scanner, which systematically compiled a dataset by incorporating every permission associated with the apps. This method ensures a thorough extraction of relevant features crucial for training the machine learning model effectively.

3.2.3 Feature Selection

After the features have been extracted, the next step is to select the most relevant features to use for training the machine learning model. This is important because irrelevant and redundant features can degrade the quality and accuracy of the model, and higher-dimensional datasets require more storage space and computation time. In this study, we chose relevant features such as permissions and activities that are important in detecting malicious Android apps. Permissions are especially important because they protect users' privacy and sensitive data, and apps must request permission to access these features.

3.2.4 Data Labelling

Labelling each app as malicious or benign completes the dataset. This is significant because it enables the machine learning model to study both kinds of apps and learn from them in order to produce more precise predictions. In this study, we manually categorised the apps by reviewing each one and classifying them as malicious or benign based on their behaviour.

In general, gathering a large number of apps, extracting pertinent features, choosing the most crucial features, and classifying each app as malicious or benign are the steps involved in creating a dataset of banking malware. In order to ensure the security of mobile devices and safeguard users' sensitive data, this dataset can then be used to train machine learning models to identify and stop malware.

3.3 Preprocessing of Dataset

Preprocessing entails converting raw data into a format appropriate for analysis or modelling. It is a critical step in data analysis and machine learning. Preprocessing is the process of preparing data in order to increase the effectiveness and precision of the machine learning algorithms and data analysis techniques that will be applied. Preprocessing typically entails a number of steps, such as, Data cleaning which entails locating and dealing with incorrect or missing data, getting rid of duplicates, and handling outliers or inconsistencies, and data transformation which entails transforming the data into a format that is appropriate for analysis, such as feature scaling, standardisation, normalisation of the data, or converting categorical variables to numerical values. Preprocessing is a crucial step in the workflow for data analysis and machine learning because it has a big impact on how accurate and effective the algorithms are. Better outcomes and more precise predictions can be obtained by making sure the data is in the right format, has been correctly cleaned and transformed, and only contains the pertinent features.

3.3.1 Missing Value Check

Missing values are a common issue in datasets, and they can occur for a number of reasons, including data entry mistakes or problems with data collection. Conducting a missing value check is the first step in handling missing values. Finding the null or missing values in the dataset is required for this. In this instance, all of the NAN data in each column of the dataset were identified by the author using the Python NumPy library. Choosing whether to replace or delete the missing values comes next after they have been located. The average value of the column can be used to fill in the missing values, which is a common strategy. This preserves the overall distribution of the data and reduces the impact of missing values on the analysis. In some cases, completely removing rows with missing values may be appropriate. If there are too many missing values or if they appear in critical variables, this strategy may be required. Handling missing values is a critical step in data cleaning and preparation in general. By correctly identifying and handling missing values, analysts can ensure the accuracy and dependability of their analyses.

3.3.2 Data Type Conversion

Data type conversion is the process of changing the format of data, such as converting a string to a number or a date to a timestamp. It is critical to use the correct data types when conducting data analysis in order to avoid errors and produce realistic results. Mathematical operations on the data may be impossible, for example, if a numerical column is mistakenly identified as a string column. To perform data type conversion, the author used the Pandas library, a well-known Python data manipulation library. Pandas' 'as type' function can be used to convert a column to a specific data type. Pandas also include additional functions for converting data types. By performing data type conversion, we can ensure that the dataset is in the correct format for analysis and that errors and inconsistencies are kept to a minimum. Data type conversion, which is an important step in data cleaning and preparation, can have a significant impact on the accuracy and dependability of data analysis.

3.4 Feature Reduction

Feature reduction is a technique used in machine learning and data analysis that involves reducing the number of features or variables in a dataset while keeping the most important and relevant data. When datasets have a lot of features or variables, which happens frequently, the dimensionality problem can appear. This may cause overfitting, slower computation times, and poorer predictive performance. By identifying the most informative features and eliminating the redundant or irrelevant ones, feature reduction techniques seek to resolve these problems. Techniques for reducing the number of features can increase the precision and effectiveness of machine learning algorithms, especially when there are a lot of features. These methods can lessen overfitting, increase generalisation, and improve the interpretability of models by only choosing the most crucial features. The code that is being offered removes columns from the dataset according to a predetermined standard. In this instance, the criterion is that a column is dropped if the mean value of the data points in that column that are equal to 0 is greater than or equal to 0.75.

This criterion is based on the idea that a column may not provide useful information for the analysis and may even be redundant if a significant portion of its values are zero. By eliminating these columns, the code streamlines the dataset, lowers the problem's dimensionality, and may even enhance model performance and computational effectiveness. Out of the original 458 features, 438 columns have been removed by the code in this case, greatly reducing the dimensionality of the dataset and the most important 20 kept which are described in Table 4. This reduction can help to increase the precision and effectiveness of the machine learning algorithms while also making the analysis and modelling tasks more manageable. The criteria for dropping columns may change depending on the particular dataset and the issue being addressed, it is important to note. In light of the unique demands of the analysis, it is crucial to carefully assess the effects of dropping columns and to select the proper criteria for feature selection or reduction.

This indicates that the original 438 features had a high percentage of zeros and were therefore deemed redundant and less important, whereas the remaining 20 features were thought to be more significant and useful for the analysis. It is crucial to remember that choosing 0.75 as the threshold for dropping columns was somewhat arbitrary and could change depending on the specific dataset and analysis. A different threshold point might be preferable in certain situations. It is important to carefully consider how feature reduction might affect the analysis's precision and understandability. In some cases, removing excessive amounts of features could result in the loss of important data and harm the results.

Table 4: The most important permissions after feature reduction

Permission	Description
INTERNET	Allows applications to open network sockets
GET_TASKS	This constant was deprecated in API level 21. No longer enforced
CHANGE_WIFI_STATE	Allows applications to change Wi-Fi connectivity state
WRITE_EXTERNAL_STORAGE	Allows an application to write to external storage
READ_PHONE_STATE	Allows read-only access to phone state, including the phone number of the device, current cellular network information, the status of any ongoing calls, and a list of any Phone Accounts registered on the device
SYSTEM_ALERT_WINDOW	Allows an app to create windows using the type of TYPE_APPLICATION_OVERLAY, shown on top of all other apps
CALL_PHONE	Allows an application to initiate a phone call without going through the Dialer user interface for the user to confirm the call
READ_CONTACTS	Allows an application to read the user's contacts data
READ_SMS	Allows an application to read SMS messages
RECEIVE_SMS	Allows an application to receive SMS messages
WRITE_SMS	Allows an application to write SMS messages
SEND_SMS	Allows an application to send SMS messages
ACCESS_NETWORK_STATE	Allows applications to access information about networks
ACECESS_WIFI_STATE	Allows applications to access information about Wi-Fi networks
RECEIVE_BOOT_COMPLETED	Allows an application to receive the ACTION_BOOT_COMPLETED that is broadcast after thesystem finishes booting
VIBRATE	Allows access to the vibrator
WAKE_LOCK	Allows using PowerManager WakeLocks to keep processor from sleeping or screen from dimming
RECEIVE	Allows an app to receive certain types of messages or broadcasts sent by the system or other apps on the device
WRITE_SETTINGS	Allows an application to read or write the system settings

Neural network algorithms, such as ANN, CNN, and DNN, are artificial neural networks commonly used in machine learning and deep learning. These neural network algorithms have various architectural, functional, and design features. For instance, while DNNs are more general-purpose and can be used for a variety of tasks, CNNs are created specifically for image recognition tasks. The simplest type of neural network is an ANN, which can be used for straightforward tasks but lacks the flexibility and power of more intricate architectures like CNNs and DNNs. Overall, neural network algorithms have become increasingly popular in recent years due to their ability to learn complex patterns and relationships in data and their performance on a wide range of tasks. Fig. 3 illustrates the architectural framework of the proposed methodology used in this paper, spanning the entire process.

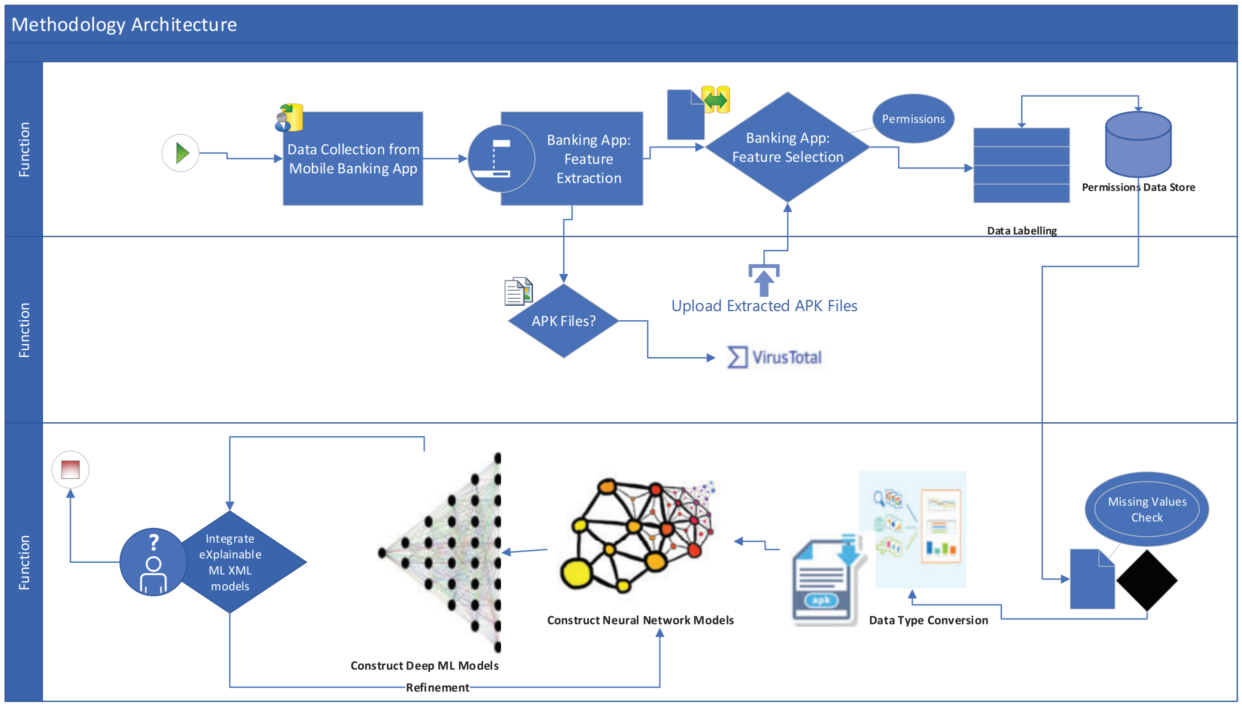

Methodology Architecture
Function
Function
Function
Data Collection from Mobile Banking App
Banking App: Feature Extraction
Banking App: Feature Selection
Permissions
Permissions Data Store
Data Labelling
APK Files?
Upload Extracted APK Files
VirusTotal
Missing Values Check
Data Type Conversion
Construct Neural Network Models
Construct Deep ML Models
Integrate eXplainable ML XML models
Refinement

Figure 3: Architectural framework for the proposed methodology

4 Performance Evaluation

We developed tuned neural network models with Python and the Scikit-learn, Keras, and TensorFlow libraries. We used the Python programming language to train and validate our neural network classifier on our banking malware dataset. For array operations and reading data from files, the NumPy and Pandas libraries are used. The simulation is divided into the following stages: Defining the network's parameters, such as node numbers and learning rate; reading the dataset; training the neural network; and finally validating the neural network with the remaining dataset. the process of developing tuned neural network models using various Python libraries such as Scikit-learn, Keras, and TensorFlow, specifically for the purpose of classifying banking malware.

The simulation of the neural network development process is divided into several stages. 1) Defining Network Parameters: In this stage, the parameters of the neural network are defined. This includes determining the number of nodes (or neurons) in each layer of the network and setting the learning rate, which controls how quickly the model adapts to the training data. 2) Reading the Dataset: The banking malware dataset is read into memory using the functionalities provided by the Python libraries. This step prepares the data for subsequent processing and training. 3) Training the Neural Network: The neural network model is trained using the prepared dataset. The model learns from the input data, adjusts its internal weights, and updates its predictions based on the provided training examples. This process continues until the model reaches a satisfactory level of performance. 4) Validating the Neural Network: Once the training is completed, the neural network's performance is evaluated using the remaining dataset that was not used during training. This validation step helps assess the generalization ability of the model, ensuring it can accurately classify new, unseen data.

Note that all the detailed simulation setup can be found in the shared source code in GitHub of the proposed method its link is given in the footnote under the abstract section.

We used 10-fold stratified cross-validation to evaluate neural network models' ability to generalise on the reduced-feature dataset. This involved randomly dividing the dataset into 10 subsets and subjecting it to five cycles of training and testing. In each cycle, one subset was excluded from the training process and used for testing. The performance metrics of the classifier were collected for each cycle, and if the variance between these metrics was high, it indicated that the classifier was over-fitted and did not generalise well. However, if the variance was low, the mean values of the performance metrics were considered reliable.

4.1 Experimental Metrics

We used the Python programming language and a number of libraries, including sci-kit-learn, Keras, and TensorFlow, in our experiments. We used a variety of metrics, including Accuracy, Precision, Recall, and F1, to assess the effectiveness of our method. Eqs. (1)–(4) define these metrics, respectively. To give you a better understanding of these metrics, let's dive into their definitions. True Positives, abbreviated as TP, refers to the number of positive samples that a binary machine learning classifier correctly identified as positive. It is calculated by dividing the total number of test instances with a true value of 1 by the number of test instances for which both the true and predicted values are equal to 1 (positive).

False Positives, on the other hand, are instances where a negative sample is incorrectly predicted by a binary machine learning classifier as positive. FP is calculated by dividing the total number of test instances with true values of 0 by the number of test instances with true values of 0 and 1, respectively. The number of negative samples that the binary classifier correctly identified as negative is known as True Negatives (TN). It is calculated by counting the test instances that had both a true value of 0 and a predicted value of 0, and then dividing that number by the total number of test instances that had a true value of 0. Positive samples that are mistakenly labelled as negative are known as False Negatives (FN). The test instances with a true value of 1 and a predicted value of 0 are counted, and the difference between these counts is divided by the total number of test instances with a true value of 1. This yields the value of FN.

Eq. (1)'s equation for accuracy gives a general idea of the model's performance. It is calculated by dividing the total product (TP + TN) by the total product (TP + TN + FP + FN).

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{1}$$

Precision, defined in Eq. (2), describes the proportion of predicted positive samples (Banking malware, in this case) that are actually positive. It is calculated by dividing TP by the sum of TP and FP.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

Recall, as expressed in Eq. (3), represents the percentage of correctly classified positive samples (banking malware) out of the total number of positive samples. It is calculated by dividing TP by the sum of TP and FN.

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

To consider both precision and recall, we also utilize the F1 score, which is a number between 0 and 1 and represents the harmonic mean of precision and recall. Eq. (4) demonstrates how the F1 score is calculated, where precision and recall are multiplied by 2 and divided by their sum.

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{4}$$

By utilising these evaluation metrics, we can comprehensively assess the performance of our approach and gain insights into the effectiveness of our binary ML-based classifiers.

4.2 Experimental Results

This section describes the experiments and compares the proposed method to other well-known classifiers as well as the most relevant previous research in this field. All experiments were carried out on a 64-bit Microsoft Windows 11 Professional operating system with hardware including an Intel(R) Core (TM) i5-8365U @ 1.60 GHz 1.90 GHz CPU, 16.00 GB RAM, and an Intel UHD Graphics 620 GPU.

4.2.1 Neural Network Models

Neural networks are a type of machine learning algorithm that is inspired by the structure and function of the human brain. They consist of interconnected nodes, or "neurons," that process and transmit information. When given a large amount of data, neural networks can learn to recognize patterns and relationships in the data and use that knowledge to make predictions or classifications. Using a deep-learning neural network is an effective strategy for detecting banking malware. Malware is challenging to detect using conventional techniques because it can manifest in a wide variety of ways and evolve quickly. The system can learn to recognise patterns and features indicative of banking malware using a neural network trained on a large dataset of examples of malware and non-malware behaviour, even if those patterns are complex and challenging to describe manually. Because of their adaptability, neural networks are well suited for difficult tasks like malware detection because they can handle a wide range of input data types and structures. It is important to keep in mind, though, that neural networks can also be prone to overfitting or underfitting the training set. As a result, careful tuning and model validation are necessary to guarantee the model's accuracy and generalisability. Experimental results with neural networks models is presented in Table 5.

Table 5: Experimental results with neural network models

	Accuracy % (Std)	Precision % (Std)	Recall % (Std)	F1 % (Std)
ANN	99.77 (+/−0.08%)	99.25 (+/−0.36%)	99.51 (+/−0.19%)	99.38 (+/−0.16%)
DNN	99.60 (+/−0.05%)	99.49 (+/−0.24%)	99.08 (+/−0.27%)	99.28 (+/−0.13%)
CNN	99.77 (+/−0.05%)	98.84 (+/−0.39%)	99.20 (+/−0.16%)	99.02 (+/−0.21%)

Artificial Neural Networks (ANN)

With an input layer, one or more hidden layers, and an output layer, ANNs are the most fundamental type of neural network. Numerous neurons are present in each layer and are linked by weights. Numerous tasks, such as classification, regression, and pattern recognition, can be performed using ANNs. One input layer, two dense layers, and one output layer make up the model's total of four layers.

1. Input layer: This layer is not explicitly defined in the code, but it is created implicitly when the first Dense layer with input_dim = 458 is added. It represents the data's input shape.
2. Dense layer (256 units): The model is used to add this layer.add(Dense(256, input_dim = 458, activation = 'relu'). It consists of 256 units and employs the ReLU activation function.
3. Dropout layer: This layer is added with the help of the model.add(Dropout(0.3)). It is added after the first dense layer to prevent overfitting during training by randomly setting a fraction of input units to 0.
4. Dense layer (128 units): The model is used to add this layer.add(Dense(128, activation = 'relu'). It has 128 units and is activated by the ReLU function.
5. Dropout layer: The model is used to add another dropout layer.add(Dropout(0.3)). It is added after the second dense layer to prevent overfitting even more.
6. Dense layer (1 unit): The model is used to add this layer. (Dense(1, activation = 'sigmoid')). It has a single unit and employs the sigmoid activation function. It provides the model's final output, which represents the probability of the input falling into the positive class.

Deep Neural Network (DNN)

DNNs are neural networks that typically have more than three layers. In complex processing-intensive tasks like speech recognition, natural language processing, and computer vision, DNNs are frequently used. The given model consists of a total of 7 layers, so, there are three hidden layers (two specified explicitly and one additional hidden layer) and one output layer, making a total of four dense layers. Additionally, there are three dropout layers added after each hidden layer and the extra dropout layer after the additional hidden layer.

Convolutional Neural Network (CNN)

CNNs are a type of neural network that is commonly used in image recognition and computer vision tasks. They use a series of convolutional layers to detect and extract features from the input data, before passing the features through one or more fully connected layers to produce the output.

4.2.2 Machine Learning (ML) Models

It is essential to emphasise the need for a technique that not only effectively detects banking malware but also provides explanations for its decisions. By combining a neural network-based approach with XML techniques, we tackle the challenge of transparency and interpretability in malware detection for Android apps. Our method analyses permission requests from an application, a commonly used indicator of application behaviour, to train a neural network for detecting malware. However, instead of solely relying on the neural network's output, we incorporate XML techniques to generate explanations for the classification decisions. This provides security analysts with insights into why an application is classified as either benign or malicious, enabling them to validate and understand the model's decisions.

By highlighting the importance of XML in the proposed method, we ensure that our approach not only achieves high accuracy in detecting banking malware but also enhances transparency and interpretability. This is crucial for building trust in the detection system and enabling effective collaboration between security analysts and the AI model. Overall, the need for an approach that combines effective malware detection with XML is emphasized in both the related works and proposed method sections. By addressing this need, our method contributes to improving the security of mobile banking applications and protecting users from financial harm while providing valuable insights into the decision-making process of the detection system.

To obtain the features for each application, the VirusTotal scanner was utilised. The VirusTotal scanner is a widely used online service that analyses files for potential malware by scanning them with multiple antivirus engines and providing additional insights, including the permissions requested by the application. The VirusTotal scanner was employed to analyse APK files, which are the installation files used by Android applications. By submitting the APK files to the VirusTotal scanner, the service performed a comprehensive analysis and extracted the permissions associated with each application. These permissions are valuable features for Android banking malware detection because they act as crucial indicators of an application's behaviour and security requirements. The classification model was built and evaluated by the study's code using the PyCaret library, Table 6 lists the configuration settings used in our experiment. By automating several steps, including data preprocessing, model selection, hyperparameter tuning, model interpretation, and deployment, the PyCaret library streamlines the machine-learning pipeline. The setup function was used to initialise the PyCaret environment after PyCaret and its necessary dependencies had been installed. This function included incorporating the permissions extracted by the VirusTotal scanner as part of the data preprocessing steps required to get the dataset ready for modelling. The permission-based features that the VirusTotal scanner extracted from the labelled instances of mobile applications served as the main set of input features in the dataset used for modelling. The target variable was specified as the 'label' column, which shows whether or not an application is categorised as banking malware. The compare_models function was used to compare the performance of different machine learning models in detecting Android banking malware. This function evaluated and ranked various models automatically based on a default evaluation metric, taking into account the dataset's specific characteristics, including the permissions extracted by the VirusTotal scanner. Following the model comparison, the selected model's hyperparameters were fine-tuned using the tune_model function. The goal of this step was to optimise the model's configuration so that it could effectively leverage the permission-based features extracted by the VirusTotal scanner to detect Android banking malware.

Table 6: PyCaret settings configuration

	Description	Value
1	Target	Label
2	Target type	Binary
3	Original data shape	(3300, 459)
4	Transformed data shape	(3300, 459)
5	Transformed train set shape	(2310, 459)
6	Transformed test set shape	(990, 459)
7	Numeric features	458
8	Rows with missing values	0.0%
9	Preprocess	True
10	Imputation type	Simple
11	Numeric imputation	Mean
12	Categorical imputation	Mode
13	Fold generator	StratifiedKFold

(Continued)

Table 6 (continued)

	Description	Value
14	Fold number	10
15	CPU jobs	−1
16	Use GPU	False

4.2.3 *Model Interpretability and Explanation*

In addition to evaluating the performance metrics of our model, we also leverage XML techniques to provide interpretations and explanations for the model's predictions. This enhances transparency and trust in the detection system. Fig. 4 shows the ROC curve for our tuned Random Forest model, illustrating its ability to distinguish between malware and benign apps with a high AUC score of 0.9971. The confusion matrix in Fig. 5 provides the breakdown of true positives, true negatives, false positives and false negatives. We can observe a high degree of accuracy, with only 6 false positives and 4 false negatives out of 990 test samples. The classification report in Fig. 6 presents precision, recall and F1 score for each class. We achieve strong results for both malware and benign app detection. Fig. 7 displays calibration plots assessing how well the predicted probabilities correlate with the actual fraction of positives. The close alignment to the diagonal in both plots indicates good calibration. The feature importance chart in Fig. 8 ranks the top 20 most influential features for classification. We see permissions related to access to sensitive resources like SMS, contacts, storage, etc., have high importance. Fig. 9 shows a SHAP summary plot highlighting features that pushed the model's output from the base value (the average model output over the dataset) to the actual output for a sample. Red indicates features that increased the probability of malware, while blue represents features that decreased the probability. The position on the x-axis indicates the impact of the feature. We can see that certain dangerous permissions like READ_SMS and SEND_SMS pushed the sample to be classified as malware. Table 5 tabulates the experimental results of the performance metrics for ANN, DNN, and CNN accordingly.

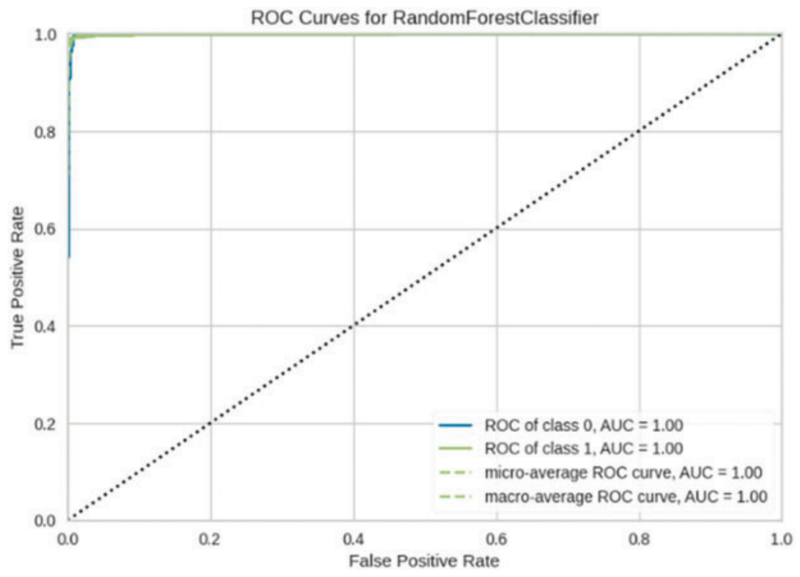


Figure 4: Tuned model ROC curve

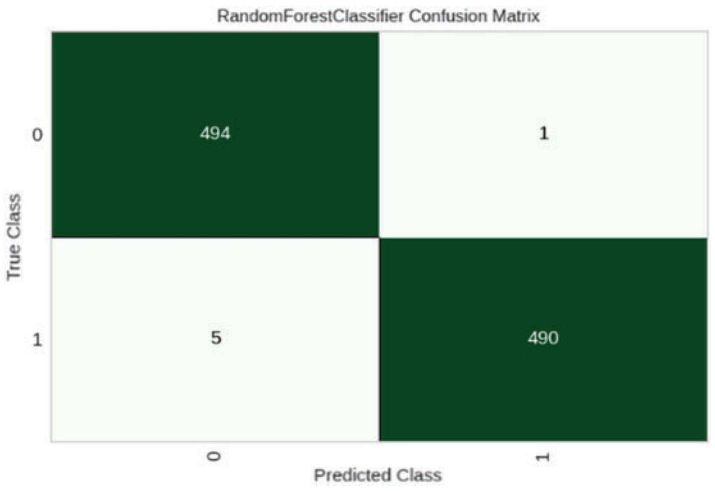


Figure 5: Tuned model confusion matrix

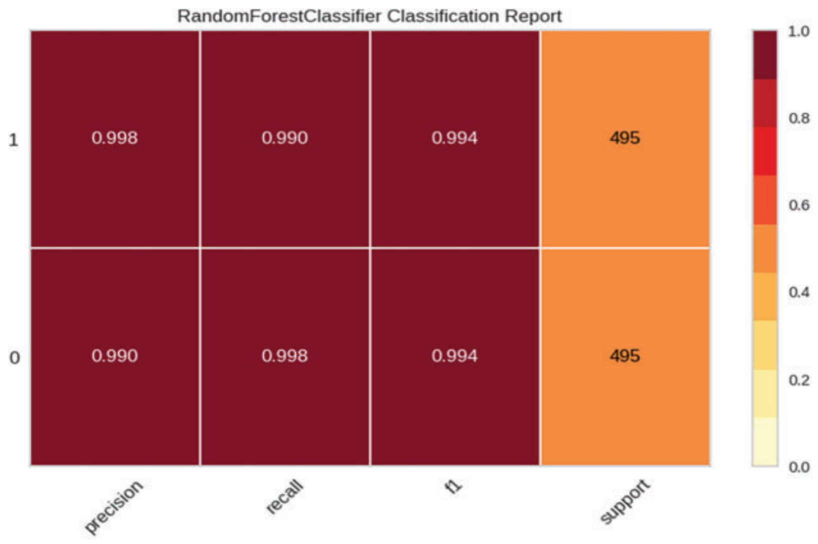


Figure 6: Tuned model classification report

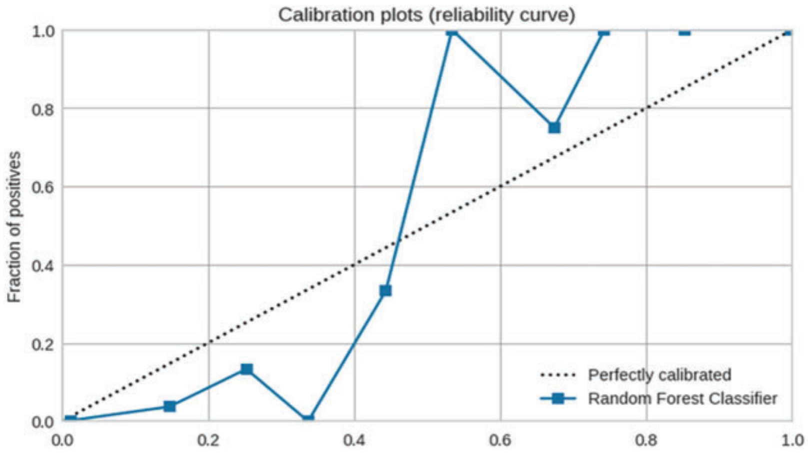


Figure 7: Tuned model calibration plots

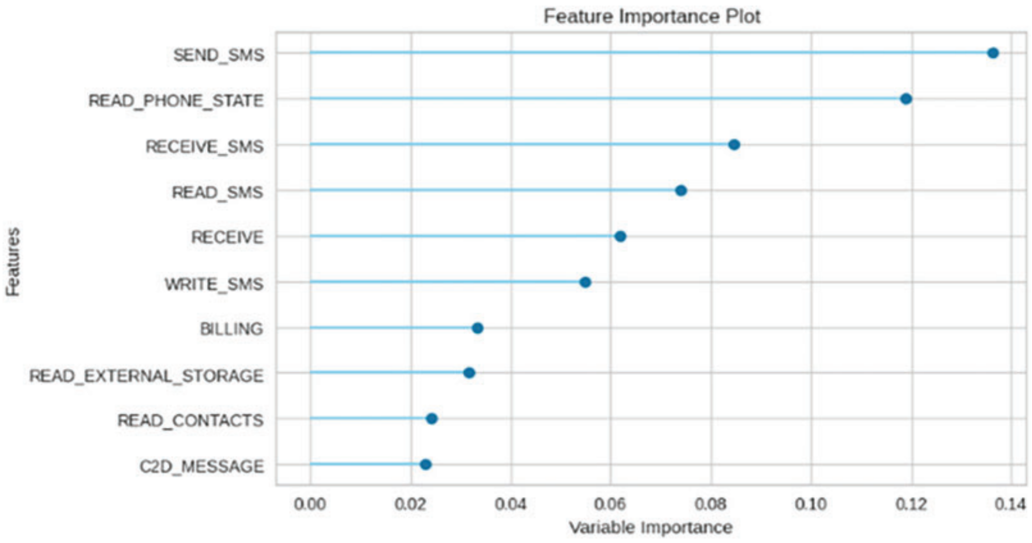


Figure 8: Feature importance plot

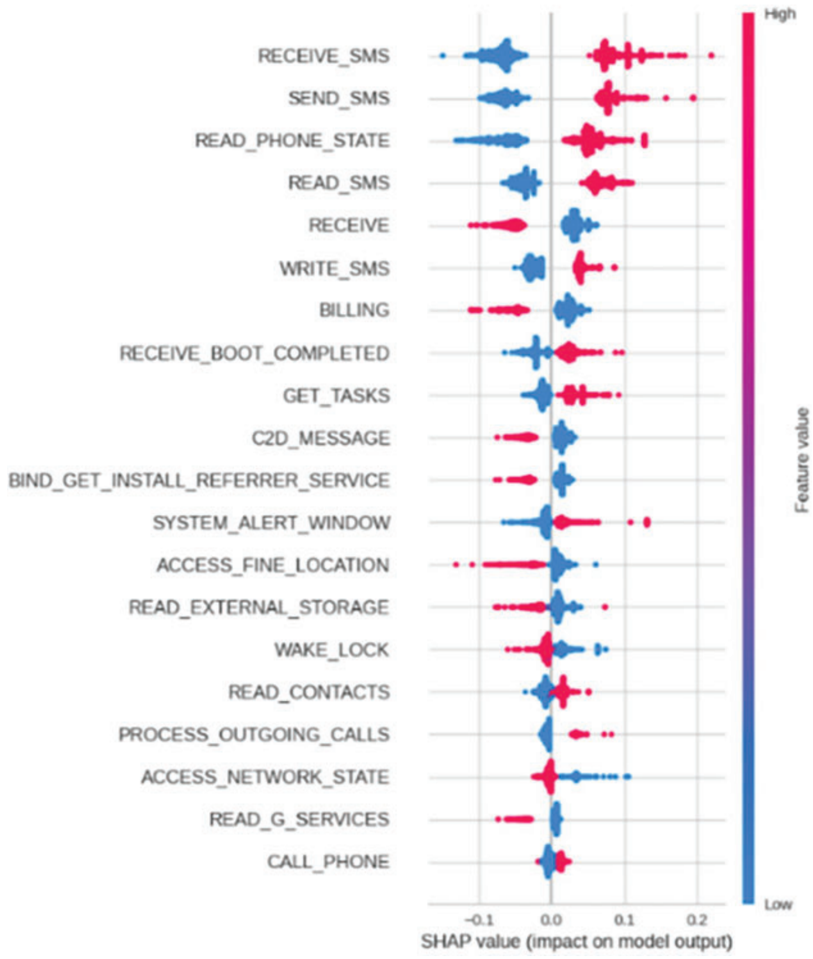


Figure 9: Shape value

In Table 7, the results of fitting 10 folds for each of the 10 candidates in terms of accuracy, recall, precision, and F1 score are presented. Table 8 reports the outcomes of various standard machine learning classifiers, including accuracy, AUC, recall, precision, and F1 score.

4.2.4 Comparison Results with Related Works

Table 9 shows the obtained results from the proposed neural networks and tuned ML models using the proposed dataset. On the other hand, Table 10 provides a comprehensive comparison of the classification performance of the proposed neural network models and the tuned ML model with results from recent related works. The comparison is based on key evaluation metrics commonly used in classification tasks: Accuracy, Precision, Recall, and F1 score. In comparison with ABT, the proposed method achieved high performance across all metrics, indicating a robust classification system. In comparison with APTAi and SMB, compared to ABT, this study demonstrates slightly lower accuracy and precision but still maintains reasonable performance in terms of recall and F1 score, though the proposed method in this paper has outperformed both methods.

Table 7: Fitting 10 folds for each 10 candidates

Fold	Accuracy	Recall	Precision	F1
0	0.9784	0.9565	1.0000	0.9778
1	0.9913	0.9826	1.0000	0.9912
2	0.9481	0.9304	0.9640	0.9469
3	0.9870	0.9739	1.0000	0.9868
4	0.9870	0.9826	0.9912	0.9869
5	0.9697	0.9397	1.0000	0.9689
6	0.9740	0.9483	1.0000	0.9735
7	0.9740	0.9569	0.9911	0.9737
8	0.9870	0.9741	1.0000	0.9869
9	0.9913	0.9828	1.0000	0.9913
Mean	0.9788	0.9628	0.9946	0.9784

Table 8: Standard ML classifier results

Model	Accuracy	AUC	Recall	Precision	F1
Random forest classifier	0.9887	0.9971	0.9879	0.9897	0.9888
Extra trees classifier	0.9879	0.9965	0.9853	0.9905	0.9879
CatBoost classifier	0.9861	0.9984	0.9827	0.9896	0.9861
SVM-linear kernel	0.9848	0.0000	0.9801	0.9897	0.9848
Extreme gradient boosting	0.9835	0.9979	0.9844	0.9828	0.9835
Gradient boosting classifier	0.9827	0.9976	0.9801	0.9853	0.9826
Logistic regression	0.9810	0.9981	0.9792	0.9828	0.9809
Ada boost classifier	0.9779	0.9973	0.9775	0.9786	0.9779
K neighbors classifier	0.9771	0.9932	0.9766	0.9776	0.9770
Decision tree classifier	0.9771	0.9785	0.9835	0.9713	0.9773
Ridge classifier	0.9727	0.0000	0.9662	0.9790	0.9725
Linear discriminant analysis	0.9701	0.9891	0.9671	0.9734	0.9700
Quadratic discriminant analysis	0.8827	0.8827	0.8086	0.9543	0.8726
Naive Bayes	0.8052	0.8052	0.9965	0.7213	0.8367
Dummy classifier	0.4978	0.5000	0.5000	0.2489	0.3324

Table 9: Tuned model results

Model	Accuracy	Recall	Precision	F1
Random forest classifier	0.9970	0.9958	0.9982	0.9970

Table 10: Comparison results with recent related works

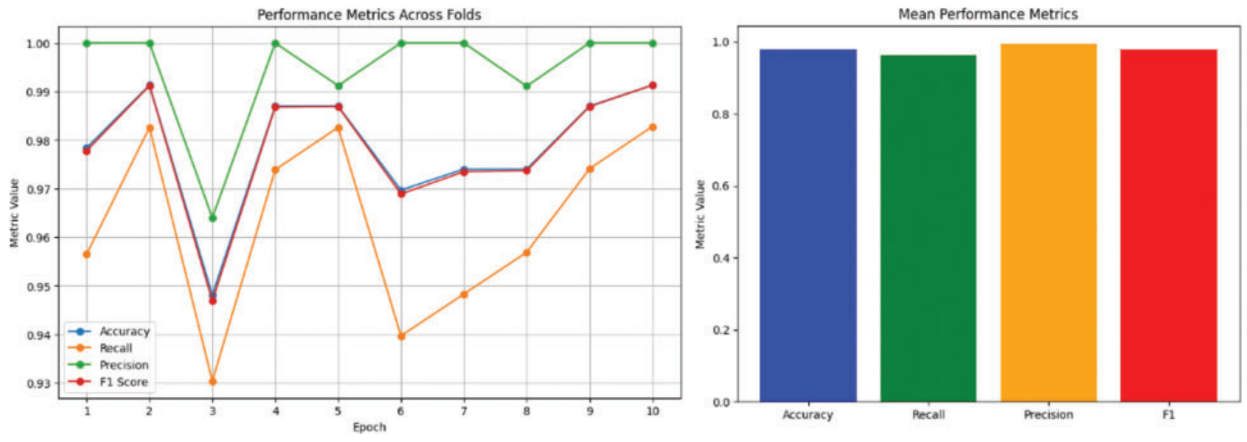
Method	Accuracy	Precision	Recall	F1
ABT	97.9	97.9	97.9	97.9
APTAi	91.8	90.9	92.7	91.8
SMB	95.0	94.0	96.0	95.0
Proposed ANN	**99.7**	**99.2**	**99.5**	**99.3**
Proposed DNN	**99.6**	**99.4**	**99.0**	**99.2**
Proposed CNN	**99.7**	**98.8**	**99.2**	**99.0**
Proposed tuned ML model	**99.7**	**99.5**	**99.8**	**99.7**

In contrast, the proposed ANN significantly outperforms the referenced studies, demonstrating superior accuracy, precision, recall, and F1 score. This indicates the effectiveness of the proposed neural network architecture in the classification task. The DNN model also shows exceptional performance, slightly lower than the ANN in terms of recall, but still significantly outperforming the referenced studies. On the other hand, the proposed CNN achieves outstanding accuracy and recall, demonstrating its efficacy in handling spatial information and patterns in the dataset. The tuned ML model also demonstrates excellent classification performance, with high values across all evaluation metrics. In summary, Table 10 indicates that the proposed neural network models (ANN, DNN, CNN) and the tuned ML model outperform the referenced studies in terms of accuracy, precision, recall, and F1 score, showcasing the effectiveness of the proposed approaches in the given classification task.

In order to compare the time complexity of the proposed work in comparison with the existing works, the best-performed Random Forest model out of the XML implemented model has been modelled using the Big O notation model. The time complexity for Training: O (iter $*$ (n $*$ d + d^2)), where *iter* is the number of iterations, *n* is the number of samples, and *d* is the number of features. The time complexity for Prediction: O (n $*$ d), where *n* is the number of samples and *d* is the number of features. In our proposed method, the number of features was efficiently maintained due to the reasoning mechanism that offers wise justification in selecting the best set.

Despite the promising results, a few potential limitations need to be further considered. For instance, the proposed method relies heavily on permissions as features to train the model for classification. Changes in permission structures or the emergence of new permissions could impact the model's performance. Regular updates on the developed dataset in this paper and adaptations may be necessary to address this limitation. Also, model overfitting risk could be one of the potential limitations that need to be carefully handled. Careful consideration of model generalisability and robustness is essential for real-world deployment. Besides, further training and testing of more than 10 epochs will be explored as a result of the update on the developed dataset will require further investigation.

Fig. 10a demonstrates the curve plot of the performance metrics, including accuracy, recall, precision, and F1 score, across 10 epochs. Each obtained value point represents the trend of a specific metric over the ten epochs, showcasing variations and patterns in the model's performance. On the other hand, Fig. 10b plots the bar shape of the obtained mean values for key performance metrics in Fig. 10a, including accuracy, recall, precision, and F1 score. The blue bar represents the mean value, emphasising the overall performance of the model across the evaluated metrics. This concise illustration provides a clear overview of the average model performance.



(**a**) Performance Metrics Across Epochs (**b**) Mean Performance Metrics

Figure 10: Performance measure comparison across 10 epochs with their mean values

5 Conclusions and Future Work

We have presented a novel method for detecting banking malware in the Android operating system by utilising permissions and employing optimized neural network models. Our approach, which is the first to use large number of permissions as features and apply neural network models for Android banking malware detection, has shown promising results. The increasing prevalence of mobile banking and the rise in malware attacks targeting financial applications emphasise the need for effective detection techniques. Traditional machine learning models have demonstrated their potential in identifying malware based on various features, but the lack of transparency and interpretability has been a concern. To address this limitation, we incorporated XML techniques into our detection method, providing human-understandable explanations for the decisions made by our model. By doing so, we enhance transparency and interpretability, crucial factors in building trust in the detection system. Our experiments, conducted using a dataset consisting of benign and malicious banking APK files, demonstrated that our proposed neural network models outperformed several well-known and standard machine learning models. The results indicate that the permissions provided by Android applications can be effectively utilized to detect banking malware.

As part of future work, we plan to expand our feature set by combining other types of features with permissions and integrating more diverse feature sets. This will enable us to detect sophisticated banking malware, considering additional aspects of application behaviour and characteristics. Besides, as banking malware is heavily dynamic, constantly evolving to bypass detection mechanisms, adaptation to such highly evolved malware tactics is vital. Furthermore, more experimental results will be part of the plan for future work, such as the use of multiple validation and explanations for a trustworthy ML system [28].

Acknowledgement: This work was supported by the Deanship of Scientific Research (DSR), King Khalid University, Abha, under Grant No. RGP.1/260/45. The author, therefore, gratefully acknowledges the DSR's technical and financial support.

Funding Statement: The authors would like to acknowledge the support given by the project title "PRAVE: Proactive Authentication and Verification Embedded Model for Critical Cyber-Physical Systems". PRAVE was supported by the Department of Computer Science, Nottingham Trent University, QR Fund, and ECR Fund.

Author Contributions: S.S.: Writing—Original Draft Preparation, Conceptualisation, Dataset Development, Software. A.S.S.: Supervision, Methodology of XML, Revising, and Editing. O.K. and A.K.: Supervision and Editing. M. Aljaidi, M. Ali, and M. Abazeed: Conceptualisation, Revising and Editing. All authors have read and agreed to the published version of the manuscript.

Availability of Data and Materials: The source code and dataset can be found on GitHub: https://github.com/alisafaa12/MoBShield.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] A. Z. Zaidi, C. Y. Chong, Z. Jin, R. Parthiban, and A. S. Sadiq, "Touch-based continuous mobile device authentication: State-of-the-art, challenges and opportunities," *J. Netw Comput. Appl.*, vol. 191, no. 11, pp. 103162, 2021. doi: 10.1016/j.jnca.2021.103162.

[2] S. Seraj, E. Pimenidis, M. Pavlidis, S. Kapetanakis, M. Trovati and N. Polatidis, "BotDroid: Permission-based android Botnet detection using neural networks," in *Int. Conf. Eng. Appl. Neural Netw.*, Cham, Springer Nature Switzerland, Jun. 2023, pp. 71–84.

[3] D. Hayes, F. Cappa, and N. A. Le-Khac, "An effective approach to mobile device management: Security and privacy issues associated with mobile applications," *Digit. Bus.*, vol. 1, no. 1, pp. 100001, 2020. doi: 10.1016/j.digbus.2020.100001.

[4] C. E. Rubio-Medrano, P. K. D. Soundrapandian, M. Hill, L. Claramunt, J. Baek and G. J. Ahn, "DyPolDroid: Protecting against permission-abuse attacks in android," *Inform. Syst. Front.*, vol. 25, no. 2, pp. 529–548, 2023. doi: 10.1007/s10796-022-10328-8.

[5] F. Akbar *et al.*, "Permissions-based detection of android malware using machine learning," *Symmetry*, vol. 14, no. 4, pp. 718, 2022. doi: 10.3390/sym14040718.

[6] O. Sanda, M. Pavlidis, S. Seraj, and N. Polatidis, "Long-range attack detection on permission-less blockchains using deep learning," *Expert. Syst. Appl.*, vol. 218, no. 1, pp. 119606, 2023. doi: 10.1016/j.eswa.2023.119606.

[7] V. Belle and I. Papantonis, "Principles and practice of explainable machine learning," *Front. Big Data*, vol. 39, pp. 27, 2021. doi: 10.3389/fdata.2021.688969.

[8] F. Ullah, A. Alsirhani, M. M. Alshahrani, A. Alomari, H. Naeem and S. A. Shah, "Explainable malware detection system using transformers-based transfer learning and multi-model visual representation," *Sens.*, vol. 22, no. 18, pp. 6766, 2022. doi: 10.3390/s22186766.

[9] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should I trust you?" Explaining the predictions of any classifier," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, Aug. 2016, pp. 1135–1144.

[10] A. Barredo Arrieta *et al.*, "Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI," *Inform. Fusion*, vol. 58, no. 3, pp. 82–115, 2020. doi: 10.1016/j.inffus.2019.12.012.

[11] A. Adadi and M. Berrada, "Peeking inside the black-box: A survey on explainable artificial intelligence (XAI)," *IEEE Access*, vol. 6, pp. 52138–52160, 2018. doi: 10.1109/ACCESS.2018.2870052.

[12] T. Miller, "Explanation in artificial intelligence: Insights from the social sciences," *Artif. Intell.*, vol. 267, no. 2, pp. 1–38, 2019. doi: 10.1016/j.artint.2018.07.007.

[13] S. Adhikari, S. Nepal, and R. Bista, “A framework for the detection of banking trojans in android,” *Int. J. Netw. Secur. Appl.*, vol. 14, no. 6, pp. 27–39, 2022. doi: 10.5121/ijnsa.2022.14603.

[14] M. A. Kazi, S. Woodhead, and D. Gan, “Comparing the performance of supervised machine learning algorithms when used with a manual feature selection process to detect Zeus malware,” *Int. J. Grid Util. Comput.*, vol. 13, no. 5, pp. 495–504, 2022. doi: 10.1504/IJGUC.2022.126167.

[15] P. Black, I. Gondal, and R. Layton, “A survey of similarities in banking malware behaviours,” *Comput Secur*, vol. 77, no. 7, pp. 756–772, 2018. doi: 10.1016/j.cose.2017.09.013.

[16] N. Etaher and G. Weir, “Understanding the threat of banking malware,” in *Cyberforensics 2014-Int. Conf. Cybercrime, Sec. Digi. Forensics*, Jun. 2014.

[17] E. D. Ansong and T. Q. Synaepa-Addision, “A comparative study of user data security and privacy in native and cross platform Android mobile banking applications,” in *2019 Int. Conf. Cyber Sec. Internet Things (ICSIoT)*, IEEE, May 2019, pp. 5–10.

[18] S. Chen *et al.*, “An empirical assessment of security risks of global android banking apps,” in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, Jun. 2020, pp. 1310–1322.

[19] J. H. Jung, J. Y. Kim, H. C. Lee, and J. H. Yi, “Repackaging attack on android banking applications and its countermeasures,” *Wireless Pers. Commun.*, vol. 73, no. 4, pp. 1421–1437, 2013. doi: 10.1007/s11277-013-1258-x.

[20] G. Koala, D. Bassolé, A. Zerbo/Sabané, T. F. Bissyandé, and O. Sié, “Analysis of the impact of permissions on the vulnerability of mobile applications,” in *e-Infrastructure and e-Services for Developing Countries*. Springer, Cham, 2019, pp. 3–14.

[21] Y. Kouraogo, K. Zkik, N. E. J. E. Idrissi, and G. Orhanou, “Security model on mobile banking application: Attack simulation and countermeasures,” *Int. J. Intell. Enterp.*, vol. 4, no. 1–2, pp. 155–167, 2017. doi: 10.1504/IJIE.2017.087014.

[22] A. Razaghpanah *et al.*, “Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem,” in *25th Annu. Netw. Distrib. Sys. Sec. Symp. (NDSS 2018)*, 2018.

[23] N. Kumar, V. Kumar, and M. Gaur, “Banking trojans APK detection using formal methods,” in *2019 4th Int. Conf. Inform. Sys. Comp. Netw. (ISCON)*, IEEE, Nov. 2019, pp. 606–609.

[24] G. Iadarola, F. Martinelli, F. Mercaldo, and A. Santone, “Formal methods for android banking malware analysis and detection,” in *2019 6th Int. Conf. Internet Things: Sys., Manag. Sec. (IOTSMS)*, IEEE, Oct. 2019, pp. 331–336.

[25] S. Bojjagani and V. N. Sastry, “VAPTAi: A threat model for vulnerability assessment and penetration testing of android and iOS mobile banking apps,” in *2017 IEEE 3rd Int. Conf. Collab. Internet Comp. (CIC)*, IEEE, 2017, October, pp. 77–86.

[26] M. Abusharekh, M. Hamad, and M. Al-Akhras, “Analyzing the security of banking android applications,” *J. Inform. Secur.*, vol. 13, no. 1, pp. 1–15, 2022.

[27] M. R. Rahaman, M. S. Islam, and M. M. Hossain, “Investigating the security of mobile banking applications,” in *Proc. 7th Int. Conf. Comp. Commun. Sys.*, 2022, pp. 537–543.

[28] M. M. Hasan, C. N. Watling, and G. S. Larue, “Validation and interpretation of a multimodal drowsiness detection system using explainable machine learning,” *Comput. Meth. Prog. Bio.*, vol. 243, no. 21, pp. 107925, 2024. doi: 10.1016/j.cmpb.2023.107925.