

Network Traffic-Based Malware Detection for Android Devices

Dincy R. Arikkat¹, Hasna Jabeen N. K.¹, Mirko Lai², Serena Nicolazzo^{2,*}, Antonino Nocera³, Anafa P.¹, Vinod P.¹ and Rafidha Rehiman K. A.¹

¹Department of Computer Applications, Cochin University of Science and Technology, India

²Department of Science and Technological Innovation, University of Eastern Piedmont, Italy

³Department of Electrical, Computer and Biomedical Engineering, University of Pavia, Italy

Abstract

The widespread adoption of Android smartphones has made them increasingly attractive and vulnerable to malicious cyber threats. Such malware can jeopardize user privacy, steal personal data, and inflict considerable damage. This creates an urgent need for effective malware detection methods for Android systems. Traditional signature-based approaches fall short as malware developers continuously devise new evasion techniques. Consequently, developing sophisticated and cutting-edge malware detection strategies is essential to safeguard Android users' security and protect their personal information. To tackle this issue, in this paper, we propose a novel approach leveraging several Machine Learning (ML) and Deep Learning (DL) techniques to analyze network traffic to identify and detect malware on Android-based systems. We applied ML and DL techniques to analyze a comprehensive dataset consisting of malicious and benign APKs for network traffic. Specifically, we implemented the model using Decision Tree, KNearestNeighbour, Random Forest, XGBoost, Logistic Regression, Support Vector Machine, and Deep Neural Network. XGB Classifier outperforms other models with an F1-score (weighted) of 0.9323. Furthermore, we interpreted the model's decisions using SHapley Additive exPlanations, Local Interpretable Model-agnostic Explanations, and counterfactual analysis. Our method enhances the precision and clarity of mobile malware detection, deepens our understanding of the characteristics of malicious Android applications, and advances the overarching goal of improving mobile security.

Keywords

Android, Malware Detection, Network, Machine Learning, Deep Learning.

1. Introduction

The Android operating system¹, created by Google and released in 2008, is built on Linux and features an official Google Play app store², which hosts over 2.5 million apps. Additionally, there are numerous third-party app stores available for Android users. As smartphones increasingly facilitate communication activities such as voice calls, SMS, email, and social networking, concerns about privacy invasion and tracking have intensified. The open architecture of the Android system makes it especially vulnerable to malicious apps, posing significant threats to user privacy and security [1].

As malware prevalence rises, so do efforts to mitigate associated risks. Malware detection predominantly relies on two fundamental approaches: static analysis and dynamic analysis techniques. Static analysis involves examining malware without executing it, while dynamic analysis involves observing malware behavior in an isolated environment [2]. However, static analysis can struggle to detect malware that uses code polymorphism or obfuscation techniques, allowing multiple variants to evade detection. Although dynamic analysis shows promise in detecting and analyzing malware, it

ITADATA2025: The 4th Italian Conference on Big Data and Data Science, September 9–11, 2025, Turin, Italy

*Corresponding author.

✉ dincyrarikkat@cusat.ac.in (D. R. Arikkat); hasna.pg.cusat.ac.in (H. J. N. K.); mirko.lai@uniupo.it (M. Lai); serena.nicolazzo@uniupo.it (S. Nicolazzo); antonino.nocera@unipv.it (A. Nocera); anafa@pg.cusat.ac.in (A. P.); vinod.p@cusat.ac.in (V. P.); rafidharehimanka@cusat.ac.in (R. R. K. A.)

ORCID 0000-0003-1042-0861 (M. Lai); 0000-0003-2719-9526 (S. Nicolazzo); 0000-0003-2120-2341 (A. Nocera); 0000-0001-6078-2014 (V. P.); 0009-0004-7330-1178 (R. R. K. A.)



© 2025 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

¹<https://www.android.com/>

²<https://play.google.com/store>

requires numerous executions to fully understand an app's behavior, making it challenging to use on resource-constrained smart devices [3].

Researchers continually seek innovative approaches to improve mobile malware detection methodologies. This work analyzes network traffic to identify malware applications, as malware often performs malicious activities through network connections. In this paper, mobile network traffic is captured and analyzed using TCPdump³. Relevant static features are then extracted with CICFlowMeter⁴, a network traffic flow generator and analyzer. Six distinct ML models were evaluated, and the XGBoost classifier was ultimately selected for its superior performance.

Additionally, guaranteeing the interpretability of ML models is essential for improving the detection methods of mobile malware and building trust. For this reason, we explained the predictions using SHapley Additive exPlanations (SHAP) values, Local Interpretable Model-agnostic Explanations (LIME), and Counter Factuals. This approach contributes to the understanding of mobile malware detection, provides greater transparency in the detection process, and builds confidence in the detection system. Furthermore, we investigate the potential of SHAP for identifying important features and acting as a feature selection mechanism.

In summary, the primary objectives of this paper are to design and implement an effective Android malware detection system by harnessing network traffic data from a wide range of Android applications. Specifically, the main contributions of this research are:

- We develop a feature extraction pipeline that analyses network packets to extract relevant features for subsequent ML and DL-based classification models.
- We investigate state-of-the-art algorithms to automate the detection process and improve malware detection accuracy.
- We employ explainability tools to ensure our model's transparency and interoperability.
- We investigate the potential of SHAP as a feature selection technique, examining its ability to identify the most influential features and reduce the feature space.

This paper is structured as follows: Section 2 provides an overview of the relevant literature and research related to our approach. Section 3 delves into the technical details of our framework, including its key components and the steps taken for malware identification and detection. Section 4 presents the experimental setup and results, including the performance evaluation of our solution and an interpretation of our model. Finally, Section 5 concludes the paper and discusses potential future research directions.

2. Related Works

The problem of privacy and security related to mobile apps has been a topic of research in the last decade. Numerous studies have examined the behavior of mobile apps and their effects on users' privacy and security [4, 5, 6, 7]. One area of interest in this field is network traffic analysis, where researchers analyze the network traffic generated by mobile apps to identify malicious activities or privacy violations.

The paper proposed by Onur Zungur et al. [8] develops a dynamic analysis system for Android apps called LIBSPECTOR that attributes network traffic to the method and library responsible for sending or receiving the packets. The authors used LIBSPECTOR to analyze 25,000 Android apps and found that advertisement libraries comprise a significant quarter of mobile app network traffic. Notably, 35% of the analyzed apps were found to have only advertisement and tracker traffic. Shanshan Wang et al. [9] present an automated approach for malware detection using the text semantics of HTTP flows. By analyzing the text-level properties of network traffic, their method achieves a remarkable accuracy rate of 99.15% in detecting malware, outperforming existing techniques. Moreover, it surpasses standard anti-virus scanners, which have a detection rate of 54.81% and can efficiently identify newly discovered

³<https://www.tcpdump.org/>

⁴<https://pypi.org/project/cicflowmeter/>

malware with minimal samples. This detection technique is also effective for analyzing encrypted traffic and can be applied to various network types, including business, residential, and mobile networks.

Malik et al. [10] introduced CREDROID, which detects malicious Android apps by analyzing their DNS requests and network activity through pattern-based detection. This system considers various factors to assess the trustworthiness of an app, including the remote server, the data being transmitted, and the communication protocol. The study found that 63% of malware apps in a common dataset produced network traffic. Abderrahmane et al. [11] proposed an approach for detecting malicious applications by analyzing the system calls generated by Android apps during their operation, through several ML techniques. Specifically, a convolutional neural network is used on a dataset containing both malicious and benign apps, achieving an accuracy rate of 93.29% in just a few seconds of analysis.

Abuthawabeh et al. [12] proposed an enhanced malware detection, categorization, and family classification model in the Android environment using conversation-level network traffic features extracted from the CICAndMal2017 dataset that enhances malware categorization and family classification by up to 31.14% and 41.09% for recall, respectively. The results demonstrate the effectiveness of conversation-based features and provide valuable insights for improving malware detection in Android environments. Also, the proposal of [13] utilizes the CICAndMal2017 dataset to test a network traffic-based approach for detecting Android malware using deep learning algorithms.

The authors of [14] describe a hybrid Android malware detection approach named CoDroid using CNN-BiLSTM-Attention. This approach builds a sequence-based model using sequences of static opcodes and dynamic system calls. The work of Vinod P et al. [15] designs a system call-based approach to detect mobile malware on the Android platform, leveraging ML techniques to extract system calls and then performing empirical estimation of system calls derived from diverse datasets. The authors conducted experiments on artificially generated system calls, employing two distinct feature selection methods, and subsequently validated their findings on five diverse datasets. The results show an accuracy exceeding 99.9% and an Area Under the Curve of 1.0. The authors of [16] present a new feature selection mechanism called the Selection of Relevant Attributes for Improving Locally Extracted Features using Classical Feature Selectors (SAILS). SAILS is constructed upon established feature selection techniques, including mutual information, distinguishing feature selectors, and Galavotti–Sebastiani–Simi, which serve as local feature selectors. Furthermore, a pioneering global feature selection approach, weighted feature selection, is introduced to complement the existing methods.

While many researchers have focused on Android malware detection, few have explored network traffic for this purpose. Previous studies primarily concentrated on static features and system calls. In contrast, our work emphasizes network traffic for Android malware detection, as a significant number of attacks occur through the network. Additionally, while many works leverage XAI tools to explain model decision-making, the use of SHAP as a feature selection mechanism remains underexplored. XAI tools highlight important features that drive specific decisions, but the potential of SHAP for feature selection has yet to be fully investigated. In our work, we explore whether SHAP can be used as an effective feature selection technique.

3. Proposed Approach

The proposed methodology is depicted in Figure 1. As the first step of *Data Collection*, we used the Androzoo⁵ dataset, a well-established resource for Android malware detection and classification. This dataset comprises a diverse range of Android applications collected from various sources. To confirm whether these applications were benign or malicious, we evaluated them using VirusTotal⁶. For our study, we selected a total of 8,819 applications from this dataset.

Secondly, we deployed APK files for installation and execution within an Android emulator, a virtual environment that replicates the functionality of an Android device on a computer. Specifically, we used an API level 24 emulator on a host machine with internet connectivity to optimize the execution

⁵<https://androzo.uni.lu/>

⁶<https://www.virustotal.com/>

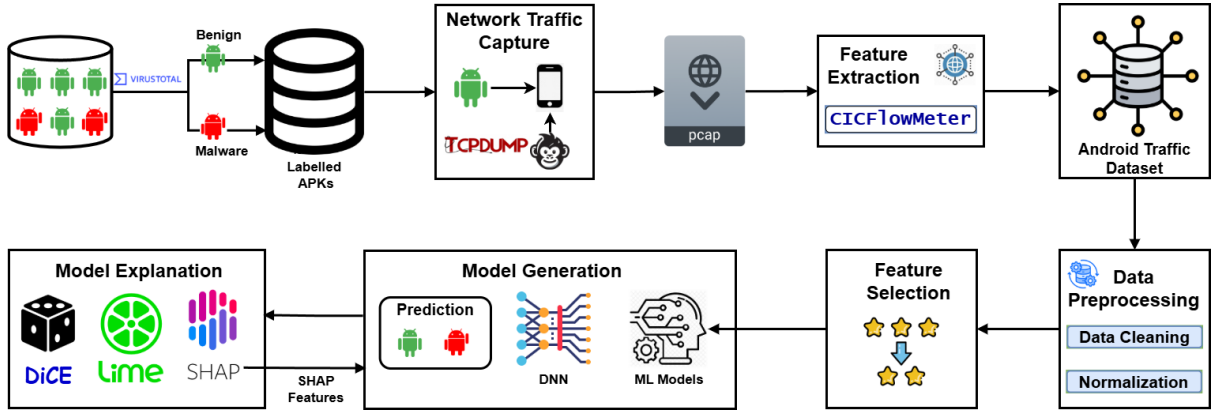


Figure 1: The architecture of our framework

of Android applications. This high-performance emulator, developed by a third party, leverages x86 architecture virtualization technology for enhanced speed and efficiency. Further, to capture network traffic, we employed *Tcpdump*, a robust command-line packet analyzer tool designed for Linux/Unix systems. *Tcpdump* facilitates the real-time or file-based examination of network traffic, enabling a detailed understanding of the communication patterns and behaviors within the application. It enabled us to identify pertinent features for developing ML models for detecting malware in Android applications. During this phase, the Android Monkey tool⁷ is utilized to simulate random user interactions, effectively mimicking real user behavior within an Android application running on an emulator. For each application, we performed 1,000 injection events, including actions such as tapping buttons and swiping screens. This comprehensive simulation facilitates a thorough evaluation of the application’s behavior under realistic conditions. Additionally, it aids in uncovering potential malicious activities, thereby significantly enhancing the detection rate of Android malware. After completing the simulation, the network traffic generated during testing is captured and stored in PCAP files for detailed analysis. To maintain the integrity and reliability of the testing environment, the application is uninstalled, and the emulator is restored to its original state after each test cycle. This ensures a clean and controlled setup for subsequent tests, minimizing the risk of cross-contamination and guaranteeing consistent, reliable results. At the end of the *Data Collection* step, we generated a Network Traffic APK Dataset comprising 2,980 malware samples and 5,866 benign samples.

The second step of our framework is *Feature Generation*, where we extract meaningful insights from PCAP files by transforming or combining data in specific ways. After capturing network traffic data from both malware and benign applications, we utilized *CICFlowMeter*⁸, a portable and scalable open-source tool, to extract relevant features. This versatile tool supports various input file formats, including PCAP, PCAP Next Generation, and raw binary files, and produces output in CSV format. Using *CICFlowMeter*, we extracted a total of 76 relevant features, encompassing metrics such as flow bytes, total packets in the forward and backward directions, and the mean inter-arrival time of flows, among others. These features provide valuable input for subsequent analysis and model training.

The third step of our framework consists of the *Data Preprocessing*, in which we identified and addressed errors, missing values, and inconsistent data, which must be corrected or removed before proceeding with the analysis. For this aim, we used *min-max* normalization, which involves scaling feature values to a range between 0 and 1. This technique standardizes the features and eliminates biases resulting from different measurement scales, providing consistent and comparable data for analysis. Furthermore, we removed certain columns from the dataset deemed irrelevant to the analysis, such as source and destination IP addresses, ports, flow ID, Protocol, and Timestamp. This practice minimizes noise and enhances data quality.

The fourth step is *Feature Selection*, where we pinpoint the most influential features that significantly

⁷<https://monkeytools.ca/>

⁸<https://github.com/ahlashkari/CICFlowMeter>

impact a model's predictive accuracy. For the case of the Network Traffic APK dataset, we exploited the *SelectKBest* method for feature selection, a univariate feature selection approach. This method selects the top k features based on their scores, which are computed through the ANOVA F-value, and uses `f_classif` as the scoring function. The k best features were calculated for various ML models, including KNN, logistic regression, SVM, decision tree, random forest, and XGBoost.

After the feature selection, we performed the *Parameter Tuning* in which we applied *Randomized-SearchCV* to tune these hyperparameters for different models: Random Forest, XGBoost, SVM, Logistic Regression, KNN, and Decision Tree. Subsequently, the *Model Training* was carried out. During this step, we applied several ML and DL techniques to classify and predict malware, including Decision Trees, Random Forest, Support Vector Machines, Logistic Regression, K-Nearest neighbor, XGBoost, and DNN.

After building the malware detection model, we applied XAI (eXplainable Artificial Intelligence, hereafter) methods to provide interpretability of our findings. In particular, we exploited SHAP, LIME, and CounterFactuals. SHAP [17] is a game-theoretic approach proposed by Lundberg and Lee to explain any ML model using Shapley values. Shapley values are a cooperative game theory approach that represents the average marginal contribution of an instance of a feature among all possible coalitions. LIME [18] stands for Local Interpretable Model Agnostic Explanation. As a model-agnostic method, it can explain any black-box models, and as a local interpretation method, it can explain a particular instance. Counterfactuals are hypothetical scenarios generated by altering specific feature values of a given instance to understand how those changes affect the model's predictions.

Finally, we conducted an additional round of feature selection using SHAP, leveraging its ability to provide global explanations and identify the most important features contributing to predictions. Based on this insight, we investigated whether SHAP could serve as an effective feature selection technique to enhance model performance.

4. Experiment Results

In this section, we present the experiments conducted to evaluate the performance of our framework, including the environment setup and the metrics used. Moreover, we explain the output of models through two XAI tools.

4.1. Environment Setup

The experiments were carried out on a Ubuntu 22.04-based computing environment featuring an Intel Core i7 processor and 16 GB of RAM. Data collection was facilitated through the Android Software Development Kit (SDK), while code execution was performed on Google Colab.

4.2. Evaluation Metrics

In our research study, we assessed performance using the below key evaluation metrics:

- *Precision* (\mathcal{P}) measures the accuracy of malicious application detection by calculating the ratio of correctly identified malicious applications, TP (True Positive), to the total number of applications classified as malicious ($TP + \text{False Positives (FP)}$). This metric, as shown in Equation 1, is represented as a percentage.

$$\mathcal{P} = \frac{TP}{TP + FP} \quad (1)$$

- *Recall* (\mathcal{R}) quantifies the detection rate of malicious applications by calculating the proportion of correctly identified malicious applications (TP) relative to the total number of actual malicious applications in the dataset ($TP + FN$). This metric, depicted in Equation 2, is expressed as a percentage.

$$\mathcal{R} = \frac{TP}{TP + FN} \quad (2)$$

- The *F1-score* is a balanced performance metric that combines precision and recall into a single value, ranging from 0 to 1. It is calculated as the harmonic mean of precision and recall depicted in Equation 3.

$$F1 - score = 2 * \frac{\mathcal{P} * \mathcal{R}}{\mathcal{P} + \mathcal{R}} \quad (3)$$

4.3. Results and Discussion

As discussed in the Introduction, our objective is to develop an accurate ML or DL model and then provide an explainer to help interpret the model. To achieve this, we experimented with several models, including Decision Trees, Random Forest, Support Vector Machines, Logistic Regression, K-Nearest neighbor, XGBoost, and DNN, for binary class prediction on the datasets. For the feature selection process, we used the SelectKBest method to identify the top k features based on their scores. The best k features were computed for various machine learning models, including KNN, Logistic Regression, SVM, Decision Tree, Random Forest, and XGBoost. The selected features for each model are presented in Table 1.

Model	Features
Logistic Regression	39
KNN	37
SVM	49
Decision Tree	57
Random Forest	53
XGBoost	66

Table 1
Feature Selection Results for ML Models

In order to account for the significant class imbalance present in the datasets, we utilized the weighted F1-score as a performance metric. Our analysis focused on the highest-performing models, assessing their precision, recall, and F1 score. To optimize the performance of these models, we tuned their hyperparameters using various methods, including TandomizedSearchCV and GridSearchCV. The hyperparameter values obtained as a result of tuning are depicted in Table 2.

Model	Hyper Parameter
KNN	metric=manhattan, n_neighbors=22
Logistic Regression	C=100, class_weight=balanced, max_iter=1,000, penalty=l1, solver=liblinear
Decision Tree	max_depth=20, min_samples_leaf=2, min_samples_split=5
SVM	C=40, class_weight='balanced', gamma=1
Random Forest	class_weight='balanced', criterion='entropy', max_depth=90, max_features='auto', min_samples_split=5, n_estimators=2,400
XGBoost	enable_categorical=False, gamma=0.1, learning_rate=0.25, max_features='sqrt', min_samples_leaf=4, min_samples_split=5, missing=nan, n_estimators=2,000

Table 2
Hyperparameter Configuration for ML Models

Following the optimization of hyperparameters, we developed and compared diverse ML and DL models for the detection of Android malware utilizing network traffic data. The results, as presented in Table 3, demonstrate that the XGBoost model exhibited exceptional performance, boasting a precision of 93.23%, recall of 93.25%, and F1-score of 93.23%. The Random Forest model displayed a notable performance, achieving a precision of 90.87%, recall of 90.93%, and F1-score of 90.85%.

Model	Precision	Recall	F1-Score
Logistic Regression	0.7870	0.7732	0.7772
KNN	0.8211	0.8237	0.8184
Decision Tree	0.8527	0.8515	0.8520
SVM	0.8715	0.8656	0.8673
DNN	0.8944	0.8951	0.8938
Random Forest	0.9087	0.9093	0.9085
XGBoost	0.9323	0.9325	0.9323

Table 3

Performance Comparison of Android Malware Detection Models using network traffic

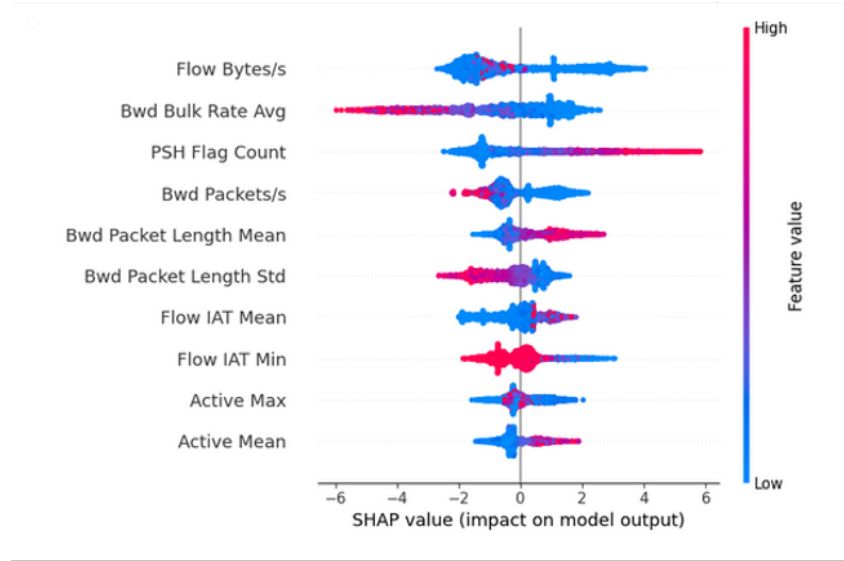


Figure 2: SHAP Summary Plot of XGBoost Network Traffic Model

4.4. XAI results

In this section, we interpret the model’s decision-making process using SHAP, LIME, and counterfactuals. We leveraged SHAP to perform global explanations. For the global explanation, we aggregated SHAP values across multiple predictions to gain insights into the overall importance of each feature for the model.

As for the *global explanation*, in Figure 2 we present the beeswarm plot concerning the correlation between specific features and the prediction outcome for Android malware detection. Notably, the Flow Bytes/s feature shows a positive correlation with the prediction outcome when its values are low, as indicated by the points on the right side of the vertical line. However, high values of this feature (shown in red) are associated with a negative impact on the prediction, as they appear on the left side of the line. Similarly, the Bwd Bulk Rate Avg feature exhibits a positive correlation with the prediction outcome at low values, located on the right side of the plot. In contrast, higher values of this feature, positioned on the left side, are linked to a negative impact on the prediction.

For the *local explanation* of our models, we used the LIME method that allowed us to understand how the used models predict for a specific instance of data. Figures 3 and 4 show LIME explanations, for instance, predicted as Malicious and as Benign, respectively. From these figures, we can conclude that the features Flow Bytes, Bwd Bulk Rate Avg, Active Max, and Idle Min contribute more to Malware class, whereas the features Packet Length Variance, Bwd Packet Length Std, Flow packets and Fwd Packets Length Std contribute more to the Benign class.

We also analyzed the model’s decisions by generating counterfactual samples for both benign and malware instances using DiCE. From the experiments, we observed that when counterfactual samples

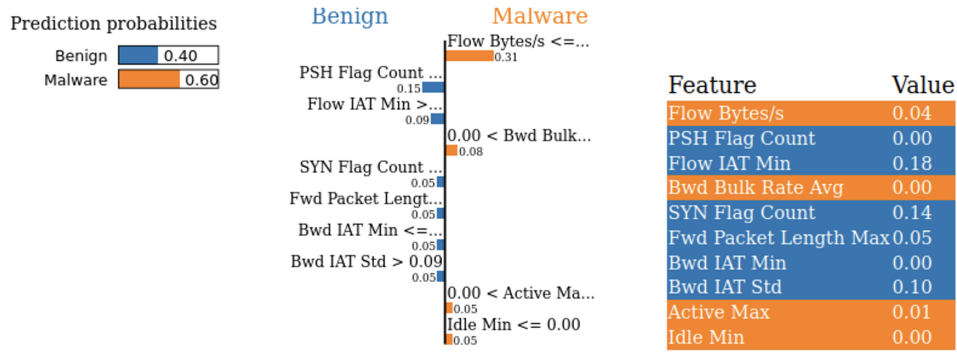


Figure 3: Lime explanation of Malware class

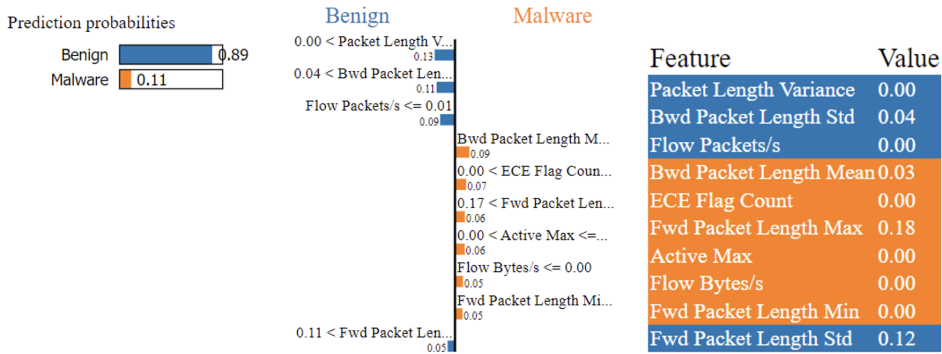


Figure 4: Lime explanation of Benign class

for malware were generated based on a randomly selected benign instance, the features Flow IAT Min, Packet Length Min, and Bwd Init Win Bytes were primarily altered. This implies that these features have a significant role in making reliable predictions. Similarly, when generating counterfactuals for benign samples from malware instances, we found that the Flow IAT Min and Bwd IAT Min feature values were modified to classify the instance as benign.

As discussed in the methodology section, we explored the use of SHAP to identify important features and develop the model based on these ranked features. For this, we evaluated subsets of features by selecting the top 10, 20, 30, 40, 50, and 60 features according to their SHAP values. Figure 5 illustrates the results of feature selection based on the SHAP approach. From Figure 5, we observe that when only the top 10 features are selected, the performance decreases to 0.9026. However, the performance gradually improves as more features are included. Notably, when selecting 50 or 60 features, the model's performance closely aligns with that achieved using the SelectKBest feature selection method. This demonstrates that SHAP can effectively serve as a feature selection mechanism, reducing the feature space while maintaining optimal model performance.

5. Conclusion and Future Work

In this work, we collected a dataset of malicious and benign APKs to study the behavior of Android malware. By applying several ML and DL techniques, we made significant strides in classifying APKs based on network traffic. Our experiments demonstrated that the XGBoost classifier outperformed other algorithms, achieving an impressive weighted F1-score of 93.23% for network traffic analysis. These results underscore the effectiveness of ML and DL models in detecting and distinguishing Android malware. We also provide an explainer to help interpret the model. Our findings contribute to the advancement of security solutions aimed at reducing the risks associated with Android malware. In the future, we plan to integrate additional features such as dynamic analysis and permissions-based data

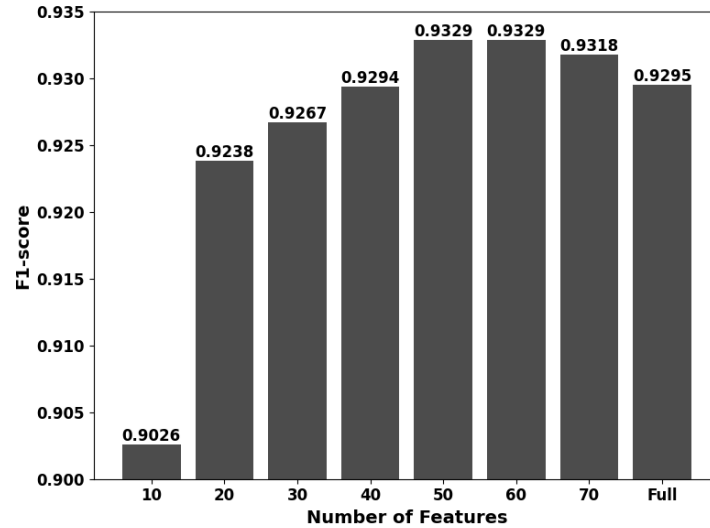


Figure 5: F1-scores of the models using the SHAP-based features

from APKs. Additionally, we aim to understand better malware behavior across diverse platforms and communication channels, such as those exploiting multiple social network APIs, thereby enhancing the robustness and generalizability of malware detection systems [19, 20].

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

- [1] T. Vidas, D. Votipka, N. Christin, All your droid are belong to us: A survey of current android attacks, in: 5th USENIX Workshop on Offensive Technologies (WOOT 11), 2011, p. 10.
- [2] Y. Pan, X. Ge, C. Fang, Y. Fan, A systematic literature review of android malware detection using static analysis, *IEEE Access* 8 (2020) 116363–116379.
- [3] T. Bhatia, R. Kaushal, Malware detection in android based on dynamic analysis, in: 2017 International conference on cyber security and protection of digital services (Cyber security), IEEE, 2017, pp. 1–6.
- [4] S. Holla, M. M. Katti, Android based mobile application development and its security, *International Journal of Computer Trends and Technology* 3 (2012) 486–490.
- [5] A. Sarkar, A. Goyal, D. Hicks, D. Sarkar, S. Hazra, Android application development: a brief overview of android platforms and evolution of security systems, in: 2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC), IEEE, 2019, pp. 73–79.
- [6] F. Buccafurri, G. Lax, S. Nicolazzo, A. Nocera, A privacy-preserving localization service for assisted living facilities, *IEEE Transactions on Services Computing* 13 (2016) 16–29.
- [7] F. Buccafurri, G. Lax, S. Nicolazzo, A. Nocera, Accountability-preserving anonymous delivery of cloud services, in: International Conference on Trust and Privacy in Digital Business, Springer, 2015, pp. 124–135.
- [8] O. Zungur, G. Stringhini, M. Egele, Libspector: Context-aware large-scale network traffic analysis of android applications, in: 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), IEEE, 2020, pp. 318–330.
- [9] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao, M. Conti, Detecting android malware leveraging text semantics of network flows, *IEEE Transactions on Information Forensics and Security* 13 (2017) 1096–1109.

- [10] J. Malik, R. Kaushal, Credroid: Android malware detection by network traffic analysis, in: Proceedings of the 1st acm workshop on privacy-aware mobile computing, 2016, pp. 28–36.
- [11] A. Abderrahmane, G. Adnane, C. Yacine, G. Khireddine, Android malware detection based on system calls analysis and cnn classification, in: 2019 IEEE wireless communications and networking conference workshop (WCNCW), IEEE, 2019, pp. 1–6.
- [12] M. Abuthawabeh, K. W. Mahmoud, Enhanced android malware detection and family classification, using conversation-level network traffic features., *Int. Arab J. Inf. Technol.* 17 (2020) 607–614.
- [13] M. Gohari, S. Hashemi, L. Abdi, Android malware detection and classification based on network traffic using deep learning, in: 2021 7th International Conference on Web Research (ICWR), IEEE, 2021, pp. 71–77.
- [14] N. Zhang, J. Xue, Y. Ma, R. Zhang, T. Liang, Y.-a. Tan, Hybrid sequence-based android malware detection using natural language processing, *International Journal of Intelligent Systems* 36 (2021) 5770–5784.
- [15] P. Vinod, A. Zemmari, M. Conti, A machine learning based approach to detect malicious android apps using discriminant system calls, *Future Generation Computer Systems* 94 (2019) 333–350.
- [16] A. Ananya, A. Aswathy, T. Amal, P. Swathy, P. Vinod, S. Mohammad, Sysdroid: a dynamic ml-based android malware analyzer using system call traces, *Cluster Computing* 23 (2020) 2789–2808.
- [17] S. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, 2017. URL: <https://arxiv.org/abs/1705.07874>. arXiv:1705.07874.
- [18] M. T. Ribeiro, S. Singh, C. Guestrin, "why should i trust you?": Explaining the predictions of any classifier, 2016. URL: <https://arxiv.org/abs/1602.04938>. arXiv:1602.04938.
- [19] F. Buccafurri, G. Lax, S. Nicolazzo, A. Nocera, A model to support multi-social-network applications, in: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", Springer, 2014, pp. 639–656.
- [20] A. Nocera, D. Ursino, Phis: a system for scouting potential hubs and for favoring their "growth" in a social internetworking scenario, *Knowledge-Based Systems* 36 (2012) 288–299.