On Impact of Semantically Similar Apps in Android Malware Datasets

Roopak Surendran, Md. Meraj Uddin
Kerala Security Audit and Assurance Centre
Kerala University of Digital Sciences, Innovation and Technology
Thiruvananthapuram, India
roopak.s,meraju@duk.ac.in

Abstract—Malware authors often reuse program segments from other applications (apps) to perform similar kind of malicious activities, such as information stealing and sending SMS to premium rate numbers. Consequently, multiple semantically similar malware samples may exist within a malware family or dataset. The presence of such apps, unbeknownst to many researchers, may inadvertently influence the evaluation of their Machine Learning (ML) models. In this paper, we investigate the impact of semantically similar apps on the performance measures of ML-based Android malware detectors. Through experiments on the Drebin dataset, we assessed the performance of distinct ML models based on opcode, permissions and API call features of both malware and goodware applications, with and without semantically similar apps. Our findings show that after removing exact duplicate apps from the dataset, the malware detection rate (True Positive Rate) of opcode-based ML models decreased from 0.94 to 0.85, permission-based ML models decreased from 0.94 to 0.90 and API call-based ML models decreased from 0.95 to 0.91. To address this issue, we propose the use of the Euclidean distance metric to identify and eliminate the similar features before evaluating malware detection mechanisms. Implementing this recommendation can enhance the accuracy and reliability of ML-based Android malware detectors.

Index Terms—Android, Malware, Machine Learning

I. INTRODUCTION

It is known that malware apps frequently reuse program segments from previously detected malware applications (apps) [1]. They may also add junk codes or remove redundant codes to alter their signatures. However, despite these changes, these malicious apps often retain the core malicious program segments intended for specific functionalities, such as information stealing and sending SMS. As a result, it is evident that common malicious program segments can be shared among different Android malware families.

Android is an open-source operating system that provides specific APIs (Application Programming Interfaces) for executing sensitive operations, such as sending SMS and making phone calls [2]. For instance, the sendTextMessage() API call enables the sending of SMS messages to others. Initially, malware authors construct malicious program segments designed to perform specific malicious activities by invoking certain APIs in a particular manner. Subsequently, as malware evolves, authors often choose to reuse these existing malicious program segments to replicate the same kind of malicious behavior. Additionally, various frameworks, such as kwetza 1, facilitate the injection of these malicious program segments into benign apps for further propagation.

Many existing works employ Machine Learning (ML) algorithms for malware classification [3]. These approaches often randomly select malware and goodware samples from the dataset to train and test the classifiers. However, some of the malware or goodware apps within the dataset may be semantically similar and share common features. Consequently, the presence of such semantically similar apps can lead to an overestimation of the ML classifier’s performance. Thus, the reported accuracies in these papers may be biased.

In recent years, numerous research papers have explored Android malware detection, categorizing the approaches into static, dynamic, and hybrid analysis. Static analysis relies on source code-level features, such as API calls and permissions, while dynamic analysis focuses on runtime features, including system calls and network packets. Hybrid analysis combines both static and dynamic features. For evaluation, many studies utilize the Drebin dataset, a publicly available collection of 5560 malware apps from 179 malware families [4]. Due to Drebin’s popularity, we also selected this dataset to investigate the impact of semantically similar apps on ML models. The presence of apps with similar program segments in experimental evaluations can introduce bias, necessitating the identification of such apps.

In this study, we examine the influence of semantically similar apps on ML models for malware detection. We assess the performance of ML models on various features of malware and goodware samples, both with and without semantically

1https://github.com/sensepost/kwetza
similar apps. To filter out exactly identical feature vectors from the training datasets, we employ the Euclidean similarity measure [5]. Our findings indicate a drop in the performance of ML models when semantically similar apps are removed from the dataset. This highlights the significance of addressing the presence of semantically similar apps to ensure unbiased results in experimental evaluations.

The remainder of the paper is organized as follows: In Section II, we present the literature review. The performance of ML models in the Drebin dataset, both with and without semantically similar apps, is discussed in Section III. Section IV covers the limitations of our work and outlines potential future directions.

II. REVIEW ON MALWARE DETECTION MECHANISMS

In existing works, ML algorithms are widely used for malware analysis due to their ability to predict malicious behavior in unseen data points [6]. The Drebin dataset is a popular choice for evaluation, containing 5560 malware apps from 179 malware families [4], which includes malicious apps from the MalGenome dataset.

Android malware detection mechanisms typically use either static features, such as API calls, permissions, etc., or dynamic features, such as system calls, network packets, etc., or a combination of both for analysis. Some popular static and dynamic malware analysis mechanisms in the Drebin/MalGenome dataset are discussed below:

   - **Opcode Analysis**
   - **Permission Analysis**
   - **API Calls**

In static analysis, the features associated with the source code of an application is used for malware detection. In [7], the authors used probabilistic ML classifiers trained with API call based features for malware detection. In [8], the app permissions are used as input features of a ML classifier for malware detection. In [9], the data flows are extracted from an application for finding malicious behavior. In [10], the intent based features are used in a ML classifier for malware detection. In [11], n-gram frequencies of opcode level features are used in a ML classifier for malware detection.

In dynamic analysis, an application is executed in an emulator or in a real device and collected the features such as system calls, network packets using the third party utilities. In [12], the runtime API calls are used for malware detection. In [13], authors used system metric level features such as CPU, memory usages for malware detection. In [14], the authors used system calls as features of supervised binary classifiers for malware detection. In [15], the authors used network packets as features for malware detection.

Despite these mechanisms, many authors use entire samples from the dataset for experimental purposes, without accounting for the fact that malware authors often reuse existing malicious code to generate new variants. As a result, these datasets may contain several semantically similar apps. In this paper, we investigate the impact of semantically similar apps on ML models for malware detection. To filter out exactly identical feature vectors from the training datasets, we employ the Euclidean similarity measure [5]. Our experimental evaluations demonstrate that the presence of semantically similar apps can lead to an overestimation of the performance of ML models in malware detection.

III. EVALUATION OF DREBIN MALWARE SAMPLES WITH AND WITHOUT SEMANTICALLY SIMILAR APPS

In this section, we conducted a study on the impact of semantically similar applications on ML-based malware detection mechanisms. For our experiments, we collected 5500 goodware samples from the Androzoo dataset [16] and 5500 malware samples from the Drebin dataset [4]. To ensure an unbiased comparison, we selected goodware samples from the same period and API level as the Drebin dataset (2010 to 2012).

The statistics of apps in the datasets are presented in Table I. We evaluated all the datasets using ML algorithms trained with various types of features. The used features are given as follows:

1. Opcodes;
2. Permissions;
3. API Calls.

| TABLE I |
| --- |
| **DISTRIBUTION OF SEMANTICALLY DISSIMILAR APPS** |
| Dataset | Number of Malware Samples | Number of Goodware Samples |
| Duplicated Dataset | 5500 | 5500 |
| Filtered Dataset | 2642 | 4655 |

A. Opcode Analysis

In this section, we re-implemented the opcode frequency-based malware detection mechanism in our datasets to analyze the impact of semantically similar apps. The list of opcodes used for this analysis is provided in Table II.

To extract the counts of opcode-based features from the malware and goodware apps in the dataset, we constructed a Comma Separated Value (CSV) file. This CSV file was then supplied to the Weka framework and tested with ML classifiers using the 10-fold cross-validation technique. Among the classifiers tested, the random forest classifier showed high accuracy, so we used it for assessing the impact of semantically similar apps in malware detection.

As shown in Table III, after removing the semantically similar apps from the dataset, the malware detection rate (True Positive Rate, TPR) dropped from 0.94 to 0.85. This result suggests that the presence of semantically similar apps significantly affects the performance of the malware detection mechanism based on opcode frequencies.

B. Permission Analysis

In this section, we re-implemented the permission-based malware detection mechanism in our datasets (with and without semantically similar apps) to analyze the impact of these apps
### TABLE II
**LIST OPCODES IN ANDROID OPERATING SYSTEM**

| Hex Value | Opcode          | Hex Value | Opcode          | Hex Value | Opcode          | Hex Value | Opcode          |
|-----------|-----------------|-----------|-----------------|-----------|-----------------|-----------|-----------------|
| 00        | nop             | 01        | move            | 02        | move/from 16    | 03        | move/16         |
| 04        | move-wide/from  | 05        | move-wide/16    | 06        | move-object     | 08        | move-object/from 16 |
| 09        | move-object/16  | 0A        | move-result     | 0B        | move-result-wide| 0C        | move-result-object |
| 0D        | move-exception  | 0E        | return-void     | 0F        | return          | 10        | return-wide     |
| 11        | return-object   | 12        | const/4         | 13        | const/16        | 14        | const           |
| 15        | const           | 16        | const-wide/16   | 17        | const-wide/32   | 18        | const-wide      |
| 19        | const-wide/high 16 | 1A       | const-string    | 1B        | const-string-jumbo | 1C       | const-class     |
| 1D        | monitor-enter   | 1E        | monitor-exit    | 1F        | check-cast      | 20        | instance-of     |
| 21        | array-length    | 22        | new-instance    | 23        | new-array       | 24        | filled-new-array |
| 25        | filled-new-array-range | 26    | fill-array-data | 27        | throw           | 28        | goto            |
| 29        | goto/16         | 2A        | goto/32         | 2B        | packed-switch   | 2C        | sparse-switch   |
| 2D        | cmpf-long       | 2E        | cmpg-float      | 2F        | cmpf-double     | 30        | cmpg-double     |
| 31        | cmp-long        | 32        | if-eq           | 33        | if-ne           | 34        | if-lt           |
| 35        | if-ge           | 36        | if-gt           | 37        | if-le           | 38        | if-eqz          |
| 39        | if-nez          | 3A        | if-gez          | 3B        | if-fgez         | 3C        | if-gez          |
| 3D        | if-lez          | 3E        | unused_3E       | 3F        | unused_3F       | 40        | unused_40       |
| 41        | unused_41       | 42        | unused_42       | 43        | unused_43       | 44        | aget            |
| 45        | aget-wide       | 46        | aget-object     | 47        | aget-object     | 48        | aget-byte       |
| 49        | aget-char       | 4A        | aget-short      | 4B        | aget-short      | 4C        | aget-byte       |
| 4D        | aget-byte       | 4E        | aget-byte       | 4F        | aget-byte       | 50        | aget-char       |
| 51        | aget-short      | 52        | aget          | 53        | aget-wide       | 54        | aget-object     |
| 55        | aget-boolean    | 56        | aget-byte      | 57        | aget-byte      | 58        | aget-byte       |
| 59        | aget-char       | 5A        | aget-byte      | 5B        | aget-byte      | 5C        | aget-byte       |
| 61        | aget-byte       | 5E        | aget-byte      | 5F        | aget-byte      | 60        | gset            |
| 65        | aget-byte       | 6A        | aget-byte      | 6B        | gset            | 63        | aget-byte       |
| 69        | aget-byte       | 6E        | aget-byte      | 6F        | aget-byte      | 6G        | aget-byte       |
| 71        | aget-byte       | 72        | aget-byte      | 73        | aget-byte      | 74        | aget-byte       |
| 75        | aget-byte       | 76        | aget-byte      | 77        | aget-byte      | 78        | aget-byte       |
| 79        | aget-byte       | 7A        | aget-byte      | 7B        | aget-byte      | 7C        | aget-byte       |
| 7D        | aget-byte       | 7E        | aget-byte      | 7F        | aget-byte      | 80        | aget-byte       |
| 81        | aget-byte       | 82        | aget-byte      | 83        | aget-byte      | 84        | aget-byte       |
| 85        | aget-byte       | 86        | aget-byte      | 87        | aget-byte      | 88        | aget-byte       |
| 89        | aget-byte       | 8A        | aget-byte      | 8B        | aget-byte      | 8C        | aget-byte       |
| 8D        | aget-byte       | 8E        | aget-byte      | 8F        | aget-byte      | 90        | aget-byte       |
| 91        | aget-byte       | 92        | aget-byte      | 93        | aget-byte      | 94        | aget-byte       |
| 95        | aget-byte       | 96        | aget-byte      | 97        | aget-byte      | 98        | aget-byte       |
| 99        | aget-byte       | 9A        | aget-byte      | 9B        | aget-byte      | 9C        | aget-byte       |
| 9D        | aget-byte       | 9E        | aget-byte      | 9F        | aget-byte      | A0        | aget-byte       |
| A1        | or-long         | A2        | or-long         | A3        | or-long         | A4        | or-long         |
| A5        | ush-byte        | A6        | ush-byte        | A7        | ush-byte        | A8        | ush-byte        |
| A9        | div-float       | AA        | div-float       | AB        | div-float       | AC        | div-float       |
| AD        | mul-double      | AE        | div-double      | AF        | div-double      | B0        | div-double      |
| B1        | sub-int/2addr   | B2        | mul-int/2addr   | B3        | mul-int/2addr   | B4        | mul-int/2addr   |
| B5        | and-int/2addr   | B6        | or-int/2addr    | B7        | xor-int/2addr   | B8        | shl-int/2addr   |
| B9        | shr-int/2addr   | BA        | ush-int/2addr   | BB        | add-long/2addr  | BC        | sub-long/2addr  |
| BD        | mul-long/2addr  | BE        | div-long/2addr  | BF        | rem-long/2addr  | C0        | and-long/2addr  |
| C1        | or-long/2addr   | C2        | xor-long/2addr  | C3        | shl-long/2addr  | C4        | shl-long/2addr  |
| C5        | ush-long/2addr  | C6        | add-float/2addr | C7        | sub-float/2addr | C8        | mul-float/2addr |
| C9        | divf-long/2addr | CA        | rem-float/2addr | CB        | add-double/2addr | CC        | sub-double/2addr |
| CD        | mul-double/2addr | CE  | div/2double/2addr | CF  | rem/2double/2addr | D0  | add-int/16 |
| D1        | add-int/16      | D2        | sub-int/16      | D3        | mul-int/16      | D4        | div-int/16      |
| D3        | and-int/16      | D6        | or-int/16       | D7        | xor-int/16      | D8        | add-int/16      |
| D9        | sub-int/16      | DA        | mul-int/16      | DB        | div-int/16      | DC        | rem-int/16      |
| DD        | and-int/16      | DE        | or-int/16       | DF        | xor-int/16      | EG        | shl-int/16      |
| E1        | shr-int/16      | E2        | ush-int/16      | E3        | unused_E3       | E4        | unused_E4       |
| E5        | unused_E5       | E6        | unused_E6       | E7        | unused_E7       | E8        | unused_E8       |
| E9        | unused_E9       | EA        | unused_EA       | EB        | unused_EB       | EC        | unused_EC       |
| ED        | unused_ED       | EE        | exec-ecline     | EF        | unused_EF       | F0        | invoke-direct-empty|
| F1        | unused_F1       | F2        | iget-wide/quick | F3        | iget-object/quick | F4        | iget-object/quick|
| F5        | iput-quick      | F6        | iput-wide/quick | F7        | iput-object/quick | F8        | invoke-virtual-quick|
| F9        | invoke-quick/1f  | FA        | invoke-super-f1 | FB        | invoke-super-quick/1f | FC        | invoke-super-quick/1f|
| FD        | unused_FD       | FE        | unused_FE       | FF        | unused_FF       |
TABLE III
K-FOLD CROSS VALIDATION RESULTS IN OPCODE FREQUENCY BASED CLASSIFIER

| Dataset          | TPR    | FPR    | Accuracy | Precision | F1Score |
|------------------|--------|--------|----------|-----------|---------|
| Duplicated Dataset | 0.94   | 0.02   | 0.98     | 0.96      | 0.97    |
| Filtered Dataset  | 0.85   | 0.02   | 0.96     | 0.93      | 0.91    |

TABLE IV
SELECTED PERMISSIONS FOR MALWARE DETECTION

| SL.No | Permissions                      | SL.No | Permissions                      |
|-------|----------------------------------|-------|----------------------------------|
| 1     | READ_PHONE_STATE                 | 2     | WRITE_CONTACTS                   |
| 3     | CALL_PHONE                       | 4     | READ_CONTACTS                    |
| 5     | INTERNET                         | 6     | SEND_SMS                         |
| 7     | DISABLE_KEYGUARD                 | 8     | PROCESS_OUTGOING_CALLS          |
| 9     | RECEIVE_BOOT_COMPLETED           | 10    | READ_SMS                         |
| 11    | FACTORY_TEST                     | 12    | DEVICE_POWER                      |
| 13    | HARDWARE_TEST                    | 14    | CHANGE_WIFI_STATE                |
| 15    | GET_ACCOUNTS                     | 16    | READ_HISTORY_BOOKMARKS           |
| 17    | WRITE_APN_SETTINGS               | 18    | MODIFY_PHONE_STATE               |
| 19    | WRITE_HISTORY_BOOKMARKS          | 20    | ACCESS_LOCATION                   |
| 21    | EXPAND_STATUS_BAR                | 22    | WRITE_EXTERNAL_STORAGE           |
| 23    | RECEIVE_SMS                      | 24    | WRITE_SMS                        |
| 25    | ACCESS_WIFI_STATE                | 26    | MODIFY_AUDIO_SETTINGS            |
| 27    | ACCESS_NETWORK_STATE             | 28    | WRITE_SETTINGS                   |
| 29    | READ_EXTERNAL_STORAGE            | 30    | ACCESS_MOCK_LOCATION             |
| 31    | USE_CREDENTIALS                  | 32    | HARDWARE_TEST                    |
| 33    | VIBRATE                          | 34    | READ_LOGS                        |
| 35    | CHANGE_NETWORK_STATE             | 36    | ACCESS_GPS                       |
| 37    | WAKE_LOCK                        | 38    | ACCESS_COURSE_UPDATES            |
| 39    | ACCESS_LOCATION_EXTRA_COMMANDS   | 40    | ACCESS_FINE_LOCATION             |
| 41    | GET_TASKS                        | 42    | RESTART_PACKAGES                 |
| 43    | MOUNT_UNMOUNT_FILESYSTEMS        | 44    | INSTALL_PACKAGES                 |
| 45    | KILL_BACKGROUND_PROCESS          |       |                                  |

on the dataset. We used the key permission-based features mentioned in Roopak et al. [17], and the list of permission-based features can be found in Table IV.

To extract the permission-based features of malware and goodware apps in the dataset, we constructed a Comma Separated Value (CSV) file. This CSV file was then supplied to the Weka framework [18] and tested with ML classifiers using the 10-fold cross-validation technique. Among the classifiers tested, the random forest classifier yielded high accuracy, so we used it to assess the impact of semantically similar apps in malware detection.

As shown in Table V, after removing the semantically similar apps from the dataset, the malware detection rate (True Positive Rate, TPR) dropped from 0.94 to 0.90. This result indicates that the presence of semantically similar apps can affect the performance of the malware detection mechanism.

C. API Call Analysis

In this section, we re-implemented the API call-based malware detection mechanism in our datasets to analyze the impact of semantically similar apps in the dataset. We reused the key API call-based features mentioned in Roopak et al. [17], and the list of API call-based features can be found in Table VI.

To extract the API call-based features of malware and goodware apps in the dataset, we constructed a Comma Separated Value (CSV) file. This CSV file was then supplied to the Weka framework and tested with ML classifiers using the 10-fold cross-validation technique. Among the classifiers tested, the random forest classifier yielded high accuracy, so we used it for assessing the impact of semantically similar apps in malware detection.

As shown in Table VII, after removing the semantically similar apps from the dataset, the malware detection rate (True Positive Rate, TPR) dropped from 0.95 to 0.91. This result indicates that the presence of semantically similar apps can significantly affect the performance of the malware detection mechanism.

IV. DISCUSSION AND CONCLUSIONS

In this work, we conducted an assessment of the impact of semantically similar apps in an Android malware dataset. Our findings reveal that the presence of semantically similar apps has a notable influence on the performance of ML models during evaluation. As a result, we strongly recommend filtering out all semantically similar apps before conducting any evaluation to ensure unbiased and accurate results in Android malware detection.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
In this work, we considered only static features for identifying the semantically similar apps. It is known that all the static features can be hidden using dynamic loading technique [20]. That is, malicious code is stored inside the application as encrypted form and it gets decrypted during the execution time. Hence, in order to overcome this limitation, in future, we will consider dynamic features along with these static features for identifying semantically similar apps.

REFERENCES

[1] X. Sun, Y. Zhongyang, Z. Xin, B. Mao, and L. Xie, “Detecting code reuse in android applications using component-based control flow graph,” in *IFIP international information security conference*, pp. 142–155, Springer, 2014.

[2] T. McDonnell, B. Ray, and M. Kim, “An empirical study of api stability and adoption in the android ecosystem,” in *2013 IEEE International Conference on Software Maintenance*, pp. 70–79, IEEE, 2013.

[3] J. Sahs and L. Khan, “A machine learning approach to android malware detection,” in *2012 European Intelligence and Security Informatics Conference*, pp. 141–147, IEEE, 2012.

[4] D. Arp, M. Speitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, “Drebin: Effective and explainable detection of android malware in your pocket.,” in *Ndss*, vol. 14, pp. 23–26, 2014.

[5] G. Qian, S. Sural, Y. Gu, and S. Pramanik, “Similarity between euclidean and cosine angle distance for nearest neighbor queries,” in *Proceedings of the 2004 ACM symposium on Applied computing*, pp. 1232–1237, 2004.

[6] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, “A review of android malware detection approaches based on machine learning,” *IEEE Access*, vol. 8, pp. 124579–124607, 2020.

[7] L. Cen, C. S. Gates, L. Si, and N. Li, “A probabilistic discriminative model for android malware detection with decompiled source code,” *IEEE Transactions on Dependable and Secure Computing*, vol. 12, no. 4, pp. 400–412, 2014.

[8] P. Rovelli and Y. Vigfusson, “Pmds: Permission-based malware detection system,” in *International conference on information systems security*, pp. 338–357, Springer, 2014.

[9] S. Arzt, S. Rasthofer, C. Fritz, E. Boddin, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, and P. McDaniel, “Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps,” *Acm Sigplan Notices*, vol. 49, no. 6, pp. 259–269, 2014.

[10] K. Xu, Y. Li, and R. H. Deng, “Iccdetector: Icc-based malware detection on android,” *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 6, pp. 1252–1264, 2016.

[11] G. Canfora, A. De Lorenzo, E. Medvet, F. Mercaldo, and C. A. Visaggio, “Effectiveness of opcode ngrams for detection of multi family android malware,” in *2015 10th International Conference on Availability, Reliability and Security*, pp. 333–340, IEEE, 2015.

[12] T. Kim, B. Kang, and E. G. Im, “Runtime detection framework for android malware,” *Mobile Information Systems*, vol. 2018, 2018.

[13] J. Milosevic, M. Malek, and A. Ferrante, “A friend or a foe? detecting malware using memory and cpu features.,” in *SECRYPT*, pp. 73–84, 2016.

[14] G. Canfora, E. Medvet, F. Mercaldo, and C. A. Visaggio, “Detecting android malware using sequences of system calls,” in *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, pp. 13–20, 2015.

[15] M. Zaman, T. Siddiqui, M. R. Amin, and M. S. Hossain, “Malware

**TABLE V**

| Dataset         | TPR | FPR | Accuracy | Precision | F1Score |
|-----------------|-----|-----|----------|-----------|---------|
| Duplicated Dataset | 0.94 | 0.05 | 0.95     | 0.95      | 0.95    |
| Filtered Dataset  | 0.90 | 0.05 | 0.93     | 0.93      | 0.93    |

**TABLE VI**

| SL.No | API Calls                  | SL.No | API Calls                  |
|-------|----------------------------|-------|----------------------------|
| 1     | getLongitude               | 18    | getDisplayMessageBody      |
| 2     | getLongitude               | 19    | getPackageInfo             |
| 3     | loadClass                  | 20    | getLastKnownLocation       |
| 4     | getMessage                  | 21    | getAppPackageName           |
| 5     | getMethod                  | 22    | getCookies                 |
| 6     | getClassLoader             | 23    | isEnabledProvider          |
| 7     | GetLongitude               | 24    | getClassOperatorName       |
| 8     | getPackageInfo             | 25    | getDeviceId                |
| 9     | createFromPdu              | 26    | getCerStatus               |
| 10    | getInputstream             | 27    | getSimSerialNumber         |
| 11    | getoutputstream            | 28    | getLine1Number             |
| 12    | killProcess                | 29    | exec                       |
| 13    | abortBroadcast             | 30    | RequestFocus               |
| 14    | getSubscriptionId          | 31    | getAppPackageName           |
| 15    | endl safety                | 32    | setSerialNumber            |
| 16    | getClassOriginatingAddress| 33    | getSession                |
| 17    | sendTextMessage            | 34    | setCredential             |

**TABLE VII**

| Dataset         | TPR | FPR | Accuracy | Precision | F1Score |
|-----------------|-----|-----|----------|-----------|---------|
| Duplicated Dataset | 0.94 | 0.04 | 0.94     | 0.94      | 0.94    |
| Filtered Dataset  | 1    |     |          |           |         |
detection in android by network traffic analysis,” in 2015 international conference on networking systems and security (NSysS), pp. 1–5, IEEE, 2015.

[16] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, “Androzoo: Collecting millions of android apps for the research community,” in 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), pp. 468–471, IEEE, 2016.

[17] R. Surendran, T. Thomas, and S. Emmanuel, “A tan based hybrid model for android malware detection,” Journal of Information Security and Applications, vol. 54, p. 102483, 2020.

[18] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The weka data mining software: an update,” ACM SIGKDD explorations newsletter, vol. 11, no. 1, pp. 10–18, 2009.

[19] A. Liaw, M. Wiener, et al., “Classification and regression by randomforest,” R news, vol. 2, no. 3, pp. 18–22, 2002.

[20] V. Rastogi, Y. Chen, and X. Jiang, “Droidchameleon: evaluating android anti-malware against transformation attacks,” in Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security, pp. 329–334, 2013.