# Detection of malware applications from centrality measures of syscall graph

**Article** *in* Concurrency and Computation Practice and Experience · January 2022

DOI: 10.1002/cpe.6835

CITATIONS
0

READS
89

**2 authors:**

Roopak Surendran
Indian Institute of Information Technology and Management - Kerala
**10** PUBLICATIONS **62** CITATIONS
SEE PROFILE

Tony Thomas Kallivayalil
Kerala University of Digital Sciences Innovation and Technology
**64** PUBLICATIONS **533** CITATIONS
SEE PROFILE

Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

# Detection of Malware Applications from Centrality Measures of Syscall Graph

Roopak Surendran* | Tony Thomas

[1] School of Computer Science and Engineering, Kerala University of Digital Sciences, Innovation and Technology, Thiruvananthapuram, India

**Correspondence**
*Roopak Surendran Email: roopak.res15@duk.ac.in

**Summary**

These days it is found that malware authors tend to create new variants of existing Android malware by using various kinds of obfuscation techniques. These kinds of obfuscated malware applications can bypass all the current anti-malware products which rely on static analysis techniques to detect the malicious behavior. Hence, it is essential to develop innovative dynamic analysis mechanisms for Android malware detection. It is known that, the malicious behavior of statically obfuscated malware applications can get reflected in the system call (syscall) trace generated by them. Most of the existing syscall based mechanisms depend only on the features derived from the syscall counts for malware detection. These syscall count related features are inadequate to capture many other useful characteristics related to the syscalls in a sequence. In order to overcome this limitation, we modeled the syscall trace of an application as an ordered graph which enabled to infer various kinds of features in the form of centrality measures related to that syscall trace of the application. Then, these centrality measures are fed to an ML model to predict the malicious behavior. From the implementation results, we found that our mechanism can detect malware apps with an accuracy of 0.99.

**KEYWORDS:**
Android, Malware, Syscalls, Graph centralities

## 1 | INTRODUCTION

Almost 98% of smartphone malware are designed for Android devices. Kaspersky lab and Interpol survey reported that malware applications like trojan SMS and trojan banker may cause financial losses to a user. Trojan SMS can send SMS messages to premium rate numbers in the background and trojan banker can steal the online banking details of a user without the user's knowledge[1]. There are three common ways in which a malware can enter into an Android device[2]. They are through drive by-download attack, update attack and repacking attack. In drive-by-download attack, the attacker develops a malware and hosts it on a web server controlled by him. Then, the attacker sends the link of this web server to several people through SMS or e-mail or social networks. When a user visits that web page, the malicious application gets installed into the device. In update attack, a benign app is later updated with malware functionality[2]. In repacking attack, the malware creator repacks a benign application by injecting malicious code into it[2].

The current anti-malware products are still based on static and signature based mechanisms. Static analysis is a malware detection method by analyzing the call graph and the flow graph of the source code[3]. In signature based analysis, the hash

value of an application is compared with the signatures of known malicious applications for identifying the malicious behavior. These mechanisms can be evaded by code obfuscation attacks[4]. In code obfuscation attacks, a malware author can perform operations such as repacking, identifier renaming, code reordering etc. to alter the static signature of a malware application. Hence, it is essential to develop innovative malware detection mechanisms based on dynamic analysis for accurate malware detection. Dynamic analysis mechanisms consider the runtime information such as system metrics, network level information, syscalls etc. for detecting malicious activity. Malicious applications typically invoke sensitive APIs in an automated manner to perform privileged operations[5]. This automated execution of API calls gets reflected in the syscall trace of the application. It is known that, the malicious behavior of statically obfuscated malware applications can get reflected in the syscall trace generated by them. Hence, syscalls are considered as the most effective features for capturing malicious activity which are performed in the background.

Existing syscall analysis mechanisms rely on either Machine Learning (ML) algorithms, deep learning algorithms or identifying the presence of a syscall subsequence in the syscall trace for malware detection[6,7,8]. In syscall subsequence analysis mechanism, the contiguous relationship between syscalls in the sequence is considered for malware detection. It is possible for a malware author to alter a syscall subsequence by rearranging the source code without changing the semantics. Existing syscall based mechanisms which rely on traditional machine learning algorithms depend only on the features derived from the syscall counts for malware detection. That is, they are inadequate to capture many useful information related to the syscalls in the syscall sequence. In order to overcome this limitation, we modeled the syscall trace of an application as the ordered graph which enabled us to infer various kinds of features in the form of centrality measures related to that syscall trace. Then, these centrality measures are fed to an ML classifier to predict the malicious behavior. The main contributions of this paper is given below:

- Proposing a novel feature representation for Android applications using graph centrality measures;
- Eliminating multicollinearity problem by removing features having high VIF values (greater than 10);
- Proposing a novel dynamic Android malware detection mechanism using centrality measures as feature vector.

A syscall trace can be modeled as an ordered graph with the set of syscalls as vertices and their adjacency relationships as edges. syscall ordered graph makes possible to infer various kinds of centrality measures such as eigen, betweenness and closeness. Multicollinearity problem may occur while using all these centrality measures in a single ML classifier for malware detection[9]. For eliminating the multicollinearity problem, during the training time, we ignore the features those have high variable inflation factor (VIF) value greater than 10 (features which are highly correlated to other features in the model)[10]. Then, these features are given as inputs to a trained ML classifier for identifying the malicious behavior. From the implementation results, we found that our mechanism can detect malware apps with an accuracy of 0.99.

The rest of the paper is organized as follows. Literature review is given in Section 2. In Section 3, the proposed malware detection mechanism is given. In Section 4, the performance analysis of our mechanism is given. In Section 5, conclusions and future directions for the research are given.

## 2 | LITERATURE REVIEW

In static analysis, malware apps can be detected by analyzing their code level features. In[11], Aafer et al. extracted the API level features from the static source code of an application using reverse engineering tools and used them for characterizing the malware applications. DroidMiner[12] extracts malicious behavioral paths from the static source code of applications for detecting malware applications. Drebin[13] uses ML algorithms such as SVM to detect malicious applications from the statically obtained features such as API, permissions, intents etc. Dendroid[14] uses text mining techniques to statistically analyze the code structures found in malware families. FalDroid[15] extracts frequent subgraphs found in a malware families and use them for detecting unknown malware apps. In[16], the app permissions are extracted from the manifest file and used them for identifying the malicious behavior. Riskranker[17] identifies the anomalies present in the known malicious applications for identifying zero day malware applications. In Mudflow[18], an ML classifier trained with data flow based features is used for detecting the malware applications. In[19] , a trained ML classifier is used for detecting malicious behavior from the ICC (Inter Component Communication) patterns. In[20], an ML classifier trained with intent based features is used for detecting the malicious behavior. In DroidDet[21], Zhu et al. extracted static features such as sensitive API calls, permissions etc. of an application and used them

in a Rotation Forest classifier for identifying the malicious behavior. In[22], Hou et al. proposed a mechanism to detect Android malware apps by analyzing the API calls in it. In this approach, a trained Deep Belief Network (DBN) classifier is used to predict whether the application is a malware or not based on the API calls in it. In[23], Mclaughlin et al suggested a CNN based mechanism to detect Android malware apps from opcode sequences. In[24], Cai et al. suggested a Graph Convolutional Neural Network (GCN) based method to detect Android malware apps by analyzing static API function call graphs. Malware applications can easily bypass all the static malware analyzers by employing various obfuscation methods.

Dynamic analysis rely on runtime features such as syscalls, sensitive API calls, system metrics such as CPU usage, network packet level information etc. for malware detection. In[25], a sandbox is designed to log the sensitive API calls generated by an application for identifying the malicious behavior. Sometimes goodware apps invoke sensitive API calls such as sendTextMessage() for legitimate purpose. Such kinds of goodware apps are falsely alarmed as malware by these mechanisms. In[26], the network packet based features are used for malware detection. However, the malware apps do not require internet connection for certain malicious activities such as sending SMS. In such cases, false negatives may occur. In Andromaly[27], the system metric based features such as CPU memory and battery utilization are used for identifying the malicious behavior. Certain gaming apps may overuse the system resources for its efficient working. In such cases, false positives may occur in these mechanisms.

Malware apps do not require user triggers for invoking API calls to perform the malicious activities. This automated execution of API calls will get reflected in the corresponding syscall sequence. Hence, syscall sequences can be treated as the best feature to detect the malicious behavior. The existing syscall based malware detection methods analyze the frequency, density and co-occurrences of the syscalls generated by an application for identifying malware behavior. In Crowdroid[28], a client application traces the syscalls from various devices and sends to a cloud based server for malware detection. The server analyzes the frequencies of syscalls in each trace and uses k-means clustering algorithm to find out the malicious traces. In[6], syscall frequency and dependency information are used for identifying whether an application is a malware or not. In[29], the both syscall frequency and co-occurrence are used for identifying the malicious behavior. In[30], syscall densities are used as input features of a supervised binary classifier for finding malicious behavior. In[7], the syscall Markov chain transition probabilities are used as input features of supervised binary classifiers for identifying malicious behavior. In[31], LSTM classifiers are used to predict the malicious behavior from a syscall trace produced by an application. In[32], Bernadi et al sugegsted a method to detect Android malware apps by analyzing syscall Execution Fingerprint (SEF) based features. This approach needs multiple syscall logs of a single application for identifying its malicious behavior. In[33], Vinod et al. suggested accurate feature selection methods for syscall based malware detection. They obtained a very high accuracy in malware detection. In[34], Roopak et al. suggested Graph Signal Processing (GSP) based method for compact representation of syscall trace of an application. They found that the signal values in a very few syscalls were enough for malware detection. In Vizmal[35], authors constructed images from multiple syscall execution traces of malicious applications and located the portions of malicious behavior from it. In[36], Teenu et al. suggested a GCN based method to detect malware apps by analyzing syscall digraphs. The limitations of existing syscall based malware detection methods are given in Table 1.

Deep learning models are more accurate than the traditional ML models in many problems. However, deep learning based methods are accurate only if it is trained with a very large number of malware and goodware apps. Moreover, the training complexity of deep learning algorithms are higher than that of traditional ML models. The performances of existing deep learning based mechanisms are given in Table 2. Sometimes, sufficient number of evolving malware samples are not available for retraining the deep learning models. In such cases, traditional machine learning algorithms perform better than deep learning based models for detecting evolving malware apps. The accuracy of traditional ML models are determined by their feature representation mechanisms. Hence, it is required to design and develop the best feature representation mechanism that contains enough information to classify malware samples. In this paper, we propose the novel graph centrality based features that capture diverse information regarding the system call sequence. These centrality based features are given as input to the ML classifier for malware detection. These various kinds of centrality measures are derived from various features such as system call occurrences (frequency related feature), system call adjacency relationships and system call distance relationships. Hence, the centrality based features contain more information than the system call count related features. Therefore, these centrality based features contain sufficient information about the malware applications. Further, the proposed system may be used as a cloud based service with the detection mechanism running in the cloud for resource constrained Android devices.

Most of the existing machine learning based dynamic analysis methods use features related to syscall frequencies such as count, denisty, TF-IDF or relative frequencies such as occurrences and transition probability matrix for malware detection. These standalone features may not capture all the characteristics of malware. Hence, they can lead to classification errors, especially in obfuscated malware families. In order to overcome this limitation, we propose a syscall graph centrality based

approach for detecting Android malware applications. We modeled a syscall trace generated by a malware application as a digraph which enables to infer many information related to the syscall in terms of various centrality measures (contains features other than count related syscalls). These centrality measures are given as inputs of an ML classifier for malware detection.

**TABLE 1** Drawbacks of Existing Syscall Based Dynamic Malware Detection Methods

| **Category** | **DrawBacks** |
|---|---|
| Zaman et al.[37] | Unable to detect premium rate SMS senders |
| Shabtai et al.[27] | Low accuracy |
| Jamrozik et al.[25] | High false positives |
| Burguera et al.[28] | Syscall relationship is missing |
| Xiao et al.[7] | High dimensionality of feature vector |
| Vinod et al.[33] | Syscall relationship is missing |
| Dmjsevac et al.[6] | High dimensionality of feature vector |
| Hou et al.[38] | Require a large training dataset |
| Xiao et al.[29] | High dimensionality of feature vector |
| Bhandari et al.[39] | Feature extraction cost is high |
| Xiao et al.[7] | High dimensionality of feature vector |
| Bernadi et al.[32] | Requires multiple system call logs |
| Vizmal[35] | Requires multiple system call logs |
| Roopak et al.[34] | Prone to syscall replacement attacks |
| Xiao et al.[31] | Prone to syscall reordering attacks |

**TABLE 2** Performances of Deep Learning Based Malware Detection Mechanisms

| Mechanism | Analysis | Features | Dataset Size | Training Time | Accuracy |
|---|---|---|---|---|---|
| Mclaughlin et al.[23] | Static | Opcodes | 6000 | 1500 seconds | 0.98 |
| DroidDelver[22] | Static | API Calls | 5000 | N/A | 0.94 |
| Hou et al.[38] | Dynamic | System Calls | 3000 | N/A | 0.94 |
| Xiao et al.[31] | Dynamic | System Calls | 7103 | 18500 seconds | 0.94 |
| Teenu et al.[31] | Dynamic | System Calls | 2130 | N/A | 0.923 |

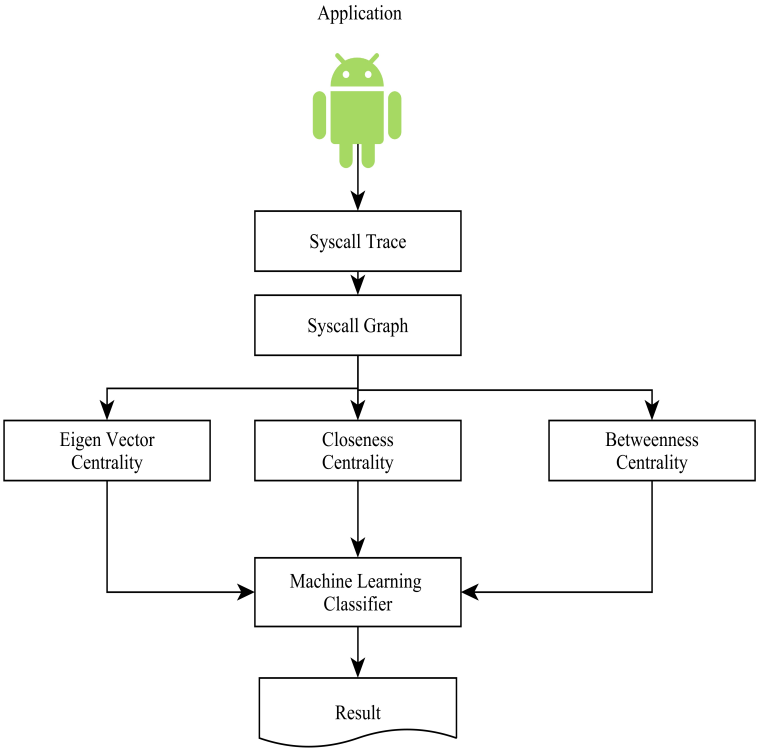
## 3 | METHODOLOGY

In our mechanism, we model the syscall trace of an application as a directed graph and extracts centrality based features from it. Then, we employ ML algorithms with these features for malware detection. The architecture of our mechanism is given in Figure 1.

### 3.1 | Running the Application and Modelling Syscall as a Directed Graph

Initially, we collect the syscall trace of an application and model it as a digraph. According to Wang et al.[40], there are some irrelevant syscalls that do not hold any significant influence in the behavior of an application. Hence, we refine the syscall log by eliminating the irrelevant syscalls such as memory management syscalls, error syscalls and information maintenance syscalls from it. The list of $n$ significant syscalls $S$ are given in Table 3. Let $X = \{X_1, X_2, \dots\}$ be the refined syscall sequence, where $X \in S$. According to Yi et al.[41] , there exists a control flow dependency among syscalls in a syscall trace. Hence, a

**FIGURE 1** Architecture of Our mechanism




syscall $X_i$ can determine the occurrence of the next syscall $X_{i+1}$. $X$ can be represented as a weighted ordered directed graph $G = (S, E, A)$, where the vertex set is $S$ the set of syscalls, $E = \{e_{i,j} : i, j = 1, 2, 3, \ldots, n\}$ is the set of directed edges and $A = (a_{i,j})_{n \times n}$ is the weighted adjacency matrix. Here, an edge $e_{i,j}$ exists only if $S_j$ occurs immediately after $S_i$ in $X$ and $a_{i,j}$ is the number of invocations from $S_i$ to $S_j$ in $X$. The ordered digraph of Walkinwat malware application after running for one minute is given in Figure 2.

**TABLE 3** List of Significant Syscalls

| Alternative Name | Syscall | Alternative Name | Syscall | Alternative Name | Syscall |
|---|---|---|---|---|---|
| A | recvfrom | B | write | C | ioctl |
| D | read | E | sendto | F | dup |
| G | writev | H | pread | I | close |
| J | socket | K | bind | L | connect |
| M | mkdir | N | access | O | chmod |
| P | open | Q | rename | R | fchown32 |
| S | unlink | T | pwrite | U | unmask |
| V | lseek | W | fcntl | X | recvmsg |
| Y | sendmsg | Z | epoll | A1 | dup2 |
| A2 | fchown | A3 | readv | A4 | chdir |

**FIGURE 2** Representing Walkinwat Malware App as Syscall Directed Graph

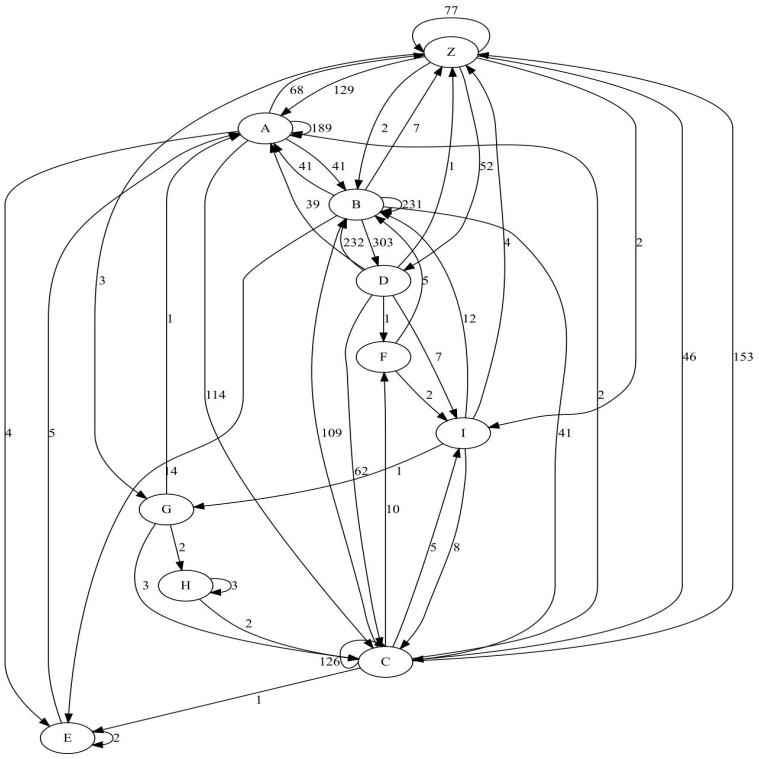



**FIGURE 3** Shortest Path Matrix of Directed Graph Corresponding to Walkinwat Malware App

$$
\begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \\ I \\ Z \end{array}
\begin{array}{c}
\begin{array}{cccccccccc} A & B & C & D & E & F & G & H & I & Z \end{array} \\
\begin{pmatrix}
0 & 1 & 1 & 2 & 1 & 1 & 1 & 3 & 2 & 1 \\
1 & 0 & 1 & 1 & 2 & 2 & 2 & 3 & 2 & 1 \\
1 & 1 & 1 & 0 & 2 & 1 & 2 & 3 & 1 & 1 \\
1 & 1 & 1 & 0 & 2 & 2 & 2 & 3 & 1 & 1 \\
1 & 2 & 2 & 3 & 0 & 3 & 3 & 4 & 3 & 2 \\
2 & 1 & 2 & 2 & 3 & 0 & 2 & 3 & 1 & 2 \\
1 & 2 & 1 & 3 & 2 & 2 & 0 & 1 & 2 & 2 \\
2 & 2 & 1 & 3 & 2 & 2 & 3 & 0 & 2 & 2 \\
2 & 1 & 1 & 2 & 2 & 2 & 1 & 2 & 0 & 1 \\
1 & 1 & 1 & 1 & 2 & 2 & 1 & 2 & 1 & 0
\end{pmatrix}
\end{array}
$$

## 3.2 | Extraction of Centrality Based Features from Syscall Digraph

In the second phase, we extract centrality values from syscall graph. Centrality is the measure of influence of a syscall $S_i$ for $i = 1, 2, \ldots, n$ in the syscall digraph $G$[42]. Here, we extract information related to syscalls using various centrality measures such as eigen, betweenness and closeness from the ordered syscall digraph $G$.

### 3.2.1 | Eigen Vector Centrality

Eigen vector centrality is the rate in which a syscall $S_i$ makes connection with syscalls $S_j$ for $j = 1, 2, \ldots, n$ and $i \neq j$ in the syscall digraph $G$. Let $B = (b_{i,j})_{n \times n}$ be a neighbourhood matrix corresponds to a graph $G$ in which each $b_{i,j}$ is calculated as:

$$b_{i,j} = \begin{cases} 1, & \text{if } a_{i,j} \neq 0; \\ 0, & \text{otherwise.} \end{cases}$$

where $a_{i,j}$ is the edge weight. The eigen vector centrality of a syscall $S_i$ is calculated as:

$$C_1(S_i) = \frac{1}{\lambda} \sum_{j=1}^{n} b_{i,j} a_{i,j}, \tag{1}$$

where $\lambda$ is the largest eigen value of the matrix $B$.

### 3.2.2 | Closeness Centrality

Closeness centrality is the rate in which a syscall is closer to other syscalls in the syscall digraph. Closeness centrality of a syscall $S_i$ is calculated as:

$$C_2'(S_i) = \frac{m-1}{\sum_{j=1}^{m} \sigma_{j,i}}, \tag{2}$$

where $\sigma_{i,j}$ is the length of a shortest path from $S_i$ to $S_j$ and $m$ is the number of syscalls that can reach $S_i$ through a path in the graph. Then, we normalize the centrality values with respect to the number of connected components. The normalized closeness centrality is computed as,

$$C_2(S_i) = \frac{m-1}{n-1} . C_{2(}'S_i). \tag{3}$$

The shortest path matrix of Walkinwat malware application is given in Figure 3.

### 3.2.3 | Betweenness Centrality

Betweenness centrality is the rate in which a syscall $S_i$ lies in the shortest path between other syscalls in the syscall digraph. Betweeness centrality of a syscall $S_i$ is calculated as:

$$C_3(S_i) = \sum_{j=1}^{n} \sum_{k=1}^{n} \frac{|\sigma_{j,k}(i)|}{|\sigma_{j,k}|}, \tag{4}$$

where $|\sigma_{j,k}(i)|$ is the number of a shortest paths from the syscall $S_j$ to other syscall $S_k$ passing through the syscall $S_i$ and $|\sigma_{j,k}|$ is the number of shortest paths from $S_j$ to $S_k$.

The centrality values of the syscall digraph of Walkinwat trojan is given in Table 4. From Table 4, we can see that the centrality values of certain syscalls such as $A$, $B$ and $C$ are higher than others. These syscalls are highly connected, have more control and more closer than other syscalls in the sequence.

## 3.3 | Malware Detection Phase

In malware detection phase, the centrality based features are given as inputs to a supervised machine learning classifier for identifying the malicious behavior. Here, any machine learning classifier such as ANN, random forest etc. can be used for malware detection.

# 4 | RESULTS

In this section, we discuss about the performance of our mechanism. In Section 4.1, we discuss about the benign and malware dataset. In Section 4.2, we discuss about the performance results in the datasets and compare it with other approaches.

**TABLE 4** Centrality Values of Syscalls in Walkinwat Digraph

| Syscall | Eigen centrality | Betweenness centrality | Closeness centrality |
|---|---|---|---|
| A | 0.4218 | 11.83 | 0.2700 |
| B | 0.4305 | 7.58 | 0.2700 |
| C | 0.4591 | 22.08 | 0.2945 |
| D | 0.1874 | 0.67 | 0.1705 |
| E | 0.1959 | 0 | 0.1906 |
| F | 0.1021 | 0 | 0.1800 |
| G | 0.2120 | 8.25 | 0.1906 |
| H | 0.0471 | 0 | 0.1409 |
| I | 0.3538 | 5.50 | 0.2314 |
| Z | 0.4120 | 10.08 | 0.2492 |

## 4.1 | Dataset

In this section, we discuss about the dataset. We built a balanced dataset consisting of 4000 malicious and benign applications for evaluating the performance of our mechanism. The malware applications were collected from Drebin[13] and AMD datasets[43]. The malware families are selected based on the diversity in:

- repacking;
- obfuscation techniques such as dynamic code loading, renaming, string encryption etc.;
- category such as Trojan, backdoor, Trojan SMS etc..

The statistics of diversity in selected malware families are given in Table 5. The number of system calls and length of system call sequence varies from application to application in a malware family. Hence, we have given the average length of system call sequence before and after preprocessing and average number of system calls in the sequence before and after preprocessing. These details are given in Table 6. We downloaded 2000 google play apps also. All benign apps were submitted to virustotal[44] for verifying their legitimacy.

We used an Android emulator to collect the syscall logs of all the malicious and benign applications in the dataset. The Android emulator was installed in an intel core i5 PC with 8GB memory. Here, we used strace tool to collect the syscall trace of applications which are installed in an Android emulator[45]. Monkey runner tool was used as an automated test case generation tool while tracing the syscalls[46]. The monkey runner tool can get its maximum code coverage within a minute[47]. After one minute, we terminated the execution of the applications and saved the syscall trace as log files. The collected syscall logs were preprocessed by removing the arguments and eliminating the irrelevant syscalls from them. From these preprocessed syscall logs, we built a csv (comma separated value) file corresponding to eigen centrality, closeness centrality and betweenness centrality based features of 2000 benign and 2000 malware applications.

## 4.2 | Performance Results

In this section, we discuss about the performance of our mechanism in the datasets described in Section 4.1. We used the features of 80% of malicious and benign apps for training the classifiers and the features of 20% of malicious and benign applications for testing the performance of the model. However, for measuring the performance in obfuscated malware apps, we used new ML models trained with the features of 80% malware (obfuscated and non obfuscated) and goodware (samples used in previous model) and used the features of 20% of malware (obfuscated) and goodware (samples used in previous models) for testing. For eliminating the multicollinearity problem, before testing, we have measured the variable inflation factor (VIF) values of the predictor variables (features) from the training dataset. Then, we removed the features those have a VIF value greater than 10 (variables that are highly correlated to other features in the model)[10]. The selected centrality based features of system calls with VIF values are given in Table 7. Also, we have not considered the centrality based features of system calls those values are zeros in all goodware and malware applications (Betweeness centrality of rename and sendmsg system calls and closeness centrality of rename system call). Here, we used the following ML classifiers for testing the performance:

**TABLE 5** Statistics of Diversity in Selected Malware Families

| Family | Number of Samples | Category | Repackaged | Dynamic Loading | Renaming | String Encryption | Native Payload |
|---|---|---|---|---|---|---|---|
| Adrd | 20 | Trojan | ✓ | | | | |
| Andrup | 45 | Adware | | | | | |
| Airpush | 92 | Adware | | | ✓ | | |
| BaseBridge | 20 | Backdoor | | | | | |
| Boqx | 38 | Trojan | | | | | |
| Youmi | 35 | Adware | | ✓ | | | |
| Utchi | 12 | Adware | | ✓ | | | |
| Winge | 16 | Trojan | | | | | |
| Stealer | 8 | Trojan | | | | | |
| Ramnit | 4 | Trojan | ✓ | | | | |
| Kyview | 28 | Adware | | | ✓ | ✓ | |
| Kuaguo | 25 | Adware | | ✓ | ✓ | ✓ | |
| Fjcon | 3 | Trojan | | | | | |
| Erop | 1 | Trojan SMS | | | | | |
| Fusob | 1 | Ransomware | | ✓ | ✓ | ✓ | |
| DrodKungFu | 326 | Backdoor | ✓ | | | ✓ | ✓ |
| GoldDream | 37 | Backdoor | ✓ | | | | |
| AndroRAT | 45 | Backdoor | | | | | |
| Boxer | 44 | Trojan SMS | | | ✓ | ✓ | |
| MobileTX | 17 | Trojan | | | | | |
| Rumms | 50 | Trojan SMS | | ✓ | | ✓ | |
| Lottor | 100 | Exploit | | ✓ | ✓ | ✓ | ✓ |
| Mseg | 131 | Trojan | ✓ | | ✓ | | |
| Penetho | 18 | Exploit | | | | | |
| Vidro | 23 | Trojan SMS | | | | | |
| Mercor | 502 | Trojan Spy | | | | | |
| MMarketPay | 9 | Trojan | ✓ | | | | |
| Lnk | 5 | Trojan | ✓ | | | | |
| FakeDoc | 21 | Trojan | | | ✓ | | |
| FakeAV | 5 | Trojan | | | | | |
| FakeInstall | 15 | Trojan SMS | | | ✓ | | |
| FakePlayer | 16 | Trojan SMS | | | ✓ | | |
| FakeTimer | 10 | Trojan | | | | | |
| FakeAngry | 10 | Trojan | | | | | |
| Zitmo | 19 | Trojan Banker | | | ✓ | | |
| Tesbo | 5 | Trojan SMS | ✓ | | ✓ | ✓ | |
| Minimob | 10 | Adware | | | ✓ | | |
| Mtk | 58 | Trojan | ✓ | ✓ | ✓ | | |
| NandroBox | 73 | Trojan | ✓ | | | | |
| Others | 100 | Trojan | | | | | |

- Ridge Regularized Logistic Regression Classifier (RRLR);
- Lasso Regularized Logistic Regression Classifier (LRLR);

**TABLE 6** Preprocessing Details of Selected Malware Families

| Family | Average Length of system calls before preprocessing phase | Average Number of system calls before preprocessing phase) | Average Length of system calls after preprocessing phase | Average Number of system calls after preprocessing phase |
|---|---|---|---|---|
| Adrd | 16366 | 36 | 8198 | 13 |
| Andrup | 5409 | 30 | 2076 | 12 |
| Airpush | 6303 | 30 | 2189 | 13 |
| Boqx | 15579 | 32 | 5277 | 13 |
| BaseBridge | 1002 | 35 | 3372 | 14 |
| Youmi | 9883 | 29 | 3999 | 12 |
| Utchi | 12685 | 32 | 3815 | 12 |
| Winge | 8913 | 32 | 3006 | 12 |
| Stealer | 16445 | 28 | 3398 | 10 |
| Ramnit | 7981 | 31 | 1990 | 10 |
| Kyview | 15213 | 33 | 5989 | 13 |
| Kuaguo | 7731 | 33 | 2687 | 13 |
| Fjcon | 18254 | 27 | 6414 | 8 |
| Erop | 3929 | 40 | 454 | 8 |
| Fusob | 9494 | 18 | 1620 | 3 |
| DrodKungFu | 6505 | 34 | 2350 | 14 |
| GoldDream | 13923 | 29 | 6428 | 12 |
| AndroRAT | 4404 | 35 | 1524 | 13 |
| Boxer | 9535 | 31 | 2956 | 11 |
| MobileTX | 8453 | 35 | 2133 | 13 |
| Rumms | 1517 | 28 | 745 | 9 |
| Lottor | 3031 | 28 | 1051 | 12 |
| Mseg | 7300 | 31 | 2358 | 13 |
| Penetho | 8158 | 33 | 3035 | 12 |
| Vidro | 3525 | 36 | 580 | 8 |
| Mercor | 4617 | 40 | 765 | 17 |
| MMarketPay | 6902 | 35 | 2187 | 14 |
| Lnk | 6838 | 33 | 1300 | 9 |
| FakeDoc | 7125 | 36 | 796 | 10 |
| FakeAV | 9902 | 38 | 1855 | 8 |
| FakeInstall | 2900 | 29 | 1230 | 12 |
| FakeAngry | 20196 | 39 | 4873 | 11 |
| FakePlayer | 2581 | 37 | 356 | 7 |
| FakeTimer | 124 | 15 | 37 | 4 |
| Zitmo | 3257 | 27 | 1197 | 9 |
| Tesbo | 14710 | 27 | 6331 | 10 |
| Minimob | 14191 | 31 | 4526 | 13 |
| Mtk | 4034 | 25 | 1494 | 10 |
| NandroBox | 2994 | 23 | 1249 | 10 |
| Others | 4148 | 31 | 1578 | 13 |

- Artificial Neural Network (ANN);
- Decision Trees;
- Random Forest.

The performance results with the centrality measures (betweennes, closeness and eigen) in various ML algorithms are given in Table 8. From Table 8, we can see that the accuracy and F1score is 0.99 while random forest is used as the ML classifier. Also, the proposed mechanism can detect the obfuscated malware apps with an accuracy and F1score of 0.98 in random forest classifier. From this, it is clear that the proposed centrality based features are accurate in detecting malware apps which employ code obfuscation techniques to bypass static analysis techniques. The overall time for evaluating our mechanism is given in Table 9. From the Table 9, we can see that the proposed mechanism require less than 50 seconds for evaluation. However, certain system call based malware detection mechanisms ( Bhandari et al.[39]) require more than 30 minutes for evaluating the mechanism. We compare our mechanism with the popular system call based dynamic analysis mechanisms. The source codes

**TABLE 7** VIF Values of Centrality Based Features of System Calls

| System Call | Eigen Centrality | Betweenness Centrality | Closeness Centrality |
|---|---|---|---|
| recvfrom | 4.03 | 2.32 | **16.86** |
| write | 2.24 | 5.30 | **26.80** |
| ioctl | 2.29 | 8.67 | **30.18** |
| read | 1.86 | 5.40 | **21.42** |
| sendto | 2.27 | 6.32 | 8.59 |
| dup | 1.78 | 6.47 | **11.13** |
| writev | 4.28 | 3.17 | 8.20 |
| pread | 1.76 | 1.78 | 6.24 |
| close | 2.38 | **12.42** | **13.27** |
| socket | 2.57 | 4.15 | 2.64 |
| bind | 3.00 | 5.33 | 7.16 |
| connect | 2.79 | 4.45 | **17.30** |
| mkdir | 2.84 | 2.08 | 5.83 |
| access | 3.56 | 4.97 | 8.52 |
| chmod | 2.67 | 5.94 | 7.17 |
| open | 3.16 | 4.71 | 7.72 |
| rename | 2.24 | - | - |
| fchown32 | 2.44 | 2.35 | 5.21 |
| unlink | 2.44 | 3.87 | 6.81 |
| pwrite | 3.06 | 7.12 | **20.07** |
| unmask | 3.69 | 2.91 | 6.71 |
| lseek | 3.58 | 5.98 | **15.18** |
| fcntl | 2.62 | 3.73 | 5.61 |
| recvmsg | 2.86 | **10.78** | **17.52** |
| sendmsg | 2.53 | - | 1.10 |
| epoll | 2.72 | 2.03 | **10.38** |

**TABLE 8** Performance of Our Approach in Different ML Classifiers

| Algorithm | Apps | True Positive Rate | False Positive Rate | Positive Predictive Value | Accuracy | F1Score |
|---|---|---|---|---|---|---|
| ANN | overall malware apps | 0.97 | 0.05 | 0.95 | 0.96 | 0.96 |
| LRLR | overall malware apps | 0.97 | 0.07 | 0.94 | 0.95 | 0.95 |
| RRLR | overall malware apps | 0.97 | 0.07 | 0.94 | 0.95 | 0.95 |
| Decision Tree | overall malware apps | 0.97 | 0.06 | 0.94 | 0.95 | 0.95 |
| Random Forest | overall malware apps | 0.99 | 0.02 | 0.98 | 0.99 | 0.99 |
| Random Forest | Obfuscated malware Apps | 0.98 | 0.02 | 0.98 | 0.98 | 0.98 |

**TABLE 9** Processing time of our approach (in seconds)

| Feature Extraction Time (4000 Apps) | Training Time (Random Forest Classifier) | Testing Time (Random Forest Classifier) |
|---|---|---|
| 41 seconds | 5.6 seconds | 0.06 seconds |

of these existing mechanisms are not released in public. Hence, it is not possible to reimplement their mechanisms in our dataset. Most of these mechanisms were tested in the various subsets of Drebin and Malgenome datasets. The details of the subsets are not explicitly mentioned in these papers. So, we cannot implement our mechanism in their actual sample subset for comparison. Hence, in our work, we used a more extensive and diverse dataset which contains Drebin/Malgenome samples for

comparison. The comparative results are given in Table 10. From Table 10, we can see that our mechanism outperforms the existing syscall based malware detection mechanisms with an accuracy and F1Score of 0.99.

**TABLE 10** Performance Comparison with Existing Well Known System Call Based Feature Representation Mechanisms

| Mechanism | Feature Vector | Dataset Size | True Positive Rate | False Positive Rate | Positive Predictive Value | Accuracy | F1Score |
|---|---|---|---|---|---|---|---|
| Our mechanism | Graph centrality based features | 4000 | 0.99 | 0.02 | 0.99 | 0.99 | 0.99 |
| Zhang et al.[48] | Transition Probability Matrix | 2407 | 0.97 | N/A | N/A | 0.98 | N/A |
| Xiao et al.[7] | Transition Probability Matrix | 2407 | 0.98 | N/A | 0.98 | N/A | 0.98 |
| Bernadi et al.[32] | System call Execution Fingerprint | 1200 | 0.94 | N/A | 0.90 | 0.91 | N/A |
| Bhandari et al.[39] | Algebraic Logic Branching Factor (ALBF) | 2000 | 0.96 | 0.07 | N/A | 0.94 | N/A |
| Canfora et al.[49] | System call subsequence count | 2000 | 0.97 | N/A | N/A | 0.97 | N/A |
| Roopak et al.[34] | System call graph signals | 2500 | 0.96 | 0.02 | 0.98 | 0.97 | 0.97 |
| Tong et al.[8] | System call subsequence analysis | 2607 | N/A | N/A | N/A | 0.91 | N/A |

## 5 | CONCLUSION

In this paper, we proposed a mechanism to detect Android malware applications from the centrality based features of syscall graphs. We found that our mechanism is very accurate in detecting malware apps which employ various obfuscation techniques. Further, our mechanism outperforms the existing syscall based dynamic detection mechanisms and ML based detection mechanisms which use various syscall based features such as system call frequency, transition probability matrix etc. with an accuracy of 0.99.

In our mechanism, some malware apps in our dataset did not exhibited malicious behavior while collecting the syscall traces. This is because of the limited code coverage problem in automated test case generation tools such as monkeyrunner. In order to overcome this limitation, in future, we will use multiple test case generation tools having different exploration strategy[50]. Further, some goodware apps were falsely detected as malware. This is because certain goodware apps try to request more privileges during their execution time. In future, we will consider syscall argument relationship information in syscall graph models to reduce the false positive rates.

## References

1. Meshram P, Thool R. A survey paper on vulnerabilities in android OS and security of android devices. In: *Wireless Computing and Networking (GCWCN), 2014 IEEE Global Conference on*, IEEE. ; 2014: 174–178.

2. Felt AP, Finifter M, Chin E, Hanna S, Wagner D. A survey of mobile malware in the wild. In: *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, ACM. ; 2011: 3–14.

3. Schmidt AD, Bye R, Schmidt HG, Clausen J, Kiraz O, Yuksel KA, Camtepe SA, Albayrak S. Static analysis of executables for collaborative malware detection on android. In: *2009 IEEE International Conference on Communications*, IEEE. ; 2009: 1–5.

4. Rastogi V, Chen Y, Jiang X. Catch me if you can: Evaluating android anti-malware against transformation attacks. *Information Forensics and Security, IEEE Transactions on* 2014; 9(1): 99–108.

5. Xie L, Zhang X, Seifert JP, Zhu S. pBMDS: a behavior-based malware detection system for cellphone devices. In: *Proceedings of the third ACM conference on Wireless network security*, ACM. ; 2010: 37–48.

6. Dimjašević M, Atzeni S, Ugrina I, Rakamaric Z. Evaluation of Android Malware Detection Based on System Calls. In: *Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics*ACM; 2016; New York, NY, USA: 1–8

7. Xiao X, Wang Z, Li Q, Xia S, Jiang Y. Back-propagation neural network on Markov chains from system call sequences: a new approach for detecting Android malware with system call sequences. *IET Information Security, Volume:11, Issue:1* 2017.

8. Tong F, Yan Z. A hybrid approach of mobile malware detection in Android. *Journal of Parallel and Distributed computing* 2017; 103: 22–31.

9. Alin A. Multicollinearity. *Wiley Interdisciplinary Reviews: Computational Statistics* 2010; 2(3): 370–374.

10. Mason RL, Gunst RF, Hess JL. *Statistical design and analysis of experiments: with applications to engineering and science*. 474. John Wiley & Sons . 2003.

11. Aafer Y, Du W, Yin H. DroidAPIMiner: Mining API-level features for robust malware detection in android. In: *Security and Privacy in Communication Networks*Springer. 2013 (pp. 86–103).

12. Yang C, Xu Z, Gu G, Yegneswaran V, Porras P. Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications. In: *European symposium on research in computer security*, Springer. ; 2014: 163–182.

13. Arp D, Spreitzenbarth M, Hubner M, Gascon H, Rieck K. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket.. In: *NDSS*; 2014.

14. Suarez-Tangil G, Tapiador JE, Peris-Lopez P, Blasco J. Dendroid: A text mining approach to analyzing and classifying code structures in android malware families. *Expert Systems with Applications* 2014; 41(4): 1104–1117.

15. Fan M, Liu J, Luo X, Chen K, Tian Z, Zheng Q, Liu T. Android malware familial classification and representative sample selection via frequent subgraph analysis. *IEEE Transactions on Information Forensics and Security* 2018; 13(8): 1890–1905.

16. Shahriar H, Islam M, Clincy V. Android malware detection using permission analysis. In: *SoutheastCon, 2017*, IEEE. ; 2017: 1–6.

17. Grace M, Zhou Y, Zhang Q, Zou S, Jiang X. Riskranker: scalable and accurate zero-day android malware detection. In: *Proceedings of the 10th international conference on Mobile systems, applications, and services*, ACM. ; 2012: 281–294.

18. Avdiienko V, Kuznetsov K, Gorla A, Zeller A, Arzt S, Rasthofer S, Bodden E. Mining apps for abnormal usage of sensitive data. In: *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, IEEE Press. ; 2015: 426–436.

19. Xu K, Li Y, Deng RH. ICCDetector: ICC-based malware detection on Android. *IEEE Transactions on Information Forensics and Security* 2016; 11(6): 1252–1264.

20. Feizollah A, Anuar NB, Salleh R, Suarez-Tangil G, Furnell S. Androdialysis: Analysis of android intent effectiveness in malware detection. *computers & security* 2017; 65: 121–134.

21. Zhu HJ, You ZH, Zhu ZX, Shi WL, Chen X, Cheng L. DroidDet: effective and robust detection of android malware using static analysis along with rotation forest model. *Neurocomputing* 2018; 272: 638–646.

22. Hou S, Saas A, Ye Y, Chen L. Droiddelver: An android malware detection system using deep belief network based on api call blocks. In: *International conference on web-age information management*Springer. ; 2016: 54–66.

23. McLaughlin N, Rincon M. dJ, Kang B, Yerima S, Miller P, Sezer S, Safaei Y, Trickel E, Zhao Z, Doupé A, others . Deep android malware detection. In: *Proceedings of the seventh ACM on conference on data and application security and privacy*; 2017: 301–308.

24. Cai M, Jiang Y, Gao C, Li H, Yuan W. Learning features from enhanced function call graphs for Android malware detection. *Neurocomputing* 2021; 423: 301–307.

25. Jamrozik K, Zeller A. Droidmate: a robust and extensible test generator for android. In: *Proceedings of the International Conference on Mobile Software Engineering and Systems*, ACM. ; 2016: 293–294.

26. Zaman M, Siddiqui T, Amin MR, Hossain MS. Malware detection in Android by network traffic analysis. In: *2015 international conference on networking systems and security (NSysS)*, IEEE. ; 2015: 1–5.

27. Shabtai A, Kanonov U, Elovici Y, Glezer C, Weiss Y. Andromaly: a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems* 2012; 38(1): 161–190.

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

29. Xiao X, Xiao X, Jiang Y, Liu X, Ye R. Identifying Android malware with system call co-occurrence matrices. *Transactions on Emerging Telecommunications Technologies* 2016; 27(5): 675–684.

30. Wei Y, Zhang H, Ge L, Hardy R. On behavior-based detection of malware on android platform. In: *Global Communications Conference (GLOBECOM), 2013 IEEE*, IEEE. ; 2013: 814–819.

31. Xiao X, Zhang S, Mercaldo F, Hu G, Sangaiah AK. Android malware detection based on system call sequences and LSTM. *Multimedia Tools and Applications* 2017: 1–21.

32. Bernardi M L, Cimitile M, Distante D, Martinelli F, Mercaldo F. Dynamic malware detection and phylogeny analysis using process mining. *International Journal of Information Security* 2019; 18(3): 257–284.

33. Vinod P, Zemmari A, Conti M. A machine learning based approach to detect malicious android apps using discriminant system calls. *Future Generation Computer Systems* 2019; 94: 333–350.

34. Surendran R, Thomas T, Emmanuel S. GSDroid: Graph Signal Based Compact Feature Representation for Android Malware Detection. *Expert Systems with Applications* 2020: 113581.

35. De Lorenzo A, Martinelli F, Medvet E, Mercaldo F, Santone A. Visualizing the outcome of dynamic analysis of Android malware with VizMal. *Journal of Information Security and Applications* 2020; 50: 102423.

36. John Teenu S, Thomas T, Emmanuel S. Graph Convolutional Networks for Android Malware Detection with System Call Graphs. In: *2020 Third ISEA Conference on Security and Privacy (ISEA-ISAP)*, IEEE. : 162–170.

37. Shastry A, Kantarcioglu M, Zhou Y, Thuraisingham B. Randomizing smartphone malware profiles against statistical mining techniques. In: *Data and Applications Security and Privacy XXVI*Springer. 2012 (pp. 239–254).

38. Hou S, Saas A, Chen L, Ye Y. Deep4maldroid: A deep learning framework for android malware detection based on linux kernel system call graphs. In: *2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW)*, IEEE. ; 2016: 104–111.

39. Bhandari S, Panihar R, Naval S, Laxmi V, Zemmari A, Gaur MS. SWORD: semantic aware Android malware detector. *Journal of information security and applications* 2018; 42: 46–56.

40. Wang X, Jhi YC, Zhu S, Liu P. Detecting software theft via system call based birthmarks. In: *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, IEEE. ; 2009: 149–158.

41. Yi Y, Lingyun Y, Rui W, Purui S, Dengguo F. DepSim: a dependency-based malware similarity comparison system. In: *International Conference on Information Security and Cryptology*, Springer. ; 2010: 503–522.

42. Nieminen J. On the centrality in a graph. *Scandinavian journal of psychology* 1974; 15(1): 332–336.

43. Wei F, Li Y, Roy S, Ou X, Zhou W. Deep ground truth analysis of current android malware. In: *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, Springer. ; 2017: 252–276.

44. VirusTotal . Virustotal-free online virus, malware and url scanner. *Online: https://www. virustotal. com/en* 2012; 2.

45. sourceforge.net . Strace Linux system call tracer. [Online] Available: https://sourceforge.net/projects/strace/; . Accessed:11-07-2016.

46. Mooney P, Corcoran P. Using OSM for LBS–an analysis of changes to attributes of spatial objects. In: *Advances in Location-Based Services*Springer. 2012 (pp. 165–179).

47. Bao L, Le TDB, Lo D. Mining sandboxes: Are we there yet?. In: *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE. ; 2018: 445–455.

48. Zhang S, Xiao X. CSCdroid: Accurately Detect Android Malware via Contribution-Level-Based System Call Categorization. In: *Trustcom/BigDataSE/ICESS, 2017 IEEE*, IEEE. ; 2017: 193–200.

49. Canfora G, Medvet E, Mercaldo F, Visaggio CA. Detecting android malware using sequences of system calls. In: *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, ACM. ; 2015: 13–20.

50. Choudhary Shauvik R, Gorla A, Orso A. Automated test input generation for android: Are we there yet?(e). In: *2015 30th IEEE/ACM International Conference on Automated Software Engineering*, IEEE. ; 2015: 429–440.