Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.046890

ARTICLE

Covalent Bond Based Android Malware Detection Using Permission and System Call Pairs

Rahul Gupta[1], Kapil Sharma[1,*] and R. K. Garg[2]

[1]Department of Information Technology, Delhi Technological University, New Delhi, 110042, India

[2]Department of Mechanical Engineering, Deenbandhu Chhotu Ram University of Science and Technology, Murthal, 131039, India

*Corresponding Author: Kapil Sharma. Email: kapil@ieee.org

Received: 18 October 2023 Accepted: 14 February 2024 Published: 26 March 2024

ABSTRACT

The prevalence of smartphones is deeply embedded in modern society, impacting various aspects of our lives. Their versatility and functionalities have fundamentally changed how we communicate, work, seek entertainment, and access information. Among the many smartphones available, those operating on the Android platform dominate, being the most widely used type. This widespread adoption of the Android OS has significantly contributed to increased malware attacks targeting the Android ecosystem in recent years. Therefore, there is an urgent need to develop new methods for detecting Android malware. The literature contains numerous works related to Android malware detection. As far as our understanding extends, we are the first ones to identify dangerous combinations of permissions and system calls to uncover malicious behavior in Android applications. We introduce a novel methodology that pairs permissions and system calls to distinguish between benign and malicious samples. This approach combines the advantages of static and dynamic analysis, offering a more comprehensive understanding of an application's behavior. We establish covalent bonds between permissions and system calls to assess their combined impact. We introduce a novel technique to determine these pairs' Covalent Bond Strength Score. Each pair is assigned two scores, one for malicious behavior and another for benign behavior. These scores serve as the basis for classifying applications as benign or malicious. By correlating permissions with system calls, the study enables a detailed examination of how an app utilizes its requested permissions, aiding in differentiating legitimate and potentially harmful actions. This comprehensive analysis provides a robust framework for Android malware detection, marking a significant contribution to the field. The results of our experiments demonstrate a remarkable overall accuracy of 97.5%, surpassing various state-of-the-art detection techniques proposed in the current literature.

KEYWORDS

Android; malware; android security; hybrid analysis; permission and system call pairs

1 Introduction

The Android operating system has maintained a dominant position in the smartphone industry for the past decade. Within the Android API framework, functions grant access to sensitive system resources. Unfortunately, this feature has allowed cyber attackers to develop and disseminate harmful

applications through alternative app stores or social media advertisements. Furthermore, an attacker may introduce malicious components in the installed Android application. These malevolent applications empower attackers to perform various operations, including information theft, SMS transmission, and remote device control. Consequently, safeguarding smartphones from these malicious applications is imperative [1–3].

Malware detection methods currently fall into three primary categories: Static, dynamic, and hybrid analysis. Static analysis is capable of discerning malicious behavior by examining an application's source code without executing it [4]. On the other hand, dynamic analysis identifies malicious behavior by analyzing the runtime information generated during the application's execution, such as system calls [5]. The strength of static analysis lies in its ability to pinpoint malicious components directly from the source code, resulting in high code coverage [6]. Dynamic analysis excels in uncovering exploits within the runtime environment [7]. Therefore, by merging the strengths of static and dynamic analysis, a hybrid analysis approach can be formulated to enhance malware detection accuracy [8,9].

Several static works have been proposed in the literature for Android malware detection. For instance, in [10], Talha et al. extracted application permissions. They then assign a score to each permission, determined by the ratio of malware instances containing that specific permission to the total number of malware instances. In [11], the study utilized pairs of permissions extracted from the manifest file, resulting in an overall accuracy of 95.44%. IPDroid, as discussed in [12], incorporated both permissions and intents from the manifest file in their analysis. They achieved a notable accuracy of 94.73% by employing a Random Forest classifier.

The TaintDroid model [13] employed dynamic taint analysis to monitor the movement of privacy-sensitive data within third-party applications. Yang et al. [14] expanded upon the TaintDroid model to not only identify data leaks from applications but also ascertain whether these leaks are a result of user intention or not. In [15], the authors introduced a proficient and automated approach for detecting malware by leveraging the textual semantics of network traffic. Specifically, they treated each HTTP flow produced by mobile applications as a textual document, allowing them to apply natural language processing techniques to extract features at the text level.

Some of the works have combined static and dynamic features to propose a hybrid Android malware detector. MADAM [16] is a host-based malware detection system designed for Android devices. It conducts concurrent analysis and correlation of attributes across four tiers: Kernel, application, user, and package. This comprehensive approach aims to identify and thwart malicious activities effectively. Monet [17] consists of a module on the user side, an application responsible for analyzing malicious activity and signatures. Conversely, the module installed on the server side is responsible for detecting malicious applications based on analysis on the client side. In [18], authors developed AppAudit which employs a combination of static and dynamic analysis to deliver highly effective real-time app auditing. It introduces an innovative dynamic analysis approach that leverages this combination to reduce false positives generated by an efficient yet conservative static analysis.

1.1 Motivation

Identifying dangerous combinations of permissions and system calls is instrumental in spotting malicious behavior. Hence, this study endeavors to scrutinize permissions and system calls in pairs and introduces a novel methodology to identify such pairs that can differentiate between benign and malicious samples. To the best of our knowledge, we are the first to use permissions and system call pairs to detect Android malware. Pairing permissions and system calls has several key benefits. Firstly, permissions are static features, and system calls are dynamic features; pairing both of

them will combine the advantages of static analysis and dynamic analysis to form a hybrid analysis technique. Second, this combination allows for a more detailed examination of an application's behavior. Permissions provide a high-level overview of what resources an app may access, while system calls offer a finer-grained view of actual interactions with the system. By correlating permissions with system calls, we can better understand how an application uses the permissions it requests. This context is crucial in distinguishing legitimate behavior from potentially malicious actions. It enables the detection of anomalies or suspicious activities. For example, if an app with camera access permission unexpectedly starts making network-related system calls, it may raise a red flag. The app requests access to the camera (Android.permission.CAMERA). Additionally, it asks permission to access the internet (Android.permission.INTERNET). Based on permissions alone, the app seems legitimate. Camera apps naturally require camera access and internet access could be justified for features like cloud storage of images. During runtime, if the app makes system calls such as open(), read(), write(), and connect(). This observation may establish suspicious behavior as the app is accessing files unrelated to image storage and making network connections to unusual domains. Hence, this study endeavors to scrutinize permissions and system calls in pairs and introduces a novel methodology to identify such pairs that can differentiate between benign and malicious samples.

1.2 Contributions

We present a covalent bond-based Android malware detection model using permissions and system call pair. We use the analogy of covalent bonds between two atoms in chemistry to form covalent bonds between every permission and system call. We also calculate bond strengths between permission and system call pairs to denote the strength of the bond they create between them. The estimated bond strength helps detect an Android application as malicious or benign. Our detection results demonstrate an overall accuracy of 97.5%, better than many state-of-the-art detection techniques proposed in the literature. The main contributions of the paper are summarized below:

- We build the permission and system call covalent bond pairs to identify and analyze the impact of these pairs.
- We proposed a novel approach to calculate the Covalent bond strength score for the permissions and system calls bond pair. Two scores, i.e., malicious and benign, are computed for each bond pair.
- We designed a technique for identifying Android applications as malicious or benign based on the malicious and benign scores of permission and system call pairs.
- We conducted a comparative analysis between our proposed model and other state-of-the-art detection techniques. Our findings demonstrate that the proposed model surpasses similar state-of-the-art models in terms of performance.

1.3 Organization

The subsequent sections of this paper are structured as follows: In Section 2, we delve into the related work. Section 3 provides an in-depth exploration of our methodology. Section 4 is devoted to presenting results and engaging in discussions. Finally, in Section 5, we conclude and outline potential future directions for this research.

2 Related Work

This section presents a literature review on detecting Android malware using machine-learning methods. Android malware analysis methods enable gathering various feature types, which are

essential for characterizing and constructing machine-learning systems. Three primary approaches are employed, depending on the type of features gathered: Static analysis, dynamic analysis, or a combination of both, known as hybrid analysis [19]. This section offers a concise overview of these approaches and the typical features of machine learning-based Android malware detection.

2.1 Static Analysis

Shahriar et al. [20] introduced a method to identify Android malware by examining requested permissions. Their initial approach involved utilizing Latent Semantic Indexing (LSI) to pinpoint frequently requested permissions within known instances of malicious applications. In a separate development, Arp et al. [21] introduced Drebin, a method for static malware detection. This technique relies on unchanging attributes from manifest file components such as permissions, hardware and application components, and intent filters. Cen et al. [22] proposed a strategy to identify malicious Android applications by scrutinizing permissions and API calls. They employ a trained probabilistic classifier to predict an application's potential maliciousness. Qui et al. [23] proposed Android malware detection model based on reverse engineering to detect zero day malware families. Ibrahim et al. [24] proposed an approach for predicting malicious applications through API deep learning model based on two new features, i.e., application size and fuzzy hash.

2.2 Dynamic Analysis

Yu et al. [25] presented a method for classifying an Android application as malicious by system call analysis with ML techniques such as SVM or naive Bayes learners. Dmjsevac et al. introduced Maline [26], a framework to detect Android malware applications. Maline utilizes binary machine learning classifiers to deduce an application's malicious behavior by examining the frequencies of individual system calls and their interdependencies. Crowdroid [27] adopts a behavior-centric approach for Android malware detection, utilizing a cloud-based infrastructure. The K-means clustering technique on the server side processes the data gathered regarding system call events on the client side to identify malicious Android applications.

In [28], authors presented an Android malware detection model for detecting malware applications based on traces of their behavioral system calls. Lu et al. [29] introduced a new encoding scheme F2D, which uses raw payload of network traffic along with neural networks to propose an Android malware detection model. Hussian et al. [30] proposed a malware detection technique using particle swam optimization, which selects traffic characteristics from network traffic data which in turn is fed to ML algorithms to build the prediction models.

2.3 Hybrid Analysis

Arora et al. [31] introduced a methodology for Android malware prediction that relies on permissions and internet traffic features. This approach leverages the FP-growth algorithm, using permissions and network traffic features to discern potentially malicious behavior. In [32] authors proposed a hybrid detector technique for identifying malicious Android applications. This method combines a sequence of API calls represented as a Markov chain from static and dynamic analyses to detect malicious behaviour effectively. AASandbox [33] employs a hybrid analyzer for the detection of malware. In their static model, they uncovered patterns that help identify malicious applications. Further, they captured system calls for dynamic analysis in an emulator. The authors in [34] used permissions, API calls, and intents as features to propose a hybrid Android malware detector.

3 The Proposed Covalent Bond Pair Detection

In this section, we present our novel Covalent Bond Pair-based model for detecting malicious Android applications. The proposed model is depicted in Fig. 1.

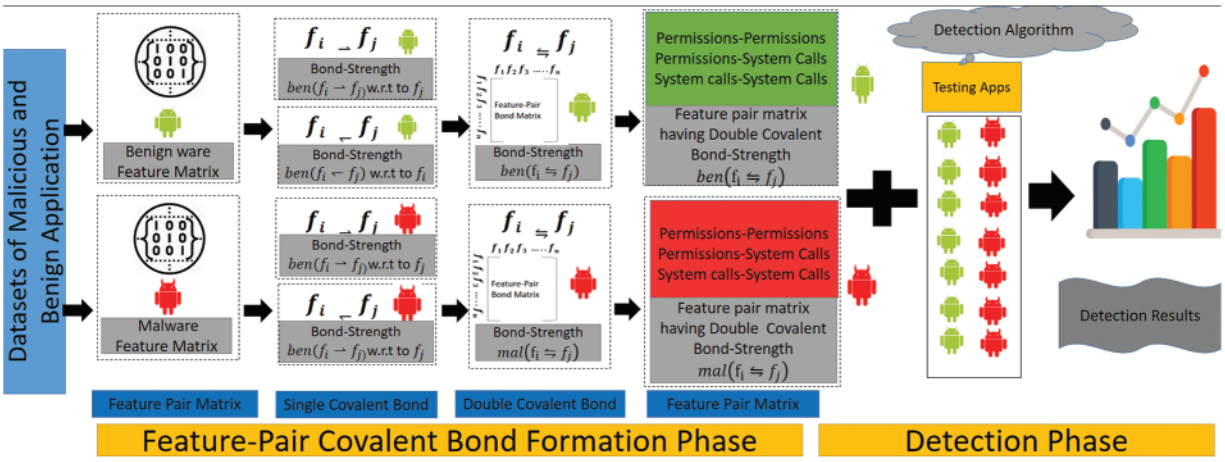

Figure 1: Proposed covalent bond pair detection model

3.1 Data Set Description

KronoDroid [35], a meticulously structured Android dataset, holds the distinction of being the largest in its category. It is distinguished by its amalgamation of static and dynamic features and the notable inclusion of timestamps. This dataset meticulously accounts for the unique characteristics of dynamic data sources, encompassing samples from over 209 distinct Android malware families. Its creation involved the fusion of diverse sources of benign and malware data, resulting in a comprehensive collection spanning a significant period. The dataset comprises 41,382 instances of malware belonging to 240 distinct malware families, along with 36,755 benign applications.

The dataset predominantly comprises permissions as static features, represented as binary indicators of whether the app requested the standard Android permissions (1) or not (0). There are a total of 166 distinct permissions in the dataset. In contrast, the dynamic feature set mainly consists of system calls, represented by the absolute frequency of each system call issued by the app at runtime. The system call set comprises 288 features. Hence, the total number of features under consideration amounts to 454.

3.2 Feature Space Transformation

As previously stated, the KronoDroid dataset is well-organized and accessible in CSV file format. These files contain information on both malware and benign applications. The feature vectors within these CSV files are represented as combinations of 0's and 1's. A 0 in the feature vector signifies the absence of a particular feature in an application, while a 1 indicates its presence. Tables 1 and 2 provide a visual representation of the feature spaces for benign and malicious applications, respectively.

Within both the instances of benign and malicious CSV files as represented in Tables 1 and 2, respectively, the labels P_1, P_2, P_3,..., and P_n represent the n permissions, while S_1, S_2, S_3,..., and S_m denote the m system calls. In our specific dataset, n is set at 166 and m at 288. The benign applications are denoted as A_{1B}, A_{2B},..., and A_{xB}, where x represents the total number of benign applications. Similarly,

the malicious applications are labeled A_{1M}, A_{2M},..., and A_{yM}, with y indicating the total number of malicious applications.

Table 1: Instance of benign CSV

Benign CSV	P_1	P_2	P_3	.	.	.	.	.	P_n	S_1	S_2	S_3	.	.	.	.	.	S_m
A_{1B}	1	0	1	1	0	0	1	1	1	0	0	1	1	1	1	1	0	1
A_{2B}	1	0	1	1	0	0	1	0	1	0	0	1	0	1	0	1	0	1
.	0	0	1	1	0	0	1	1	0	0	0	0	1	1	1	0	0	1
.	0	0	1	1	0	0	1	1	0	1	0	0	1	1	1	0	0	0
A_{xB}	1	0	1	0	0	0	1	1	0	0	0	0	1	0	1	0	0	0

Table 2: Instance of malicious CSV

Malicious CSV	P_1	P_2	P_3	.	.	.	.	.	P_n	S_1	S_2	S_3	.	.	.	.	.	S_m
A_{1M}	0	0	1	1	0	1	1	1	1	0	0	0	1	1	0	1	0	1
A_{2M}	1	1	1	0	0	0	1	1	1	0	1	1	0	0	0	1	0	0
.	0	0	1	1	0	0	1	1	0	0	0	0	1	1	1	0	0	1
.	0	1	1	1	0	0	1	1	0	0	0	0	1	1	0	1	1	1
A_{yM}	1	1	1	0	0	0	1	1	0	0	1	1	1	0	1	0	1	0

3.3 Covalent Bond Pair Formation Phase

The concept of feature pair covalent bond formation is based on the concepts of the covalent bond theory of chemistry [36]. A covalent bond arises from the mutual sharing of electrons between the involved atoms. This pair of electrons engaged in this form of bonding is referred to as a shared pair or bonding pair. Additionally known as molecular bonds, covalent bonds facilitate the attainment of outer shell stability, resembling the configuration of noble gases, by enabling the sharing of these bonding pairs. Covalent bonds are normally categorized into three types: Single covalent bonds, double covalent bonds, and triple covalent bonds. We will restrict our proposed methodology to single covalent bonds and double covalent bonds only.

A single bond is established through the sharing of only one pair of electrons between the two involved atoms, symbolized by a single dash (-). Despite having lower density and strength than double and triple bonds, this type of covalent bond is the most stable.

A double bond is created when two pairs of electrons are shared between the participating atoms, denoted by two dashes (=). Double covalent bonds exhibit significantly greater strength than single bonds, although comparatively less stable.

In the case of our proposed methodology, we calculated single covalent bond strengths and double covalent bond strengths between two arbitrary features f_i and f_j, and formed feature pair f_{ij}. We separately calculated these bond strengths from two perspectives: w.r.t benign applications and w.r.t malicious applications. Hence, the concept of covalent bond strengths helps to calculate benign and malicious feature pair scores between every possible feature pair in the dataset. This notion of covalent bond strengths gives us a perspective of separately viewing any arbitrary feature pair regarding the role played for benign and malicious applications. Algorithm 1 depicts the whole phase of Feature Pair Covalent Bond Formation.

Algorithm 1: Feature pair covalent bond formation

Input: *benign feature matrix* $ben[A_{xB}][f_n]$ *where* x *is the number of benign applications and* n *is the number of features, malicious feature matrix mal* $[A_{yB}][f_n]$ *where* y *is the number of malicious applications and* n *is the number of features.*

Output: *benign feature pair matrix having double covalent bond strengths* $ben_{=}[f_n][f_n]$ *and malicious feature pair matrix having double covalent bond strengths* $mal_{=}[f_n][f_n]$.

for each $i: 1 \rightarrow n$
  for each $j: i+1 \rightarrow n$
    $n(f_{ij}) = 0$
    $n(f_i) = 0$
    $n(f_j) = 0$
    for each $k: 1 \rightarrow x$
      If $(ben[A_{kB}][f_i] == 1 \&\& ben[A_{kB}][f_j] == 1)$
        $n(f_{ij}) = n(f_{ij}) + 1$
      end if
      If $(ben[A_{kB}][f_i] == 1)$
        $n(f_i) = n(f_i) + 1$
      end if
      If $(ben[A_{kB}][f_j] == 1)$
        $n(f_j) = n(f_j) + 1$
      end if
    end for
    $ben_{\rightharpoonup}[f_i][f_j] = n(f_{ij})/n(f_j)$
    $ben_{\leftharpoondown}[f_i][f_j] = n(f_{ij})/n(f_i)$
  end for
end for
for each $i: 1 \rightarrow n$
  for each $j: i+1 \rightarrow n$
    $n(f_{ij}) = 0$
    $n(f_i) = 0$
    $n(f_j) = 0$
    for each $k: 1 \rightarrow y$
      If $(mal[A_{kB}][f_i] == 1 \&\& mal[A_{kB}][f_j] == 1)$
        $n(f_{ij}) = n(f_{ij}) + 1$
      end if
      If $(mal[A_{kB}][f_i] == 1)$
        $n(f_i) = n(f_i) + 1$
      end if
      If $(mal[A_{kB}][f_j] == 1)$
        $n(f_j) = n(f_j) + 1$
      end if
    end for
    $mal_{\rightharpoonup}[f_i][f_j] = n(f_{ij})/n(f_j)$
    $mal_{\leftharpoondown}[f_i][f_j] = n(f_{ij})/n(f_i)$
  end for

(Continued)

Algorithm 1 (continued)

42. ***end for***
43. ***for each*** $i: 1 \rightarrow n$
44. ***for each*** $j: i+1 \rightarrow n$
45. $ben_{\rightleftharpoons}[f_i][f_j] = (ben_{\rightharpoonup}[f_i][f_j] + ben_{\leftharpoondown}[f_i][f_j])/2$
46. $mal_{\rightleftharpoons}[f_i][f_j] = (mal_{\rightharpoonup}[f_i][f_j] + mal_{\leftharpoondown}[f_i][f_j])/2$
47. ***end for***
48. ***end for***

The data set is assumed to have benign and malicious feature matrices in which each of the application feature vectors in the form of 0's and 1's is represented, respectively. Then, the feature *vs.* the feature matrix is calculated from these feature matrices, holding single covalent bond strengths. If f_i and f_j, are two arbitrary features, then we calculate two single covalent bond strengths for the feature pair f_{ij}, one w.r.t f_i and other w.r.t f_j., Calculating single bond strength is done from benign and malicious perspectives. The single covalent bond strength of feature *vs.* feature matrices is combined to form new feature *vs.* feature matrices holding double covalent bond strengths for both benign and malicious perspectives.

Let us suppose an instance of benign and malicious information systems, as shown in Tables 3 and 4. P_1, P_2, and P_3 denote permissions as features in both instances. Similarly, S_1, S_2, and S_3 denote system calls as features. A_{1B}, A_{2B}, A_{3B}, A_{4B}, and A_{5B} denote the benign applications in the supposed instance of benign information systems. Similarly, A_{1M}, A_{2M}, A_{3M}, A_{4M}, and A_{5M} denote the malicious applications in the supposed instance of a malicious information system.

Table 3: Supposed instance of benign information system

Benign	P_1	P_2	P_3	S_1	S_2	S_3
A_{1B}	0	1	1	1	0	0
A_{2B}	0	0	1	1	1	0
A_{3B}	1	0	1	0	1	1
A_{4B}	0	0	1	0	1	0
A_{5B}	1	0	1	0	1	0

Table 4: Supposed instance of malicious information system

Malicious	P_1	P_2	P_3	S_1	S_2	S_3
A_{1M}	1	0	0	1	1	1
A_{2M}	1	1	0	1	0	1
A_{3M}	1	1	0	0	1	1
A_{4M}	0	1	0	0	1	0
A_{5M}	0	0	1	1	0	0

After assuming the benign and malicious instances of the information systems, now we show how to calculate the single bond strengths of every feature pair. As discussed earlier, single bond strengths

of two arbitrary features are calculated from two perspectives, i.e., benign and malicious. For each perspective, the single bond strengths are calculated again from two aspects, i.e., w.r.t f_i and w.r.t f_j. The formulas for this are evident from Eqs. (1)–(4).

Eq. (1) denotes the single benign bond strength of the feature pair f_{ij} w.r.t feature f_j. As discussed earlier, the single bond is established by sharing only one pair of electrons between the two involved atoms, symbolized by a single dash (-). The same phenomenon is established in our concept represented by Eq. (1) as $ben_{\rightharpoonup}[f_i][f_j]$ .Here, the ($\rightharpoonup$) represents a single covalent bond w.r.t. to the feature at the right side of the arrow, simulating the sharing of only one electron pair. It gives us the benign score of the single covalent bond between f_i and f_j w.r.t. f_j, where $n(f_{ij})$ is the number of applications for which both features were present simultaneously in the benign feature matrix. In addition, $n(f_j)$ is defined as the number of applications for which the feature f_j is present. The value for Eq. (1) will be lying in the set [0,1]. A value of 1 indicates a strong single covalent bond while a value of 0 indicates a weak bond. The ratio of $n(f_{ij})$ w.r.t $n(f_j)$ denotes the the probability that the association between two features f_i and f_j in the is strong or weak w.r.t to the feature f_j, i.e., higher the ratio greater the association.

$$ben_{\rightharpoonup}[f_i][f_j] = n(f_{ij})/n(f_j) \tag{1}$$

$$ben_{\leftharpoondown}[f_i][f_j] = n(f_{ij})/n(f_i) \tag{2}$$

$$mal_{\rightharpoonup}[f_i][f_j] = n(f_{ij})/n(f_j) \tag{3}$$

$$mal_{\leftharpoondown}[f_i][f_j] = n(f_{ij})/n(f_i) \tag{4}$$

Eq. (2) denotes the single benign bond strength of the feature pair f_{ij} w.r.t feature f_i. Here ($\leftharpoondown$) represents a single covalent bond w.r.t. to the feature at the left side of the arrow, simulating the sharing of only one electron pair. It gives us the benign score of the single covalent bond between f_i and f_j w.r.t. f_i, where $n(f_{ij})$ is the number of applications for which both features were present simultaneously in the benign feature matrix. In addition, $n(f_i)$ is defined as the number of applications for which the feature f_i is present. The value for Eq. (2) will be lying in the set [0,1]. A value of 1 indicates a strong single covalent bond while a value of 0 indicates a weak bond.

Similarly, with the help of Eqs. (3) and (4), we can calculate $mal_{\rightharpoonup}[f_i][f_j]$ and $mal_{\leftharpoondown}[f_i][f_j]$ where the former is the single malicious bond strength of the feature pair f_{ij} w.r.t feature f_j while later is the single malicious bond strength of the feature pair f_{ij} w.r.t feature f_i. They are both calculated from the malicious feature pair matrix. The value for Eqs. (5) and (6) will be lying in the set [0,1]. A value of 1 indicates a strong double covalent bond while a value of 0 indicates a weak bond. Since the single covalent bonds are calculated from two perspectives, i.e., w.r.t f_i and f_j separately, taking their average will give the strength of association between the two features w.r.t both the perspectives. Higher the average value greater the association between both the features w.r.t both the perspectives.

$$ben_{\rightleftharpoons}[f_i][f_j] = (ben_{\rightharpoonup}[f_i][f_j] + ben_{\leftharpoondown}[f_i][f_j])/2 \tag{5}$$

$$mal_{\rightleftharpoons}[f_i][f_j] = (mal_{\rightharpoonup}[f_i][f_j] + mal_{\leftharpoondown}[f_i][f_j])/2 \tag{6}$$

Eqs. (5) and (6) calculate double covalent bond strength for the feature pair f_{ij}. $ben_{\rightleftharpoons}[f_i][f_j]$ denotes the double covalent benign bond strength, and $mal_{\rightleftharpoons}[f_i][f_j]$ denotes the double covalent malicious bond strength. As discussed, the double covalent bond is created when two pairs of electrons are shared between the participating atoms, denoted by two dashes (=). We used ($\rightleftharpoons$) to denote a double covalent bond for the feature pair f_{ij}. The benign single covalent bond strengths calculated in Eqs. (1) and (2) are used to calculate double covalent bond strength in Eq. (5), simulating the sharing of two pairs of

electrons between the participating atoms. Similarly, the malicious covalent bond strengths calculated in Eqs. (3) and (4) are used to calculate double covalent bond strengths in Eq. (6).

Tables 5 and 6 depict benign and malicious feature pair matrices representing benign and malicious double feature pair covalent bond strengths, respectively. Tables 5 and 6 are calculated from Tables 2 and 3 using Eqs. (1)–(6).

Table 5: Instance of benign feature pair matrix

Benign	P_1	P_2	P_3	S_1	S_2	S_3
P_1		0	0.7	0	0.7	0.75
P_2			0.6	0.75	0	0
P_3				0.7	0.6	0.6
S_1					0.37	0
S_2						0.33
S_3						

Table 6: Instance of malicious feature pair matrix

Benign	P_1	P_2	P_3	S_1	S_2	S_3
P_1		0.66	0	0.66	0.66	0.5
P_2			0	0.33	0.66	0.66
P_3				0.6	0	0
S_1					0.33	0.66
S_2						0.66
S_3						

3.4 Detection Phase

The double covalent benign and malicious bond strength calculated in the previous phase is used to detect an arbitrary application as malicious or benign. The whole process of the detection phase is depicted in Algorithm 2.

The testing application is first analyzed to form all possible distinct feature pairs. After this, the net benign and malicious scores are calculated based on the feature pairs formed for the test application. The net benign and malicious scores are calculated from the double covalent benign and malicious strengths stored in benign and malicious feature pair matrices, respectively.

Let us take an instance of the test Android application as A_t, then the net benign score and net malicious score of the application are calculated with the help of Eqs. (7) and (8), respectively.

$$net_{ben}(A_t) = net_{ben}(A_t) + ben_{=}[f_i][f_j] \tag{7}$$

$$net_{mal}(A_t) = net_{mal}(A_t) + mal_{=}[f_i][f_j] \tag{8}$$

In Eq. (7), the $net_{ben}(A_t)$ represents the net benign score of application A_t whereas in Eq. (8) the $net_{mal}(A_t)$ represents the net malicious score. Both equations sum up the benign and malicious feature

pair scores of all the distinct feature pairs, respectively. If $net_{mal}(A_t)$ score is greater than $net_{ben}(A_t)$ then we can deduce that the test application A_t is detected as malicious otherwise benign.

Algorithm 2: Feature pair covalent bond formation

Input: *benign feature pair matrix having double covalent bond strengths* $ben_{=}[f_n][f_n]$ *and malicious feature pair matrix having double covalent bond strengths* $mal_{=}[f_n][f_n]$.
Set of Applications $(A_1, A_2, \ldots.., A_n)$ *to be Tested*
Output: *Each of the applications is Malicious or Benign.*
for each application $(A_t)\ t: 1 \rightarrow n$
    $net_{ben}(A_t) = 0$
    $net_{mal}(A_t) = 0$
    for each $i: 1 \rightarrow n$
        for each $j: i+1 \rightarrow n$
            if $(f_i \in A_t \&\& f_j \in A_t)$
                $net_{ben}(A_t) = net_{ben}(A_t) + ben_{=}[f_i][f_j]$
                $net_{mal}(A_t) = net_{mal}(A_t) + mal_{=}[f_i][f_j]$
            end if
        end for
    end for
    If $(net_{mal}(A_t) > net_{ben}(A_t))$
        Return A_t *as Malicious*
    else
        Return A_t *as Benign*
    end if
end for

4 Results and Discussions

This section reports results obtained from each of the covalent bond pair models. Three types of detection models are formed with the help of covalent bonds pair: Permissions-permissions, system calls-system calls, and permissions-system calls.

4.1 Feature Pair Analysis

Table 7 shows the top ten highest-scoring permission pairs based on both malicious and benign covalent bond strengths between them. The maximum malicious permissions pair is INTERNET and READ_PHONE_STATE, with the malicious colavent bond strength score of 0.96. This behavior seems evident because pairing INTERNET and READ_PHONE_STATE permissions in an Android app may pose privacy and security risks. The INTERNET permission allows access to online resources, while READ_PHONE_STATE grants access to device details like phone numbers and network information. These permissions could enable an app to collect and transmit sensitive user data without consent, potentially indicating malicious intent. Similarly, the reason for other pairs could also be inferred.

Table 7: Top ten highest scoring permissions pair from both malicious and benign perspectives

Malicious		Benign	
Permissions-Permissions pair	Malicious score	Permissions-Permissions pair	Benign score
INTERNET-READ_PHONE_STATE	0.96	READ_SYNC_SETTINGS-WRITE_SETTINGS	0.98
ACCESS_NETWORK_STATE-INTERNET	0.93	BROADCAST_PACKAGE_REMOVED-BROADCAST_STICKY	0.96
ACCESS_COARSE_LOCATION-ACCESS_FINE_LOCATION	0.92	BROADCAST_PACKAGE_REMOVED-RESTART_PACKAGES	0.84
ACCESS_NETWORK_STATE-READ_PHONE_STATE	0.92	BIND_WALLPAPERS-BLUETOOTH	0.79
INTERNET-WRITE_EXTERNAL_STORAGE	0.91	QUERY_ALL_PACKAGES-WRITE_APN_SETTINGS	0.75
READ_PHONE_STATE-WRITE_EXTERNAL_STORAGE	0.89	ACCESS_MEDIA_LOCATION-INTERACT_ACCROSS_PROFILES	0.72
ACCESS_NETWORK_STATE-WRITE_EXTERNAL_STORAGE	0.88	ACCESS_NETWORK_STATE-WRITE_EXTERNAL_STORAGE	0.72
ACCESS_NETWORK_STATE-ACCESS_WIFI_STATE	0.87	INTERNET-READ_PHONE_STATE	0.71
ACCESS_WIFI_STATE-READ_PHONE_STATE	0.84	INTERNET-WRITE_EXTERNAL_STORAGE	0.70
ACCESS_WIFI_STATE –INTERNET	0.82	ACCESS_NETWORK_STATE-INTERNET	0.70

Table 8 shows system call-system call covalent bond pairs with their malicious and benign score arranged in descending order of covalent bond strengths. The top pair in this table with the highest malicious score is "getuid32-ioctl". The getuid32 system call retrieves the effective user ID of a process in Linux-based operating systems. On the other hand, the ioctl system call, which stands for "Input/Output Control," is employed in Unix-like systems to control devices beyond standard read and write operations. When used together, these system calls could be leveraged in a potentially malicious manner. For instance, a malicious program might use getuid32 to ascertain if the current user possesses administrative privileges. If affirmative, it could then utilize ioctl to manipulate a system device or resource, potentially resulting in a security breach or compromise.

Table 9 shows system call and permission pair covalent bonds arranged in descending order of their malicious and benign bond strength score, respectively. One of the top system call and permission pairs in malicious and benign pairs is clock_gettime and INTERNET. An application may use the precise timing obtained from clock_gettime with the internet access granted by the INTERNET permission to perform covert communication. The combination of precise timing and internet access could allow an application to engage in stealthy activities, making it harder to detect malicious behavior. The malicious score of this pair is 0.98, while the benign score is 0.90. Hence, its malicious intent is more in our case than normal intent. Still, one could not rule out that many legitimate

applications use these functionalities for legitimate purposes, such as measuring performance or synchronizing with online services.

Table 8: Top ten highest-scoring system call pair from both malicious and benign perspectives

Malicious		Benign	
System calls-system calls pair	Malicious score	System calls-system calls pair	Benign score
getuid32-ioctl	0.998	prctl-madvise	0.998
prctl– madvise	0.998	close-SYS_305	0.994
fstat64-SYS_305	0.998	fstat64-SYS_305	0.993
prctl-fstat64	0.997	getuid32-ioctl	0.993
close-SYS_305	0.997	close-fstat64	0.992
prctl-SYS_305	0.996	ioctl-writev	0.992
mmap2-SYS_305	0.996	madvise-mmap2	0.992
mprotect-fstat64	0.996	mprotect-SYS_305	0.991
prctl-mprotect	0.996	prctl-mmap2	0.991
close-mmap2	0.996	read-ioctl	0.991

Table 9: Top ten highest-scoring system call and permission pairs from both malicious and benign perspectives

Malicious		Benign	
System call-Permissions pair	Malicious score	System call-Permissions pair	Benign score
clock_gettime-INTERNET	0.98	clock_gettime-INTERNET	0.90
getuid32-INTERNET	0.97	ioctl-INTERNET	0.895
ioctl-INTERNET	0.97	getuid32-INTERNET	0.894
close-INTERNET	0.968	read-INTERNET	0.892
futex-INTERNET	0.968	writev-INTERNET	0.892
fadvise64_64 - INTERNET	0.968	write-INTERNET	0.889
SYS_305-INTERNET	0.967	close-INTERNET	0.877
fstat64-INTERNET	0.967	fadvise64_64-INTERNET	0.877
mprotect-INTERNET	0.966	fstat64-INTERNET	0.876
prctl -INTERNET	0.965	SYS_305-INTERNET	0.875

4.2 Detection Results

Table 10 shows the performance of various detection models. The proposed models are evaluated on five parameters, i.e., True Positive Rate (TPR), False Positive Rate (FPR), Precision, Accuracy, and F1-Score. The permissions-permissions model is static as it considers only permission-permission covalent bond score for detecting Android Malware applications. The system call-system call covalent bond pair model is dynamic and has better results in the evaluation parameters, which is evident from

the fact that dynamic features consider the run time behavior of the application while static feature does not. Hence, those malicious behavior that are activated at run time uncovers hidden insights that are helpful in the identification of malicious application. The next model is the permissions–system call model, a hybrid model in which a covalent bond pair is formed among permissions and system calls, and their bond strengths are used to detect malicious applications. This model, which is a hybrid, has even better evaluation parameters than the system call-system call detection model. The apparent reason seems to be the uncovering of system calls and permissions bonding. The permission requested by the application is not alone responsible for malicious behavior because benign applications may also use the same permission. The combination of permission with system calls allows a more detailed examination of an application's behavior. Permissions provide a high-level overview of what resources an app may access, while system calls offer a finer-grained view of actual interactions with the system. The Permissions-System calls model shown is the best of all. This model is a hybrid model and achieves an overall accuracy of 97.50%, which is better than both static and dynamic models. The confusion matrix for the permissions-system call model is given in Table 11.

Table 10: Performance of proposed detection models

Detection model	TPR	FPR	Precision	Accuracy	F1-Score
Permissions-permissions	92.27%	5.80%	94.98%	93.39%	93.84%
System calls-system calls	95.97%	4.43%	96.06%	95.78%	96.01%
Permissions-system calls	97.77%	2.81%	97.49%	97.50%	97.63%

Table 11: Confusion matrix of proposed detection model

		Predicted	
		Malicious	Benign
Actual	Malicious	True positive	False positive
	12415	12104	311
	Benign	False negative	True negative
	11027	275	10752

4.3 Detection Results on Unknown Samples

We comprehensively evaluate our proposed model on unknown samples. The sample is taken from the CICAndMal2017 [37] data set. A total of 1800 samples were taken, of which 1000 were malicious, and 800 were benign. These are the unseen samples as they are in the form of apks. We first installed these applications in a virtual environment, and then random clicks were done on installed applications for nearly a minute. The generated system calls are captured with the help of a strace script. The permissions were extracted from the Android manifest file of each application after unpacking each application using the apk tool. The observed result shows an accuracy of 96.20%. The details of the results are represented in Tables 12 and 13.

Table 12: Performance of proposed detection models on unknown samples

Detection model	TPR	FPR	Precision	Accuracy	F1-Score
Permissions-permissions	93.32%	7.57%	94.%	92.88%	93.62%
System calls-system calls	94.30%	5.11%	96%	94.55%	95.14%
Permissions-system calls	95.33%	2.5%	98.%	96.20%	96.64%

Table 13: Confusion matrix of proposed detection model on unknown samples

		Predicted	
		Malicious	Benign
Actual	Malicious	True positive	False positive
	1000	980	20
	Benign	False negative	True negative
	800	48	752

4.4 Comparison with Other Related Works

We comprehensively evaluate the detection results obtained from our proposed method, comparing them with findings from previous studies in the literature focusing on Android malware detection. We implemented several state-of-the-art techniques on our data set and to facilitate this comparison, we provide a concise summary in Table 14. Upon examination of these results, it becomes apparent that our proposed methodology outperforms all the aforementioned related works regarding detection accuracy, demonstrating its superior performance compared to existing approaches. Moreover, the data set used by all the approaches was old and outdated. The data set used by us is the latest, and it chronicles the entire history of Android, covering the years from 2008 to 2020 while also accounting for the distinct dynamic data sources.

Table 14: Comparison of proposed model with related works

Methodology	Approach	Features set used	No. of applications	TPR	Accuracy
Talha et al. [10]	Static	Permissions	1853 Benign and 6909 Malicious	86.23%	86.44%
PermPair [11]	Static	Permissions	5993 benign and 7533 malicious	94.11%	94.54%
Li et al. [38]	Static	Permissions	5494 malicious & 310,926 benign	91.50%	91.80%
Xiao et al. [39]	Dynamic	System call	1227 Malicious & 1189 benign	96.55%	97%

(Continued)

Table 14 (continued)

Methodology	Approach	Features set used	No. of applications	TPR	Accuracy
Xiao et al. [40]	Dynamic	System call	3567 Malicious and 3536 Benign	92%	92.60%
Surendran et al. [41]	Dynamic	System call	1150 Malicious and 1600 Benign	93.20%	93.50%
Guerra-Manzanares et al. [42]	Dynamic	System call	28,343 Malicious and 34,981 Benign	93.60%	94.40%
Guerra-Manzanares et al. [43]	Static	Permissions	4174 Malicious and 37,020 Benign	92.80%	93.50%
Guerra-Manzanares et al. [44]	Dynamic	System call	41,382 Malicious and 36,755 Benign	95.50%	95.90%
Proposed approach	**Hybrid**	**Permissions and system call**	**41,382 Malicious and 36,755 Benign**	**97.77%**	**97.50%**

4.5 Limitations

In this subsection, we address certain ambiguities in our proposed approach. Specifically, our model relies on feature pairs to assess applications. Some malware samples with a limited number of features may go undetected. To bypass detection, attackers may incorporate commonly used features into the malware, thereby generating a more significant number of ordinary feature pairs. Additionally, we have observed that when a feature pair appears only once in the malicious samples, and both individual features have a frequency of one for a specific application, it results in a malicious covalent bond strength of one. This misrepresents the actual strength of the bond, potentially elevating the significance of an otherwise insignificant feature pair and leading to misclassification. We plan to address these limitations by exploring the potential of incorporating additional components like intent filters, hardware specifications, and API call logs for more efficient detection alongside the existing focus on permissions and system calls.

5 Conclusion and Future Work

This study established covalent bonds between permissions and system calls to evaluate their combined impact. We introduced a novel methodology for calculating these pairs' Covalent Bond Strength Score, resulting in both malicious and benign scores. These scores were then utilized in our Android malware detection technique.

We thoroughly compared our proposed model and other advanced detection methods. Our results indicate that our model outperforms similar state-of-the-art models in performance. In the future, our research will analyze additional components of the manifest file, such as intent filters and hardware specifications, to further enhance detection accuracy.

Acknowledgement: Not applicable.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: All authors contributed to the study's conception and design. Material preparation, data collection and analysis were performed by Rahul Gupta. The first draft of the manuscript was written by Rahul Gupta and all authors commented on previous versions of the manuscript. All authors read and approved the final manuscript.

Availability of Data and Materials: Data sharing is not applicable to this article as no datasets were generated.

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

References

[1] P. Faruki *et al.*, "Android security: A survey of issues, malware penetration, and defenses," *IEEE Commun. Surv. Tutor.*, vol. 17, no. 2, pp. 998–1022, 2015. doi: 10.1109/COMST.2014.2386139.

[2] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proc. 1st ACM Workshop Secur. Privacy Smartphones Mobile Devic. (SPSM'11)*, New York, NY, USA, Association for Computing Machinery, 2011, pp. 3–14.

[3] R. Surendran, T. Thomas, and S. Emmanuel, "Detection of malware applications in android smartphones," *World Scient. Ref. Innov.*, vol. 1, pp. 211–234, 2018. doi: 10.1142/10209.

[4] D. Wagner and R. Dean, "Intrusion detection via static analysis," in *Proc. 2001 IEEE Symp. Secur. Privacy*, Oakland, CA, USA, 2001, pp. 156–168.

[5] B. B. H. Kang and A. Srivastava, "Dynamic malware analysis," in *Encycl. Cryptography Security*, Cham: Springer, 2011, pp. 367–368.

[6] G. Fraser and A. Arcuri, "Automated test generation for java generics," in *Int. Conf. Soft. Quality*, Springer, 2014, pp. 185–198.

[7] J. Newsome and D. Song, "Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software," *Network and Distributed System Security Symposium*, vol. 5, 2005, pp. 3–4.

[8] R. Zhang, S. Huang, Z. Qi, and H. Guan, "Combining static and dynamic analysis to discover software vulnerabilities," in *Fifth Int. Conf. Innov. Mobile Internet Serv. Ubiq. Comput. (IMIS)*, IEEE, 2011, pp. 175–181.

[9] R. Zhang, S. Huang, Z. Qi, and H. Guan, "Static program analysis assisted dynamic taint tracking for software vulnerability discovery," *Comput. Math. Appl.*, vol. 63, no. 2, pp. 469–480, 2012. doi: 10.1016/j.camwa.2011.08.001.

[10] K. A. Talha, D. I. Alper, and C. Aydin, "APK Auditor: Permission-based Android malware detection system," *Digital Invest.*, vol. 13, pp. 1–14, Jun. 2015. doi: 10.1016/j.diin.2015.01.001.

[11] A. Arora, S. K. Peddoju, and M. Conti, "PermPair: Android malware detection using permission pairs," *IEEE Trans. Inf. Forensics Secur.*, vol. 15, pp. 1968–1982, 2020.

[12] K. Khariwal, J. Singh, and A. Arora, "Ipdroid: Android malware detection using intents and permissions," in *2020 Fourth World Conf. Smart Trends in Syst., Security Sustain. (WorldS4)*, IEEE, 2020, pp. 197–202.

[13] W. Enck *et al.*, "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Trans. Comput. Syst.*, vol. 32, no. 2, 2014. doi: 10.1145/2619091.

[14] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning and X. S. Wang, "AppIntent: Analyzing sensitive data transmission in Android for privacy leakage detection," in *Proc. ACM SIGSAC Conf. on Comput. Communicati. Secur.*, 2013, pp. 1043–1054.

[15] S. Wang, Q. Yan, Z. Chen, B. Yang, C. Zhao and M. Conti, "Detecting android malware leveraging text semantics of network flows," *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 5, pp. 1096–1109, May 2018. doi: 10.1109/TIFS.2017.2771228.

[16] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and efficient behavior-based Android malware detection and prevention," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 1, pp. 83–97, 2018. doi: 10.1109/TDSC.2016.2536605.

[17] M. Sun, X. Li, J. C. S. Lui, R. T. B. Ma, and Z. Liang, "Monet: A user-oriented behavior-based malware variants detection system for Android," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 5, pp. 1103–1112, May 2017. doi: 10.1109/TIFS.2016.2646641.

[18] M. Xia, L. Gong, Y. Lyu, Z. Qi, and X. Liu, "Effective real-time Android application auditing," in *Proc. IEEE Symp. Security and Privacy*, May 2015, pp. 899–914.

[19] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, "A review on feature selection in mobile malware detection," *Digital Invest.*, vol. 13, no. 6, pp. 22–37, 2015. doi: 10.1016/j.diin.2015.02.001.

[20] H. Shahriar, M. Islam, and V. Clincy, "Android malware detection using permission analysis," in *Southeast Conf. 2017*, IEEE, 2017, pp. 1–6.

[21] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket," in *Proc. 2014 Netw. Distributed Syst. Secur. Symp.*, 2014.

[22] L. Cen, C. S. Gates, L. Si, and N. Li, "A probabilistic discriminative model for android malware detection with decompiled source code," *IEEE Trans. Dependable Secure Comput.*, vol. 12, no. 4, pp. 400–412, 2015. doi: 10.1109/TDSC.2014.2355839.

[23] J. Qiu *et al.*, "Cyber code intelligence for android malware detection," *IEEE Trans. Cybern.*, vol. 53, no. 1, pp. 617–627, Jan. 2023. doi: 10.1109/TCYB.2022.3164625.

[24] M. İbrahim, B. Issa, and M. B. Jasser, "A method for automatic android malware detection based on static analysis and deep learning," *IEEE Access*, vol. 10, pp. 117334–117352, 2022.

[25] W. Yu, H. Zhang, L. Ge, and R. Hardy, "On behavior-based detection of malware on android platform," in *2013 IEEE Global Commun. Conf.*, IEEE, 2013, pp. 814–819.

[26] M. Dimjaševic, S. Atzeni, I. Ugrina, and Z. Rakamaric, "Evaluation of android malware detection based on system calls," in *Proc. 2016 ACM Int. Workshop Secur. Priv. Anal.*, New York, NY, USA, ACM, 2016, pp. 1–8.

[27] I. Burguera, U. Zurutuza, and S. N. Tehrani, "Crowdroid: Behavior-based malware detection system for android," in *Proc. 1st ACM Workshop on Secur. Priv. Smartphones and Mobile Devices*, ACM, 2011, pp. 15–26.

[28] A. Amamra, J. M. Robert, A. Abraham, and C. Talhi, "Generative versus discriminative classifiers for android anomaly-based detection system using system calls filtering and abstraction process," *Secur. Commun. Netw.*, vol. 9, no. 16, pp. 3483–3495, 2016. doi: 10.1002/sec.1555.

[29] T. Lu and J. Wang, "F2DC: Android malware classification based on raw traffic and neural networks," *Comput. Netw.*, vol. 217, no. 4, pp. 109320, 2022. doi: 10.1016/j.comnet.2022.109320.

[30] M. S. Hossain *et al.*, "Android ransomware detection from traffic analysis using metaheuristic feature selection," *IEEE Access*, vol. 10, pp. 128754–128763, 2022.

[31] A. Arora, and S. K. Peddoju, "Ntpdroid: A hybrid android malware detector using network traffic and system permissions," in *2018 17th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./12th IEEE Int. Conf.Big Data Sci. Eng. (Trust-Com/BigDataSE)*, IEEE, 2018, pp. 808–813.

[32] L. Onwuzurike, M. Almeida, E. Mariconti, J. Blackburn, G. Stringhini, and E. de Cristo-faro, "A family of droids-android malware detection via behavioral modeling: Static vs dynamic analysis," in *2018 16th Annual Conf. Privacy, Secur. and Trust (PST)*, IEEE, 2018, pp. 1–10.

[33] T. Blasing, L. Batyuk, A. D. Schmidt, S. A. Camtepe, and S. Albayrak, "An android application sandbox system for suspicious software detection," in *2010 5th Int. Conf. Malicious and Unwanted Soft. (MALWARE 2010)*, IEEE, 2010, pp. 55–62.

[34] A. I. Ali-Gombe, B. Saltaformaggio, J. Ramanujam, D. Xu, and G. G. Richard III, "Toward a more dependable hybrid analysis of android malware using aspect-oriented programming," *Comput. Secur.*, vol. 73, no. 1, pp. 235–248, 2018. doi: 10.1016/j.cose.2017.11.006.

[35] A. Guerra-Manzanares, H. Bahsi, and S. Nõmm, "KronoDroid: Time-based hybrid-featured dataset for effective android malware detection and characterization," *Comput. Secur.*, vol. 110, pp. 102399, 2021. doi: 10.1016/j.cose.2021.102399.

[36] J. E. House and K. A. House, *Descriptive Inorganic Chemistry*, 3rd ed. Academic Press, 2016. doi: 10.1016/C2014-0-02460-4

[37] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark android malware datasets and classification," in *Proc. 52nd IEEE Int. Carnahan Conf. Secur. Technol. (ICCST)*, Montreal, Quebec, Canada, 2018.

[38] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an and H. Ye, "Significant permission identification for machine-learning-based android malware detection," *IEEE Trans. Industr. Inform.*, vol. 14, no. 7, pp. 3216–3225, Jul. 2018. doi: 10.1109/TII.2017.2789219.

[39] X. Xiao, Z. Wang, Q. Li, S. Xia, and Y. Jiang, "Back-propagation neural network on Markov chains from system call sequences: A new approach for detecting Android malware with system call sequences," *IET Inf. Secur.*, vol. 11, no. 1, pp. 8–15, 2017. doi: 10.1049/iet-ifs.2015.0211.

[40] X. Xiao, S. Zhang, F. Mercaldo, G. Hu, and A. K. Sangaiah, "Android malware detection based on system call sequences and LSTM," *Multimed. Tools Appl.*, vol. 78, no. 4, pp. 3979–3999, 2019. doi: 10.1007/s11042-017-5104-0.

[41] R. Surendran, T. Thomas, and S. Emmanuel, "On existence of common malicious system call codes in android malware families," *IEEE Trans. Reliab.*, vol. 70, no. 1, pp. 248–260, Mar. 2021. doi: 10.1109/TR.2020.2982537.

[42] A. Guerra-Manzanares, M. Luckner, and H. Bahsi, "Concept drift and cross-device behavior: Challenges and implications for effective android malware detection," *Comput. Secur.*, vol. 120, no. 10, pp. 102757, 2022. doi: 10.1016/j.cose.2022.102757.

[43] A. Guerra-Manzanares, H. Bahsi, and M. Luckner, "Leveraging the first line of defense: A study on the evolution and usage of android security permissions for enhanced android malware detection," *J. Comput. Virol. Hacking Tech.*, vol. 19, no. 1, pp. 65–96, 2023. doi: 10.1007/s11416-022-00432-3.

[44] A. Guerra-Manzanares, M. Luckner, and H. Bahsi, "Android malware concept drift using system calls: Detection, characterization and challenges," *Expert Syst. Appl.*, vol. 206, pp. 117–200, 2022. doi: 10.1016/j.eswa.2022.117200.