# Understanding Concept Drift with Deprecated Permissions in Android Malware Detection

Ahmed Sabbah, Radi Jarrar, Samer Zein, David Mohaisen, *Senior Member, IEEE*

**Abstract**—Permission analysis is a widely used method for Android malware detection. It involves examining the permissions requested by an application to access sensitive data or perform potentially malicious actions. In recent years, various machine learning (ML) algorithms have been applied to Android malware detection using permission-based features and feature selection techniques, often achieving high accuracy. However, these studies have largely overlooked important factors such as protection levels and the deprecation or restriction of permissions due to updates in the Android OS—factors that can contribute to concept drift.
In this study, we investigate the impact of deprecated and restricted permissions on the performance of machine learning models. A large dataset containing 166 permissions was used, encompassing more than 70,000 malware and benign applications. Various machine learning and deep learning algorithms were employed as classifiers, along with different concept drift detection strategies. The results suggest that Android permissions are highly effective features for malware detection, with the exclusion of deprecated and restricted permissions having only a marginal impact on model performance. In some cases, such as with CNN, accuracy improved. Excluding these permissions also enhanced the detection of concept drift using a year-to-year analysis strategy. Dataset balancing further improved model performance, reduced low-accuracy instances, and enhanced concept drift detection via the Kolmogorov–Smirnov test.

**Index Terms**—Android Malware; Machine Learning; Malware Detection, Concept Drift

## 1 Introduction

Mobile devices are an essential tool in everyday life, providing users with access to a wide range of applications for communication, banking, entertainment, and productivity. Two operating systems dominate the mobile market, Google Android and Apple iOS, with Android taking 71% of the market share by 2024 [1]. Android employs a permission-based security model that grants applications specific privileges to regulate access to sensitive resources. These permissions control access to hardware components (e.g., camera, microphone, etc.), user data (e.g., contacts, messages, location), and system functionalities (e.g., network access, storage, background processes) [2]. Android permissions aim to balance security and usability, ensuring that applications can only access resources explicitly allowed by the user.

While the permission system is designed to protect user privacy and device security, permissions can be misused, intentionally by adversaries [3] or unintentionally by novice developers who may request permissions without fully understanding their implications [4]. This is clear in the finding of more 26% of more than 83k applications examined that contained at least one bad practice of custom permission issue [5]. However, while it is important to investigate the bad practices of using permissions, this study focuses on intentional misuse by attackers by requesting excessive permissions to access user data or control device functionality. For example, some malware applications masquerade themselves as legitimate apps, such as flashlights or weather apps, but request permission to read SMS messages, access call logs, or track user locations.

Due to the risks associated with permissions and their role in granting access to user and system resources, permissions are a target for adversaries and third-party apps, enabling them to control devices and access sensitive data. This, in turn, allows defenders to extract permission-based features for malware detection with machine learning algorithms [6], [7], [8], [9]. Many studies have recognized the importance of permissions and have focused on improving the performance of detection models that use them [6], [10], often through feature selection and optimization [7], [8], [9]. Other research has explored combining permissions with additional features, such as APIs, to enhance performance [11], [12], yielding strong results. However, most of these studies assume that permissions and their usage patterns remain static over time, failing to consider how changes in permissions impact detection.

Concept drift is closely associated with machine learning models, where performance degrades over time due to changes in the underlying data distribution. Researchers have investigated concept drift from various perspectives. For instance, Hoens *et al.* reviewed methods for detecting concept drift and highlighted the challenge of class imbalance in data streams [13]. Ditzler *et al.* examined existing methods for detecting concept drift, considering both active and passive approaches [14]. Webb *et al.* introduced formal definitions of concept drift and proposed a taxonomy that includes various types, such as drift duration, drift transition, and drift recurrence [15]. Krawczyk *et al.* surveyed ensemble learning methods for data stream classification and regression tasks in the context of concept drift [16]. Moreover, some studies have focused on adaptation meth-

- *Ahmed Sabbah, Radi Jarrar, and Samer Zein are with the Department of Computer Science, University of Birzeit, Palestine. E-mail: asabah@birzeit.edu, rjarrar@birzeit.edu, szain@birzeit.edu.*
- *David Mohaisen is with the Department of Computer Science, University of Central Florida, Orlando, FL 32816 USA. E-mail: mohaisen@ucf.edu (Corresponding author).*

ods in machine learning to address concept drift [17].

In the context of Android malware detection, concept drift has been gaining momentum, due to ML widespread adoption in the defense-attack arms race. Ensemble classifiers with sliding windows and feature selection are used to adaptively detect malware in streaming data [18]. Retraining methods used for concept drift detection and sampling methods to maintain detector performance in changing environments [19]. Self-training with pseudo-labels is adapted to handle shifting data distributions, mitigating concept drift, and reducing annotation efforts when combined with active learning [20]. Chow *et al.* proposed a framework for analyzing dataset drift exploring root causes [21].

Despite these findings, concept drift studies are often limited to the natural evolution of usage patterns while ignoring an important aspect of evolution: **deprecation**. Since permissions are continuously added, deprecated, or restricted, models trained on outdated permissions may become ineffective, leading to a decline in detection accuracy. Malware detection models trained on older data may struggle to accurately classify newer malware due to changes in permission usage patterns. This issue has been largely neglected in existing research, despite its critical implications for the long-term effectiveness of malware detection.

In this work, we investigate the impact of Android permission evolution on malware detection by analyzing how deprecated and restricted permissions contribute to concept drift. We evaluate the effectiveness of permissions as features in malware classification by training models such as Random Forest (RF), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN) on historical data and testing them on newer datasets. Additionally, we assess how removing deprecated and restricted permissions affects model performance over time. By quantifying concept drift in both traditional machine learning and deep learning models, we provide insights into the challenges of maintaining effective malware detection systems in the context of a rapidly evolving Android permission landscape.

**Contributions.** We make the following contributions:

1) *Analyzing the use of permissions in malware and benign applications.* We examine how permissions are used in malicious and benign applications, identifying patterns and differences that can inform effective permission abuse prevention and malware detection strategies
2) *Investigating the impact of permission evolution on concept drift.* We analyze the effect of permission changes over time by training models on historical datasets and testing them on newer data. For example, we train on permissions from 2008 and evaluate on data from subsequent years (2009–2020).
3) *Examining the effect of deprecated and restricted permissions on model stability.* We assess how the removal of deprecated and restricted permissions influences malware detection performance, highlighting potential pitfalls in using outdated or evolving feature sets.

**Organization.** The background is outlined in section 2, followed by the methodology in section 3, the results and discussion in section 4, the related work in section 5, followed by concluding remarks in section 6.

## 2 Background

Android malware apps can be installed via official stores, third parties, or social engineering tactics [22] to gain unauthorized access and exploit root privileges without user consent [23]. Android malware varies widely and can be categorized based on shared characteristics or behaviors. Common types include banking malware, Trojans, spyware, and worms, with each type further classified into families based on specific traits or attack patterns.

### 2.1 Android Package Kit (APK)

The Android Package Kit (APK) is an archive file format used to deploy and install applications on the Android operating system. APK files can be downloaded and installed from the official Google Play Store or from third-party sources. To install APKs from outside the official app store, Android users must enable a setting feature that allows installations from unknown sources. The main components of the APK file are as follows:

- `AndroidManifest.xml`: This file contains XML metadata information about components of the application, such as security permissions, activities, and services.
- `Classes.dex`: This file contains the source code of an application written in Java and compiled to a Dalvik executable with *.edx* extension.
- `Resources.arsc`: A binary XML file that includes precompiled application resources [24].
- `Resources (res/)`: A folder containing non-compiled resources that the application requires at run time, such as menus, images, layouts, and database use.
- `Assets (assets/)`: Optional folder containing application assets that Asset Manager can retrieve.
- `Libraries (lib/)`: Optional folder containing code compiled for various processors; e.g., ARM and x86 [24].
- `META-INF`: Folder containing the `MANIFEST.MF` file. Further, it includes the application developer's signature, which can be used to authenticate an external developer.

### 2.2 Android Security

As an open-source platform, Android has been an ongoing target of hackers. Consequently, ensuring the security of applications remains a critical concern and a significant challenge. To mitigate this risk, Android OS implements a two-level security framework: one at the Linux kernel level and another at the application level [25]. The security mechanisms of Android include the following features:

- **Sandbox:** Each application is assigned a unique user ID (UID) and runs in its own process and isolated space. Thus, one application with its own UID did not allow it to be accessed or modified by another application. Additionally, the application running in its own sandbox has limited access to system resources.
- **Permissions:** Any application must define the permission needs of other application components or Android resources in the AndroidManifest.xml file. The user must grant these permissions during app installation; otherwise, the application cannot be installed on the device. Moreover, permission can be granted during the application runtime, which is called dynamic permissions.

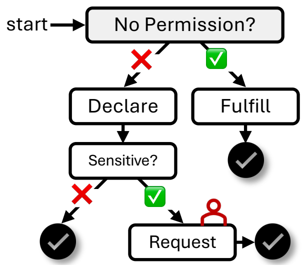


Figure 1: Workflow for Android permissions.

- **Signatures:** The Android application must be stamped and digitally signed with a certificate that identifies the developer of the application. A developer with the same certificate can update the application in the future. In addition, applications with this signature can trust each other by sharing the UID between them.
- **Secure Inter-process Communication (IPC):** IPC protocols allow an application to communicate with remote servers and also other applications.

## 2.3 Android Permissions

Android apps are required to request explicit permission from users before accessing sensitive resources or data, such as the camera, location, or microphone. This ensures that applications cannot access personal information without the user's knowledge and explicit consent. The workflow of using app permissions is illustrated in Figure 1.

The Android operating system provides normal, dangerous, signature, custom, install-time, and special permissions [2], which we highlight in the following:

1) **Normal Permissions:** These permissions are automatically granted during installation and are associated with basic functionalities like internet access, Bluetooth, and NFC. They pose minimal privacy and security risks as they do not access sensitive data.
2) **Dangerous Permissions:** Require explicit user approval during execution as they grant access to sensitive data or critical functions, such as contacts, location, microphone, and camera, posing potential privacy and security risks.
3) **Signature Permissions:** These permissions enable apps signed with the same certificate (private key) to share privileged information or resources. New apps are automatically granted these permissions if signed with the same certificate as existing applications.
4) **Custom Permissions:** These permissions are defined by developers to enforce access control between apps. Moreover, these permissions can be classified as normal or dangerous depending on the level of access they provide. This type is widely used but often undocumented and hidden from users, posing security risks by enabling indirect access to protected resources without consent [26].
5) **Install-Time Permissions:** Introduced in Android 6.0, these permissions are granted by users during installation instead of being requested at runtime.
6) **Special Permissions:** These permissions are restricted to system apps or those signed with the same certificate as system applications. Moreover, these permissions are not available to third-party apps and must be explicitly granted by the platform or device manufacturer.

## 2.4 Permission-Based Malware Detection

Android malware detection using permission-based analysis involves multiple stages to extract, preprocess, and classify apps based on their permissions. The pipeline can be structured into the following key components:

**Data Collection and Preprocessing.** The android malware detection begins with collecting APK files from various sources, including official stores, third-party repositories, and malware datasets. Each APK undergoes static analysis to extract requested permissions from the `AndroidManifest.xml` file. Let $A = \{a_1, a_2, \ldots, a_n\}$ represent a set of applications, and each app $a_i$ requests a subset of permissions $P_i = \{p_1, p_2, \ldots, p_m\}$, where $P$ is the global set of all possible permissions.

**Feature Engineering and Selection.** Extracted permissions serve as features for classification. Given an application $a_i$, its permission vector $v_i$ can be represented as $v_i = (x_1, x_2, \ldots, x_m)$, where $x_j \in \{0, 1\}$ indicates whether permission $p_j$ is requested or not in the application.

**Training and Classification.** A machine learning or deep learning classifier is trained using labeled data. Given a training dataset $D = \{(v_1, y_1), (v_2, y_2), \ldots, (v_n, y_n)\}$, where $y_i \in \{0, 1\}$ denotes whether an app is benign (0) or malicious (1), the model learns a mapping function $f : v \to y$.

Classification algorithms used in permission-based malware detection include *traditional machine learning algorithms*, such as random forest (RF), support vector machines (SVM), decision trees (DT), and *deep learning algorithms*, such as neural networks (NN), long short-term memory (LSTM), and graph neural networks (GNN).

## 2.5 Concept Drift

The phenomenon of concept drift refers to the unexpected change in the statistical properties or defining features of the target variable over time in non-stationary data distributions [27]. Mathematically, let $P_t(X, Y)$ represent the joint probability distribution of the input features $X$ and the target variable $Y$ at time $t$. Concept drift occurs when: $P_{t_1}(X, Y) \neq P_{t_2}(X, Y), \quad \text{for } t_1 \neq t_2$. This change can manifest in different forms: abrupt, incremental, gradual, and recurring [28]. Each form corresponds to distinct transitions in the data distribution:

**Abrupt Drift.** A sudden shift in the data distribution at a specific point in time, mathematically defined as: $P_t(X, Y) = \begin{cases} P_A(X, Y), & t < t_c \\ P_B(X, Y), & t \geq t_c \end{cases}$, where $t_c$ is the time at which the drift occurs, and $P_A$ and $P_B$ represent different distributions before and after the drift.

**Incremental Drift.** In this type of drift, the data distribution evolves progressively over time, making the transition smooth, expressed as follows:

$$P_t(X, Y) = \alpha_t P_A(X, Y) + (1 - \alpha_t) P_B(X, Y),$$
$$0 < \alpha_t < 1, \quad \lim_{t \to \infty} \alpha_t = 0.$$

**Gradual Drift.** In this type, both old and new distributions coexist during a transition period before the new distribution fully replaces the old one:

$$P_t(X,Y) = \begin{cases} (1-\beta_t)P_A(X,Y) + \beta_t P_B(X,Y), & t_c \le t \le t_d \\ P_B(X,Y), & t > t_d \end{cases},$$

where $\beta_t$ gradually increases from 0 to 1 over the transition period $[t_c, t_d]$ leading to the gradual drift.

**Recurring Drift.** In this type, a previously observed concept reappears at later time steps:

$$P_t(X,Y) \in \{P_A(X,Y), P_B(X,Y), P_C(X,Y), \dots\},$$
$$\text{where } P_t(X,Y) \text{ cycles over time.}$$

In conclusion, abrupt drift involves rapid concept transitions, incremental drift refers to slower changes, gradual drift involves periodic concept shifts, and recurring drift is characterized by the reappearance of earlier concepts [28]. These forms of concept drift indicate changes in the underlying data distribution, necessitating adaptive learning models to maintain performance in dynamic environments.

### 2.6 Android Permissions Deprecation

The Android permission system is not static and continuously evolves as new hardware is introduced, security policies change, and user expectations shift. Consequently, Google has introduced new permissions, deprecated old ones, and restricted access to certain sensitive permissions to enhance security [2], as shown in the timeline in Figure 2. Deprecated permissions are those removed or replaced with more secure alternatives. For example, the `GET_TASKS` permission, which allowed applications to retrieve information about running tasks, was deprecated in Android 5.0 (API Level 21) due to privacy risks [29]. Similarly, `READ_CALL_LOG` was restricted in Android 9.0 (API Level 28) to prevent unauthorized access to call history [29]. Restricted permissions are those moved to higher security levels, meaning only system or explicitly approved apps can use them. For example, Android 10 (API Level 29) introduced restrictions on `ACCESS_BACKGROUND_LOCATION`, limiting its use to prevent apps from stealthily tracking users. Additionally, changes were made to prevent silent access to the device's screen contents by restricting the scope of `READ_FRAME_BUFFER` to signature access [30].

## 3 Methodology

Concept drift occurs when the relationship between features and target labels changes over time [31]. In the context of Android malware detection, the continuous evolution of the Android permission system–driven by security updates, hardware advancements, and changes in app development practices–affects the reliability of permission-based detection models. Although permissions have been widely used as features in machine learning models for malware detection, prior studies have largely treated them as static attributes, overlooking how their deprecation, restriction, or modification over time can impact model effectiveness.

We aim to bridge this gap by: (1) reevaluating the effectiveness of permissions as features for machine learning and deep learning models, (2) assessing their sensitivity to concept drift as permissions evolve, and (3) analyzing the impact of deprecated and restricted permissions on model performance. Consequently, we seek to answer the following research questions.

- **RQ-1:** How do permission-based features impact malware detection accuracy across ML and DL models?
- **RQ-2:** How does permission sensitivity to concept drift affect Android malware detection over time?
- **RQ-3:** How do deprecated and restricted permissions influence model stability, accuracy, and drift?

To achieve the study goals and answer the questions, we propose the following strategies as steps of exploration:

1) **Ignoring the Chronological Context.** Algorithms were tested on all features, including timestamps.
2) **Year-to-Year Strategy.** Models were trained on data from one year and tested on subsequent years (e.g., trained on 2008, tested on 2009–2020).
3) **Exclusion Strategy.** Specific permission categories—deprecated (D), restricted (R), and not-for-use-by-third-party (N)—were systematically removed to assess their impact on malware detection and concept drift. Results were compared against a baseline using all permissions (ALL) in both prior strategies.

### 3.1 Model Workflow

In this work, this test was applied to a simulated dataset for concept drift detection [32]. The dataset was divided into 12 blocks, sequentially labeled, and the $p$-values were calculated over time. We used this test to compute the Cumulative CDFs of two sample groups (e.g., CDFs for the accuracy of training years 2008 and 2010). The KS test measures the maximum absolute difference between the CDFs, and pairwise comparisons were performed by computing the KS test statistic and the corresponding $p$-value for each pair of training years (T1, T2). A significant $p$-value ($< 0.05$) indicates a possible drift between distributions.

Figure 3 presents a four-stage workflow of this work. In the **pre-processing** phase, benign and malware are collected from real devices and emulators, followed by the extraction of permissions and their corresponding API levels to collect metadata. In the **feature space** stage, permissions are mapped to categories such as restricted, deprecated, or not-for-use-by-third-party, and merged with the dataset based on API release year. The **balance phase** applies the balance algorithm. Finally, in the Classification and Drift phase, temporal evaluation strategies, year-to-year, and exclusion-based are applied using machine learning and deep learning classifiers, with performance evaluated through metrics like accuracy, precision, recall, F1 score, and drift detection via the Kolmogorov–Smirnov (KS) test.

### 3.2 Dataset Overview

The Kronodroid dataset [33] was built using a combination of static and dynamic features extracted from Android applications. The source of the malware samples was selected from different databases: VirusTotal, Drebin, VirusShare, and AMD. The benign apps were selected from APKMirror, F-droid, and MARVIN. This dataset includes 489 dynamic

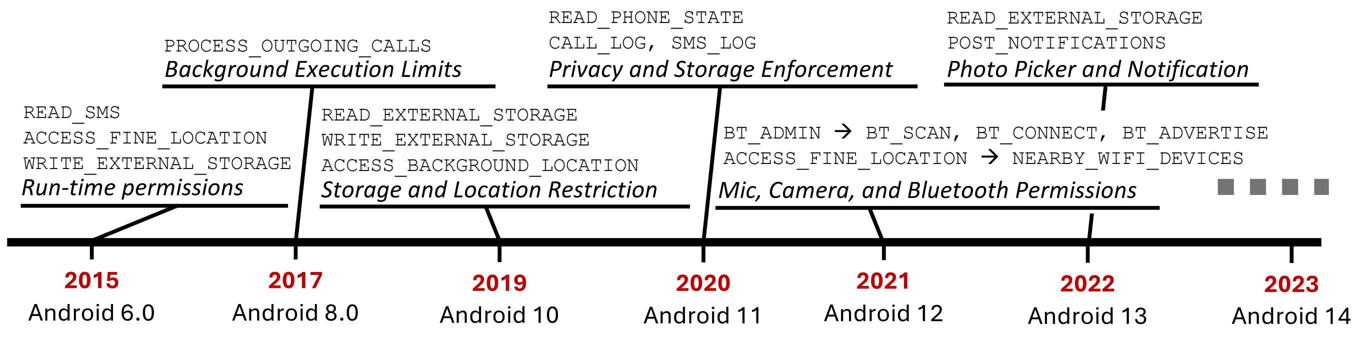


Figure 2: A timeline of the Android permission system evolution and deprecation across various versions. The permissions are added or removed, and Android 14 and 15 (2023 and 2024) were removed due to the lack of space.

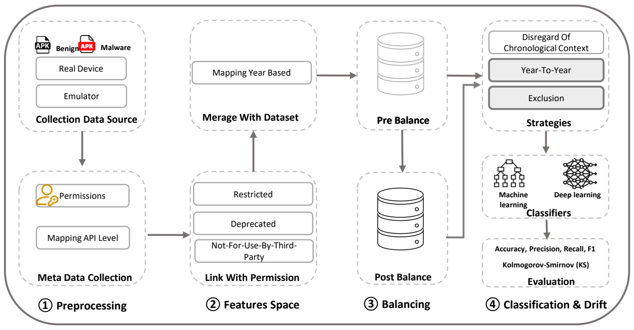


Figure 3: A four-stage framework for Android malware detection and drift analysis: data preprocessing, API-level permission-based feature extraction, dataset balancing, and classification using machine and deep learning. Concept drift is identified using KS statistics and standard metrics.

and static features extracted from Android applications that covered the years 2008 to 2020. The emulator dataset contains 28,745 malware samples and 35,246 benign samples. The real device dataset includes 41,382 malware samples and 36,755 benign samples. Figure 4 shows the number of malware and benign applications for each year. For permissions features, 166 features were selected from the real device and emulator datasets. An app takes a value of 1 when requesting permission and 0 otherwise. Kronodroid did not provide different levels of protection for each permission, such as normal, signature, and dangerous.

### 3.3 Data Analysis

Because Knorodoid lacks information about the permissions themselves, we addressed this gap by collecting metadata for each permission from the official Android website [34]. This metadata includes critical attributes, such as the four types of protection levels (normal, dangerous, signature, and not for use by third parties). Additionally, it specifies the API levels at which permissions were deprecated or restricted, allowing them to be mapped to the corresponding years. The usage of deprecated permissions over time for both the emulator and real device is shown in Figure 5.

**Obsolete Permissions Induce Drift.** Although some permissions were deprecated in earlier years, such as `GET_TASK`, which was deprecated in 2014, they continued to be used in subsequent years. However, the usage of this permission decreased significantly, particularly in 2019 and 2020. This decline suggests that the associated features became obsolete, contributing to changes in the feature space distribution and potentially causing drift.

Table 1: Summary of Permission Categories. (N) denotes "not for use by third-party."

| Category | Permissions | Deprecated | Restricted |
|---|---|---|---|
| Dangerous | 30 | 1 | 9 |
| Normal | 54 | 6 | - |
| Signature | 43 | 2 | - |
| (N). Third-Party | 39 | 1 | - |
| **Total** | **166** | **10** | **9** |

**Malware Exploits Restricted Permissions.** To investigate the usage of protection level types by malware, we categorized benign and malicious applications based on their use of permissions. Malware tends to use permissions more extensively than benign applications. However, certain permissions–such as those classified as *"not for use by third parties"*–show notable usage by malware in the emulator dataset. These permissions are generally restricted to pre-installed apps and require the requesting app to share the same digital signature as the app granting the permission. The ability of malware to access these permissions indicates potential exploitation of pre-installed apps and warrants further investigation. Figure 6 shows that malware frequently exploits dangerous permissions, highlighting the importance of monitoring these permissions for security.

After analyzing the dataset, we can summarize the number of permissions present and how their usage has changed over time. Table 1 summarizes the classification of permissions based on their metadata, including their protection levels, number of deprecations, and restriction levels for 166 permissions in the dataset.

**Evolving Permissions Challenge Models.** Per the official Android website, the total number of permissions reached 323 at the time of writing this paper. However, our dataset covers the period from 2008 to 2020, with only 166 permissions utilized up to API level 30. The new permissions introduced after 2021, as illustrated in Table 2, highlight the evolving nature of Android's security policies. These emerging permissions introduce previously unseen features that were absent in earlier years, posing a significant challenge for machine learning models trained on historical data. Since the model has not encountered these permissions before, its decision boundaries may become less reliable, leading to performance degradation and concept drift.

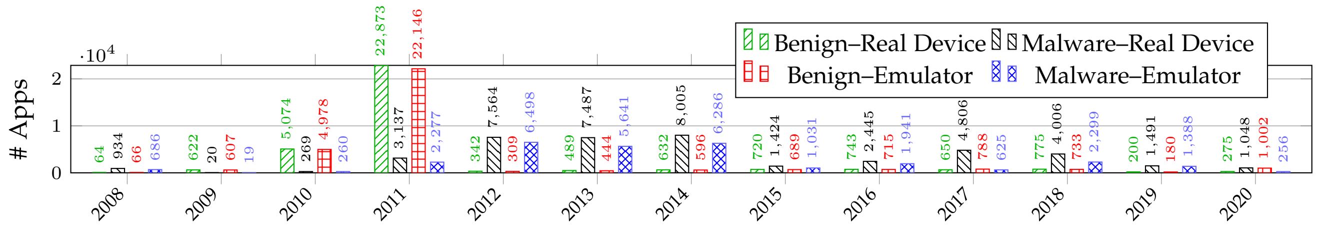


Figure 4: KronoDroid dataset distribution of both malware and benign samples across different years.

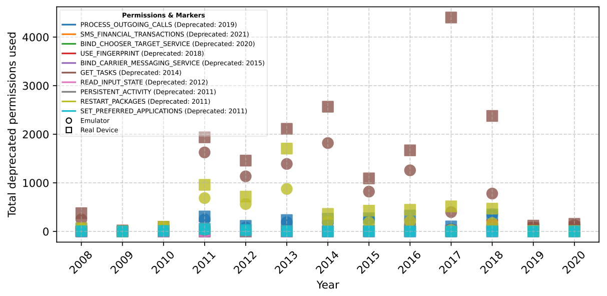


Figure 5: Deprecated permissions over time–the legend with the deprecation years for real and emulator datasets.

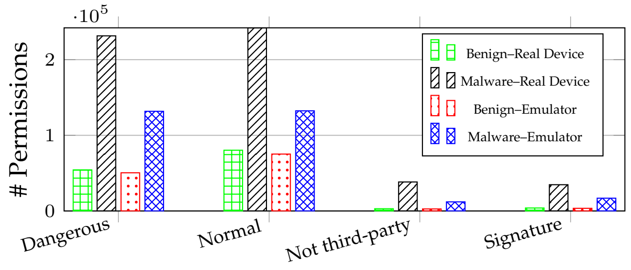


Figure 6: Protection level usage: malware vs. benign.

## 3.4 Evaluation Metrics

**Models Performance.** We utilize standard classification metrics to evaluate various aspects of model performance. These metrics are used to evaluate the effectiveness of the detection model and monitor performance changes when the model encounters concept drift. The key evaluation metrics are as follows. (1) **Precision:** Measures the accuracy of correctly classified malware apps: Precision $= \frac{\sum_{i=1}^{K} \mathrm{TP}_i}{\sum_{i=1}^{K} (\mathrm{TP}_i + \mathrm{FP}_i)}$, where $\mathrm{TP}_i$ and $\mathrm{FP}_i$ are the true and false positives for class $i$, respectively, and $K$ is the number of classes. (2) **Recall:** Represents the proportion of actual malware apps correctly classified: Recall $= \frac{\sum_{i=1}^{K} \mathrm{TP}_i}{\sum_{i=1}^{K} (\mathrm{TP}_i + \mathrm{FN}_i)}$, where $\mathrm{FN}_i$ is the false negatives for class $i$. (3) **Accuracy:** Defines the total number of correct predictions out of all predictions: Accuracy $= \frac{\sum_{i=1}^{n} \mathbf{1}(y_i = \hat{y}_i)}{n}$, where $y_i$ and $\hat{y}_i$ are the true and predicted labels, and $n$ is the total number of samples. (4) **F1 Score:** The harmonic mean of precision and recall: F1 score $= \frac{2 \cdot \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$.

**Kolmogorov-Smirnov (KS) Test.** Specific to concept drift, we use the KS test, a non-parametric statistical test used to compare the distributions of two datasets and determine whether they originate from the same underlying probability distribution. KS is particularly useful in detecting concept drift by measuring discrepancies between two empirical distributions. Given two cumulative distribution functions (CDFs), $F_1(X)$ and $F_2(X)$, the KS statistic is defined as $D_{n,m} = \sup_x |F_1(x) - F_2(x)|$ where $D_{n,m}$ is the KS statistic (maximum difference between the two distributions), $F_1(x)$ is the empirical CDF of the first dataset (e.g., past data), $F_2(x)$ is the empirical CDF of the second dataset (e.g., new data), and $\sup_x$ denotes the supremum (i.e., the greatest absolute difference over all $x$). For two samples of sizes $n$ and $m$, the null hypothesis $H_0$ states that both datasets come from the same distribution. The critical value for rejecting $H_0$ at a given significance level $\alpha$ is given by $D_\alpha = c(\alpha)\sqrt{\frac{n+m}{nm}}$, where $c(\alpha)$ is a constant determined by the significance level $\alpha$, and obtained from statistical tables.

Table 2: Protection levels of permissions across Android API levels and years. **Dang**erous, **Int**ernal, **Norm**al, **Sig**nature, Not for use by 3rd party (¬TP), and total are the metrics.

| Year (API) | Dang. | Int | Norm | ¬TP | Sig | Total |
|---|---|---|---|---|---|---|
| 2008 (1) | 17 | 0 | 26 | 25 | 6 | 74 |
| 2009 (2) | 2 | 0 | 0 | 2 | 0 | 4 |
| 2010 (8) | 1 | 0 | 1 | 1 | 2 | 5 |
| 2011 (11) | 0 | 0 | 0 | 0 | 1 | 1 |
| 2012 (16) | 3 | 0 | 0 | 0 | 1 | 4 |
| 2013 (18) | 2 | 0 | 0 | 2 | 1 | 5 |
| 2014 (20) | 1 | 0 | 0 | 0 | 0 | 1 |
| 2015 (22) | 0 | 0 | 1 | 0 | 0 | 1 |
| 2016 (24) | 0 | 0 | 1 | 0 | 3 | 4 |
| 2017 (26) | 2 | 0 | 4 | 0 | 3 | 9 |
| 2018 (28) | 1 | 0 | 3 | 0 | 0 | 4 |
| 2019 (29) | 3 | 0 | 3 | 0 | 4 | 10 |
| 2020 (30) | 0 | 0 | 5 | 0 | 3 | 8 |
| 2021 (31) | 4 | 0 | 8 | 1 | 4 | 17 |
| 2022 (32) | 0 | 0 | 1 | 0 | 0 | 1 |
| 2023 (34) | 1 | 79 | 21 | 1 | 2 | 104 |
| 2024 (35) | 0 | 6 | 4 | 2 | 0 | 12 |
| Unknown (0) | 0 | 0 | 1 | 0 | 0 | 1 |

## 3.5 Experiment Setup

**Environment.** All experiments were conducted using Google Colaboratory ("Colab"), a cloud-based platform that provides free access to computing resources, including GPU acceleration. Colab supports writing and executing Python code directly in a web browser. The experiments were run using both the cloud-hosted and local runtimes, with the default Colab configuration and no manual modifications. The *scikit-learn* library was used to build machine learning models, generate classification reports, and create confusion matrices. For deep learning models, we used *Keras*, a high-level neural network API running on top of TensorFlow.

**Experiments Design.** Each model was evaluated using different subsets of features selected from the Kronodroid dataset. These subsets included all permissions (a total of 166), as well as subsets that excluded permissions based on their protection levels–specifically deprecated, restricted, and not-for-use-by-third-party permissions.

**Parameters.** The default configurations were used for RF, with the *random state* parameter set to 42. For deep learning algorithms, we used binary cross-entropy as the loss function, Adam optimizer with a default learning rate of 0.001, an early stopping criterion based on minimum validation loss, 15 training epochs, a batch size of 15, and a validation split of 0.10. Accuracy and F1 score were used as performance metrics to evaluate the generated models.

**Architecture.** We use these deep learning architectures:

- **CNN.** Sequential input shape and two convolutional layers with 64 and 128 filters, each followed by ReLU activation and max pooling layers. This is followed by a dense layer with 128 units, also using ReLU activation, and a dropout layer with a dropout rate of 0.2. The network concludes with a final output layer that features a single sigmoid unit for binary classification.
- **RNN.** The input data was sequential, and the model included a single RNN layer with 8 units, followed by a flattening layer. A dense layer with 128 units and a dropout layer with a dropout rate of 0.2 were then added, leading to a final output layer with a single sigmoid unit. ReLU activation was used for the hidden layer, and sigmoid activation for the output.

## 4 Results and Discussion

### 4.1 Ignoring the Chronological Context

In these experiments, we ignored temporal factors and trained and tested on the entire dataset, following a common practice in the literature. We use these results as a baseline to underscore the importance of correctly accounting for concept drift. For machine learning models, the dataset was split into 80% for training and 20% for testing. The same split was applied to the deep learning models, with an additional 10% of the training set reserved for validation.

The results in Table 3 compare the performance of RF, CNN, and RNN on data collected from both a real device and an emulator. These classifiers were evaluated using 166 permissions as features, with precision, recall, and F1 score reported separately for malware and benign samples, and accuracy used as an overall metric. Although both datasets (real and emulator) used the same set of permissions, there were minor differences in precision, recall, and F1 scores. RF consistently achieved the best overall performance across both datasets, while CNN and RNN showed slightly lower performance, particularly on the emulator dataset, where they exhibited reduced recall for malware samples.

**Takeaway**

While static permissions are reliable, the source of the dataset can still influence the detection outcomes.

For the exclusion strategy, the permission protection levels (*deprecated*, *restricted*, and *not used by a third party*) slightly affected the performance of detection models differently. For **deprecated permissions (ED)**, their exclusion has minimal effects on RF models, with the accuracy remaining stable (e.g., a slight drop from 0.954 to 0.949 in the real dataset and from 0.948 to 0.946 in the emulator dataset). While RNN results have a similar drop in accuracy as RF, CNN accuracy increased from 0.944 to 0.947 in the real dataset.

The exclusion of the **restricted permissions (ER)** results in a significant decrease in performance across all models and datasets. In the real dataset, the RF accuracy dropped from 0.954 to 0.933, CNN from 0.947 to 0.933, and RNN from 0.940 to 0.924. The decline is even more pronounced in the emulator dataset, where RF accuracy decreased from 0.948 to 0.932, CNN from 0.944 to 0.931, and RNN from 0.943 to 0.924–a similar effect is observed with the emulator. These results highlight that the restricted permissions contain essential features that boost the models' accuracy.

For **not-for-third-party permissions (EN)**, their exclusion has minimal effects across models. In the real dataset, the RF and RNN accuracy remained nearly unchanged (RF: 0.954 to 0.949 and RNN: 0.943 to 0.941), while CNN improved from 0.944 to 0.946. In the emulator dataset, the RF accuracy decreased slightly from 0.948 to 0.945, while the CNN and RNN accuracy experienced a marginal decrease from 0.946 to 0.944 and from 0.941 to 0.937, respectively. This indicates that excluding not-for-third-party permissions has a limited role in influencing model performance.

**Takeaway**

Restricted permissions have a significant impact on model performance, with their exclusion causing notable drops in accuracy and F1 scores across all models and datasets—indicating their importance in distinguishing benign from malicious behavior. In contrast, removing deprecated or third-party-only permissions has minimal effect, suggesting they contribute redundant or less relevant features.

### 4.2 Year-to-Year Strategy

#### 4.2.1 Performance of Models Without KS Test.

To investigate factors of concept drift that affect the performance of the models, we define the following criteria:

① **Accuracy and F1 score drift.** Based on accuracy and F1 score for all models that ignored the temporal factor, we calculated their averages for both accuracy and F1 score with all permissions, as presented in Tables 3 and considered as baseline. Any value greater than 0.50 and less than 0.94 is considered to exhibit *drift* in the testing year and is highlighted in red on the heatmap figures.

② **For balancing results.** To evaluate the effect of balance on the performance of the models, we considered any value less than 0.50 to require improvement (*poor performance*). These values are highlighted in yellow to monitor enhancements in the results after balancing.

③ **Outperforming the baseline.** Any result greater than 0.94 is highlighted in green, indicating that it *exceeds* the performance of the baseline.

**Heatmaps For Emulator and Real Datasets.** For each model, we generated heatmaps for the real and emulator data, both before and after balancing, as shown in Figure 7. In addition, we summarize these heatmap results in tables for each model (RF, CNN, and RNN) presented in Table 4. The main observations of the results are as follows.

① **Pre-balancing.** RF models on both real and emulator datasets show a significant number of drift accuracy in

Table 3: Performance of various models on real devices and emulators. (All) All permissions: (E) Exclude, (D) deprecated, (R) Restricted, (N) Not for use by third party.

| **Model** | **Real Device** | | | | | | | **Emulator** | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Benign | | | Malware | | | **Acc** | Benign | | | Malware | | | **Acc** |
| | **P** | **R** | **F1** | **P** | **R** | **F1** | | **P** | **R** | **F1** | **P** | **R** | **F1** | |
| RF-All | 0.943 | 0.959 | 0.951 | 0.964 | 0.949 | 0.956 | 0.954 | 0.942 | 0.965 | 0.953 | 0.956 | 0.927 | 0.941 | 0.948 |
| RF-ED | 0.938 | 0.956 | 0.947 | 0.960 | 0.943 | 0.951 | 0.949 | 0.939 | 0.963 | 0.951 | 0.954 | 0.924 | 0.939 | 0.946 |
| RF-ER | 0.919 | 0.947 | 0.933 | 0.952 | 0.925 | 0.938 | 0.936 | 0.924 | 0.954 | 0.939 | 0.943 | 0.905 | 0.924 | 0.932 |
| RF-EN | 0.937 | 0.957 | 0.947 | 0.960 | 0.943 | 0.951 | 0.949 | 0.939 | 0.962 | 0.950 | 0.953 | 0.925 | 0.939 | 0.945 |
| CNN-All | 0.933 | 0.949 | 0.941 | 0.954 | 0.939 | 0.947 | 0.944 | 0.939 | 0.964 | 0.951 | 0.954 | 0.924 | 0.939 | 0.946 |
| CNN-ED | 0.936 | 0.953 | 0.945 | 0.957 | 0.942 | 0.949 | 0.947 | 0.951 | 0.941 | 0.946 | 0.929 | 0.941 | 0.935 | 0.941 |
| CNN-ER | 0.913 | 0.944 | 0.928 | 0.948 | 0.919 | 0.933 | 0.931 | 0.923 | 0.934 | 0.928 | 0.919 | 0.905 | 0.912 | 0.921 |
| CNN-EN | 0.938 | 0.948 | 0.943 | 0.953 | 0.943 | 0.948 | 0.946 | 0.936 | 0.957 | 0.946 | 0.946 | 0.921 | 0.934 | 0.941 |
| RNN-All | 0.914 | 0.968 | 0.940 | 0.971 | 0.920 | 0.944 | 0.943 | 0.935 | 0.960 | 0.947 | 0.949 | 0.917 | 0.933 | 0.941 |
| RNN-ED | 0.931 | 0.941 | 0.936 | 0.948 | 0.939 | 0.944 | 0.940 | 0.937 | 0.953 | 0.945 | 0.942 | 0.923 | 0.932 | 0.939 |
| RNN-ER | 0.929 | 0.906 | 0.917 | 0.920 | 0.940 | 0.930 | 0.924 | 0.927 | 0.934 | 0.931 | 0.920 | 0.912 | 0.916 | 0.924 |
| RNN-EN | 0.928 | 0.948 | 0.938 | 0.953 | 0.936 | 0.944 | 0.941 | 0.930 | 0.956 | 0.943 | 0.946 | 0.914 | 0.930 | 0.937 |

heatmap cells (100+ in most cases), highlighting substantial concept drift in accuracy over many testing years. However, RF demonstrates slightly less drift in accuracy compared to CNN (e.g., 108 for the real dataset and 111 for the emulator dataset). RNN follows a similar trend, and RF generally has fewer drifts than RNN. In contrast, the count of poor accuracy results, representing values below 0.50, is relatively high, particularly for the emulator dataset, which reached 47 with RF after excluding restricted permissions, while CNN and RNN achieved 28 and 29, respectively. For exceeding baseline accuracy, the models achieved more than 20 results in some cases, but the impact of balancing significantly reduced this number by approximately half, revealing the strong influence of class imbalance on performance. Excluding D and R features reduced the drift in accuracy for RF compared to using all permissions, while a decrease was observed in CNN for both real and emulator datasets, and only in the real dataset for RNN. In contrast, excluding N permissions reduced the accuracy drift only in the RNN.

**Takeaway**

Excluding D, R, and N permissions generally improves performance stability across years. However, this can come at the cost of reduced accuracy, particularly on emulator datasets, highlighting a trade-off between stability and performance. Moreover, fewer instances surpass baseline performance after exclusion, suggesting limited gains in certain scenarios.

② **Post-Balancing.** The results of the models' performance post-balancing show that balancing the datasets positively impacts the F1 scores across all models (RF, CNN, and RNN). This improvement highlights that addressing the imbalance of data before analyzing the drift is crucial to accurately assess the behavior of the models. The increase in F1 scores indicates that the models benefit from a more representative distribution of benign and malicious samples, which enhances their ability to generalize over testing years. Moreover, the reduction in poor performance in accuracy is clear, which indicates that model performance improved when compared with pre-balance. However, the increase in drift count in accuracy, especially for CNN and RNN, reflects the fundamental reality of concept drift. This indicates that balancing the dataset improves model performance and reveals the extent of concept drift previously obscured by imbalance. Once this imbalance is addressed, the models' results reveal more accurate trends, highlighting the impact of drift over time.

**Takeaway**

Balancing the datasets before addressing the concept drift is critical to revealing the drift's extent and mitigating its impact.

Focusing on the effect of excluding D, R, and N permissions, the RF results demonstrate resilience to concept drift, with fewer increases in the number of drifts compared to CNN and RNN. Excluding D for RF with real is not affected, and concept drift occurred with the emulator in one new year, similar to excluding R (i.e., an increase from 145 to 146 poor performance instances). However, the drift is reduced when N is excluded–by one year in real and two in the emulator. CNN is the most sensitive to the exclusion of permissions, specifically D and R, where the poor accuracy values increase significantly. RNN shows a moderate response to the exclusion scenarios. Excluding D and R permissions increases concept drift, especially in emulator datasets. However, RNN still achieves improvements in the post-balance F1 scores for the real dataset with all permissions. Additionally, deep learning models show less poor accuracy and higher F1 scores after balancing.

**Takeaway**

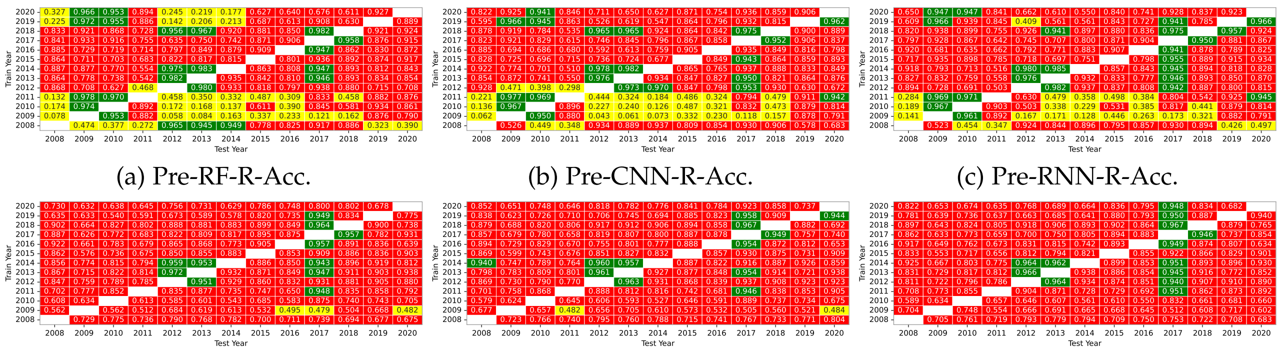
Although deprecated or restricted permissions may seem outdated, they remain relevant for many applications and aid malware detection. Results show that excluding them increases concept drift in CNN and RNN models, highlighting their continued real-world predictive value.

### *4.2.2 Concept Drift Detection Using KS Test*

In this set of experiments, we aim to detect the presence of concept drift using the Kolmogorov-Smirnov (KS) test. We employ the KS test to analyze feature distributions across different time periods, datasets (real vs. emulator), and permission configurations (e.g., exclusion of deprecated, restricted, or not-for-use-by-third-party permissions).

The null hypothesis ($H_0$) proposes that there is no significant difference in the accuracy and F1 score distributions between the two years being compared. If the $p$-value resulting from the KS test is below a chosen significance threshold ($p \leq 0.05$), we reject the null hypothesis, indicating that a statistically significant concept drift has occurred.

We conducted a large number of experiments, totaling 32 heatmap results for each model and Figure 8 illustrates an example of the results obtained for the three classification



(a) Pre-RF-R-Acc. (b) Pre-CNN-R-Acc. (c) Pre-RNN-R-Acc.

(d) Post-RF-R-Acc. (e) Post-CNN-R-Acc. (f) Post-RNN-R-Acc.

Figure 7: Accuracy performance of various models pre/post balancing with all permissions. (R) real device dataset. Cells with red color indicate decreased accuracy, yellow represent values less than 0.50, and green show improvements.

Table 4: RF, CNN, and RNN accuracy and F1 scores for different configurations of permission sets with a comparison to the baseline. Red indicates the count of testing years with decreased accuracy, yellow represents values less than 0.50, and green shows the number of years improved. (D) Deprecated, (R) Restricted, and (N) Not for use by third party.

| Balance | Accuracy | | | F1 score | | |
|---|---|---|---|---|---|---|
| | Red | Yellow | Green | Red | Yellow | Green |
| | | | **RF - Random Forest** | | | |
| Pre | 100 | 34 | 22 | 111 | 44 | 1 |
| Pre (Excl.) | 99 (D) / 95 (R) / 101 (N) | 35 (D) / 40 (R) / 34 (N) | 22 (D) / 21 (R) / 21 (N) | 109 (D) / 104 (R) / 112 (N) | 45 (D) / 51 (R) / 42 (N) | 2 (D) / 1 (R) / 2 (N) |
| Post | 142 | 3 | 11 | 136 | 9 | 11 |
| Post (Excl.) | 142 (D) / 146 (R) / 141 (N) | 4 (D) / 4 (R) / 5 (N) | 10 (D) / 6 (R) / 10 (N) | 135 (D) / 136 (R) / 135 (N) | 11 (D) / 14 (R) / 11 (N) | 10 (D) / 6 (R) / 10 (N) |
| | | | **CNN - Convolutional Neural Network** | | | |
| Pre | 108 | 27 | 21 | 111 | 44 | 1 |
| Pre (Excl.) | 112 (D) / 109 (R) / 110 (N) | 24 (D) / 29 (R) / 23 (N) | 20 (D) / 18 (R) / 23 (N) | 110 (D) / 113 (R) / 120 (N) | 44 (D) / 42 (R) / 34 (N) | 2 (D) / 1 (R) / 2 (N) |
| Post | 142 | 2 | 12 | 136 | 8 | 12 |
| Post (Excl.) | 146 (D) / 146 (R) / 143 (N) | 0 (D) / 1 (R) / 1 (N) | 10 (D) / 9 (R) / 12 (N) | 142 (D) / 140 (R) / 139 (N) | 4 (D) / 7 (R) / 5 (N) | 10 (D) / 9 (R) / 12 (N) |
| | | | **RNN - Recurrent Neural Network** | | | |
| Pre | 110 | 23 | 23 | 116 | 38 | 2 |
| Pre (Excl.) | 107 (D) / 108 (R) / 108 (N) | 28 (D) / 27 (R) / 26 (N) | 21 (D) / 21 (R) / 22 (N) | 117 (D) / 112 (R) / 114 (N) | 38 (D) / 42 (R) / 40 (N) | 1 (D) / 2 (R) / 2 (N) |
| Post | 143 | 0 | 13 | 140 | 3 | 13 |
| Post (Excl.) | 142 (D) / 147 (R) / 144 (N) | 1 (D) / 2 (R) / 0 (N) | 13 (D) / 7 (R) / 12 (N) | 140 (D) / 141 (R) / 139 (N) | 3 (D) / 8 (R) / 5 (N) | 13 (D) / 7 (R) / 12 (N) |

models (RF, CNN, and RNN) before and after balancing, using both the real and emulator datasets. Each heatmap represents the $p$-values of the KS tests applied to the accuracy and F1 score distributions for year-to-year comparisons under various permission configurations.

To simplify interpretation, we summarize each heatmap by counting the number of year-to-year comparisons with a $p$-value less than 0.05, indicating a statistically significant concept drift. These summarized results are presented separately for each model in Table 5.

① **Impact of Balancing.** Across all models, the number of significant $p$-values noticeably increased after balancing, indicating that balancing better captures concept drift in both accuracy and F1 scores. This suggests that balancing has a significant impact on improving the sensitivity of drift detection. For example, in RF, the number of significant $p$-values for accuracy increased from 52 to 84 (real, all permissions) and from 70 to 96 (emulator, all permissions) after balancing. For CNN, a similar trend is observed, with $p$-values increasing from 52 to 60 and from 58 to 76. For RNN, the increase was from 40 to 60 and from 48 to 64.

**Takeaway**

Balancing the dataset not only improves concept drift detection in general but also reveals variations in model sensitivity to concept drift.

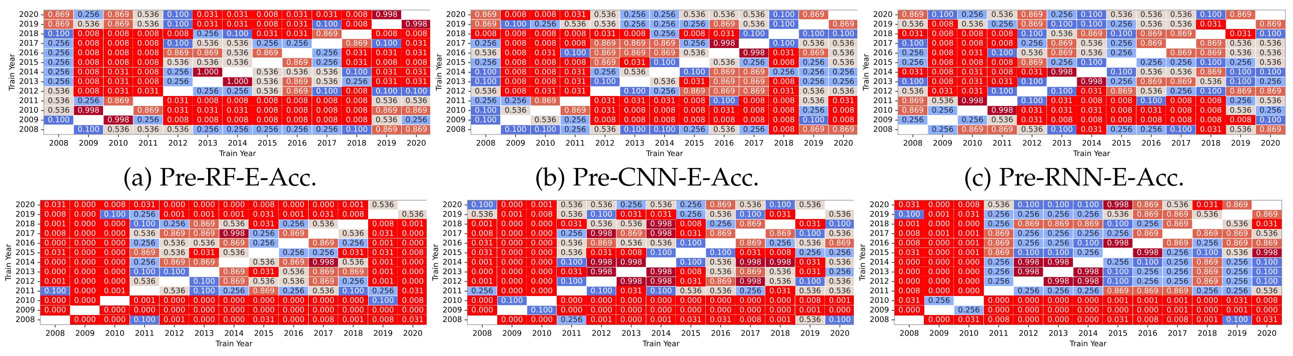
② **Dataset-Specific Observations (Real vs. Emulator).** For all models, the $p$-value counts are generally higher when using the emulator dataset compared to the real dataset. For the RF model, the significant $p$-values related to post-balancing accuracy increased to 96 for the emulator dataset, compared to 84 for the real dataset. Similarly, for the CNN and RNN models, the $p$-values for accuracy reached 76 compared to 60, and 64 compared to 60, respectively.

③ **Impact of Excluding Permissions.** Across all models, the exclusion of permissions increased concept drift, particularly after balancing. For RF 5, post-balancing accuracy $p$-values for the real dataset increased from 84 (All Permissions) to 94 (Exclude D). Furthermore, CNN and RNN showed improvements, with post-balancing accuracy $p$-values increasing from 60 to 72 for the real dataset.

We further compared the performance of the models with all permissions and after excluding D permissions in Figure 9 for RF with post-balancing on the real dataset. The results of excluding deprecated permissions lead to some, but not significant, effect on model performance across most years. For example, in 2010, the average accuracy improved from 0.660 (all permissions) to 0.665 (D excluded). In 2011, accuracy increased from 0.801 to 0.836, and in 2016, it increased from 0.815 to 0.846. However, in some years, such as 2013, the exclusion has minimal impact, with accuracy remaining nearly the same (0.878 vs. 0.879).

**Takeaway**

While excluding permissions does not have a significant effect on the model's performance, it is statistically significant in improving concept drift detection.


(a) Pre-RF-E-Acc.

(b) Pre-CNN-E-Acc.

(c) Pre-RNN-E-Acc.

(d) Post-RF-E-Acc.

(e) Post-CNN-E-Acc.

(f) Post-RNN-E-Acc.

Figure 8: Concept drift detection of various algorithms pre/post balancing, where $p$-value $\leq 0.05$ (red color). (E) emulator

Table 5: RF, CNN, and RNN model $p$-value counts ($\leq 0.05$) for accuracy and F1 score across real devices and emulators. (Pre) Pre-balanced, (Post) Post-balanced, (All) All permissions, (D) Deprecated, (R) Restricted, (N) Not for use by 3rd-party.

| Balance | Real Device | | | | Emulator | | | |
|---|---|---|---|---|---|---|---|---|
| | Accuracy ($p \leq 0.05$) | | F1 score ($p \leq 0.05$) | | Accuracy ($p \leq 0.05$) | | F1 score ($p \leq 0.05$) | |
| | All | Excl. | All | Excl. | All | Excl. | All | Excl. |
| **RF - Random Forest** | | | | | | | | |
| Pre | 52 | 58 (D) / 42 (R) / 58 (N) | 50 | 64 (D) / 50 (R) / 60 (N) | 70 | 68 (D) / 58 (R) / 68 (N) | 66 | 64 (D) / 56 (R) / 58 (N) |
| Post | 84 | 94 (D) / 92 (R) / 92 (N) | 80 | 94 (D) / 94 (R) / 92 (N) | 96 | 102 (D) / 96 (R) / 92 (N) | 96 | 102 (D) / 98 (R) / 94 (N) |
| **CNN - Convolutional Neural Network** | | | | | | | | |
| Pre | 52 | 52 (D) / 44 (R) / 52 (N) | 46 | 38 (D) / 44 (R) / 34 (N) | 58 | 52 (D) / 54 (R) / 58 (N) | 60 | 62 (D) / 48 (R) / 50 (N) |
| Post | 60 | 72 (D) / 68 (R) / 58 (N) | 62 | 76 (D) / 70 (R) / 54 (N) | 76 | 70 (D) / 72 (R) / 64 (N) | 78 | 70 (D) / 72 (R) / 62 (N) |
| **RNN - Recurrent Neural Network** | | | | | | | | |
| Pre | 40 | 36 (D) / 42 (R) / 54 (N) | 36 | 38 (D) / 40 (R) / 42 (N) | 48 | 42 (D) / 40 (R) / 52 (N) | 42 | 38 (D) / 46 (R) / 42 (N) |
| Post | 60 | 72 (D) / 68 (R) / 64 (N) | 60 | 72 (D) / 68 (R) / 62 (N) | 64 | 60 (D) / 76 (R) / 58 (N) | 64 | 60 (D) / 78 (R) / 58 (N) |

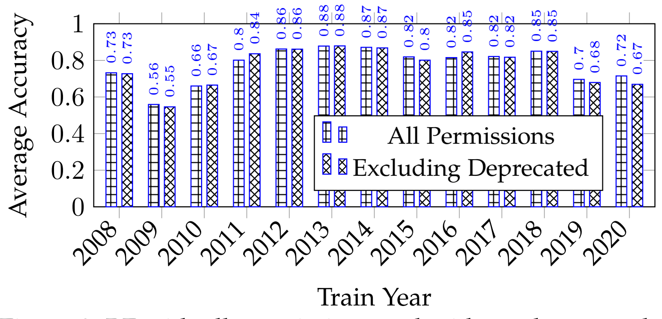


Figure 9: RF with all permissions and without deprecated.

### 4.3 Discussion

Our results provide strong empirical evidence that concept drift is prevalent in Android malware detection and is influenced by multiple factors. Below, we highlight those results by answering the research questions in section 3.

**AQ-1:** The findings indicate that permissions play a significant role in both the accuracy and effectiveness of malware detection across all models. Specifically, as shown in the results for the real and emulator datasets in Table 3, the RF model achieved an accuracy of 0.954 on the real dataset using only 166 permissions out of 489 dynamic and static features. When using these hybrid features, the models achieved accuracies of 0.980 (RF), 0.970 (CNN), and 0.980 (RNN). These results underscore the critical importance of permissions in Android malware detection.

**AQ-2:** The sensitivity of permissions to concept drift significantly impacts the temporal performance of Android malware detection models. The results reveal that permissions in the emulator dataset are more prone to drift, as evidenced by fluctuations in accuracy and consistently higher $p$-value counts in this dataset compared to the real device dataset.

Drift became more pronounced after data balancing, as observed across all models. Although accuracy improved in low-performing cases, this improvement also highlights the heightened sensitivity of permissions to distributional changes. For example, accuracy (yellow) increased from 47 to 8 after balancing and excluding restricted permissions in the emulator dataset Table 4. While this is considered a performance improvement, it also coincides with an increase in detected concept drift, which rose from 96 to 146. Additionally, for the same case, the number of significant $p$-values increased from 58 to 96, as shown in Table 5.

**AQ-3:** The exclusion strategy revealed a marginal impact on the models when *deprecated* and *not for use by third-party* permissions were omitted. In some cases, such as with the CNN model, performance metrics like accuracy and F1 score showed slight improvements, suggesting that these permissions—while historically valuable—may contribute less to a model's ability to generalize in certain scenarios.

From the perspective of concept drift, the exclusion of these permissions played a significant role in enhancing the detection of drift patterns. Using the KS test or heatmap cells to highlight drift, the results indicated that removing these permissions made concept drift more apparent. This can be attributed to the elimination of outdated or less relevant features, which appeared in certain years but were absent in others, as shown in the analysis section (Figure 5).

Thus, while excluding *deprecated* and *not-for-use-by-third-party* permissions might not always boost immediate model performance, it contributes to the understanding and detection of concept drift–an essential factor for the long-term stability and reliability of malware detection systems.

## 5 Related Work

The Android permission system plays a central role in both malware detection and platform security. In the following, we review work on its evolution, abuse, and implications for machine learning-based detection.

**Android Permission System.** Since API 6.0, Android's permission model has undergone several changes, expanding to support hardware rather than refining security granularity [35]. Studies reveal overprivileged apps—especially pre-installed ones—contributing to user confusion and increased risk [36]. Tools like PScout [37] and Cusper [38] exposed structural flaws, while fine-grained enforcement via frameworks like Sorbet [39] and DroidCap [40] sought to address least-privilege enforcement. Recent efforts also uncovered issues with undocumented permissions [41] and inter-version inconsistencies [35], highlighting challenges in sustaining a secure permission framework.

**Privilege Escalation via Permission Abuse.** The permission system has been exploited through custom permission spoofing [42], exposed components for sensor access [43], covert collusion channels [44], and physical hijacking [45]. Tools like PmDroid [46] detect permission violations by ad libraries, while others reveal permissionless exfiltration paths via shared resources [47]. Detection techniques span static (e.g., COVERT [48]), dynamic (e.g., VetDroid [49]), and hybrid approaches (e.g., Permlyzer [50]), reinforcing the need for multi-pronged analysis and mitigation.

**Runtime Permission Issues and Testing.** The shift to runtime permissions introduced new vulnerabilities. Tools such as RevDroid [51] and Aper [52] analyzed app behavior post-revocation, showing malware often handles revocation more gracefully than benign apps. Testing frameworks like PermDroid [53], PATDroid [54], and DPC [55] automate state-based testing or introduce dynamic controls to restrict third-party library misuse.

**Android Malware Detection.** Permissions are a critical feature in Android malware detection. Numerous works highlight their utility: Ilham *et al.* [6] and Shatnawi *et al.* [10] showed high accuracy with selected permissions using traditional classifiers. Sahin *et al.* [9] demonstrated the benefit of reducing permission sets, while Kato *et al.* [7] introduced composition ratios to classify malware. Recent models combine permissions with APIs, intents, and behavior traces. Tools such as PermPair [56], DroidXGB [57], and MalPat [58] enhance classification by mining permission patterns and graphs. Multi-feature models, including CNNs with opcode and API information [11], improve zero-day detection. Federated learning approaches like FEDroid [12] address privacy-preserving malware detection using permission-based features. Permissions also support behavior-specific multi-label classification frameworks [59]. These studies reinforce the critical role of permissions in malware detection. However, their effectiveness is susceptible to changes in permission availability and semantics over time, introducing a potential source of concept drift.

**Concept Drift.** To improve model adaptability under drift, approaches like TRANSCEND [60], TRANSCENDENT [61], and DREAM [62] incorporate conformal prediction, rejection mechanisms, or semi-supervised learning. Active learning and retraining techniques [19], [20], [63] reduce labeling cost and improve drift responsiveness. Other models use self-training [20], contrastive learning [64], and ensemble classifiers with adaptive feature selection [18].

Novel architectures include cluster-based drift detection [65], GNNs for invariant feature learning [66], and hybrid models enhanced with evolutionary strategies [67]. Recent insights from Chow *et al.* [21] show that drift stems not only from evolving malware but also goodware changes and family-specific shifts. Molina *et al.* [19] further advocate for retraining-on-drift rather than fixed intervals to conserve resources and improve stability.

**This Study.** While prior work broadly addresses evolving threats and concept drift, our study focuses on the underexplored impact of **permission deprecation**—a system-level change—on model stability and performance drift. We offer new insights into how Android's evolving permission architecture affects the reliability of ML-based malware detection. Effectively addressing concept drift is critical for sustaining detection accuracy, and this study examines the temporal evolution of permission-based features, demonstrating that even simple models are susceptible to degradation. By isolating the effects of permission deprecation and quantifying drift using accuracy and F1 scores, our findings support the design of more adaptive and resilient detection frameworks.

## 6 Conclusion

This work explored the role of Android permissions in malware detection, their sensitivity to concept drift, and the impact of deprecated and restricted permissions on model stability. Using the KronoDroid dataset (including real and emulator devices), it found permissions to be strong features for detection. Excluding deprecated/restricted permissions had minimal impact on performance and even improved accuracy in some models (e.g., CNN). Two strategies assessed whether this exclusion mitigates concept drift. Results showed improved drift detection–especially in a year-to-year setup—since deprecated permissions persisted in updated apps. Additionally, dataset balancing improved model accuracy and enhanced drift detection via the KS test.

## References

[1] —, "Manifest.permission — android developers," https://www.counterpointresearch.com/insights/global-smartphone-os-market-share, 2025.

[2] ——, "Permissions on android android developers," https://developer.android.com/guide/topics/permissions/overview, 2025.

[3] Y. Sharma and A. Arora, "A comprehensive review on permissions-based android malware detection," *Int. J. Inf. Sec.*, vol. 23, no. 3, pp. 1877–1912, 2024. [Online]. Available: https://doi.org/10.1007/s10207-024-00822-2

[4] N. Carlini, A. P. Felt, and D. Wagner, "An evaluation of the google chrome extension security architecture," in *USENIX Security Symposium*, 2012, pp. 97–111.

[5] X. Zhang, Z. Yu, X. Li, C. Zhang, C. Sun, N. Zhang, and R. H. Deng, "Understanding the bad development practices of android custom permissions in the wild," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–17, 2025.

[6] S. Ilham, A. Ghadi, and A. B. Abdelhakim, "Permission based malware detection in android devices," in *SCA*. ACM, 2018, pp. 83:1–83:6. [Online]. Available: https://doi.org/10.1145/3286606.3286860

[7] H. Kato, T. Sasaki, and I. Sasase, "Android malware detection based on composition ratio of permission pairs," *IEEE Access*, vol. 9, pp. 130 006–130 019, 2021. [Online]. Available: https://doi.org/10.1109/ACCESS.2021.3113711

[8] S. R. T. Mat, M. F. A. Razak, M. N. M. Kahar, J. M. Arif, and A. Firdaus, "A bayesian probability model for android malware detection," *ICT Express*, 2022.

[9] D. Ö. Sahin, O. E. Kural, S. Akleylek, and E. Kiliç, "A novel permission-based android malware detection system using feature selection based on linear regression," *Neural Comput. Appl.*, vol. 35, no. 7, pp. 4903–4918, 2023. [Online]. Available: https://doi.org/10.1007/s00521-021-05875-1

[10] A. S. Shatnawi, Q. Yassen, and A. A. Yateem, "An android malware detection approach based on static feature analysis using machine learning algorithms," in *ANT*, vol. 201, 2022, pp. 653–658. [Online]. Available: https://doi.org/10.1016/j.procs.2022.03.086

[11] S. Millar, N. McLaughlin, J. M. del Rincón, and P. Miller, "Multi-view deep learning for zero-day android malware detection," *J. Inf. Secur. Appl.*, vol. 58, p. 102718, 2021. [Online]. Available: https://doi.org/10.1016/j.jisa.2020.102718

[12] W. Fang, J. He, W. Li, X. Lan, Y. Chen, T. Li, J. Huang, and L. Zhang, "Comprehensive android malware detection based on federated learning architecture," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 3977–3990, 2023. [Online]. Available: https://doi.org/10.1109/TIFS.2023.3287395

[13] T. R. Hoens, R. Polikar, and N. V. Chawla, "Learning from streaming data with concept drift and imbalance: an overview," *Prog. Artif. Intell.*, vol. 1, no. 1, pp. 89–101, 2012. [Online]. Available: https://doi.org/10.1007/s13748-011-0008-0

[14] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in nonstationary environments: A survey," *IEEE Comput. Intell. Mag.*, vol. 10, no. 4, pp. 12–25, 2015. [Online]. Available: https://doi.org/10.1109/MCI.2015.2471196

[15] G. I. Webb, R. Hyde, H. Cao, H. Nguyen, and F. Petitjean, "Characterizing concept drift," *Data Min. Knowl. Discov.*, vol. 30, no. 4, pp. 964–994, 2016. [Online]. Available: https://doi.org/10.1007/s10618-015-0448-4

[16] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Wozniak, "Ensemble learning for data stream analysis: A survey," *Inf. Fusion*, vol. 37, pp. 132–156, 2017. [Online]. Available: https://doi.org/10.1016/j.inffus.2017.02.004

[17] F. Bayram, B. S. Ahmed, and A. Kassler, "From concept drift to model degradation: An overview on performance-aware drift detectors," *Knowl. Based Syst.*, vol. 245, p. 108632, 2022. [Online]. Available: https://doi.org/10.1016/j.knosys.2022.108632

[18] D. Hu, Z. Ma, X. Zhang, P. Li, D. Ye, and B. Ling, "The concept drift problem in android malware detection and its solution," *Secur. Commun. Networks*, vol. 2017, pp. 4 956 386:1–4 956 386:13, 2017. [Online]. Available: https://doi.org/10.1155/2017/4956386

[19] B. Molina-Coronado, U. Mori, A. Mendiburu, and J. Miguel-Alonso, "Efficient concept drift handling for batch android malware detection models," *Pervasive Mob. Comput.*, vol. 96, p. 101849, 2023. [Online]. Available: https://doi.org/10.1016/j.pmcj.2023.101849

[20] M. T. Alam, R. Fieblinger, A. Mahara, and N. Rastogi, "MORPH: towards automated concept drift adaptation for malware detection," *CoRR*, vol. abs/2401.12790, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2401.12790

[21] T. Chow, Z. Kan, L. Linhardt, L. Cavallaro, D. Arp, and F. Pierazzi, "Drift forensics of malware classifiers," in *AISec*. ACM, 2023, pp. 197–207. [Online]. Available: https://doi.org/10.1145/3605764.3623918

[22] G. Meng, M. Patrick, Y. Xue, Y. Liu, and J. Zhang, "Securing android app markets via modeling and predicting malware spread between markets," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 7, pp. 1944–1959, 2019. [Online]. Available: https://doi.org/10.1109/TIFS.2018.2889924

[23] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. A. Wagner, "A survey of mobile malware in the wild," in *SPSM*. ACM, 2011, pp. 3–14. [Online]. Available: https://doi.org/10.1145/2046614.2046618

[24] W. Ali, "Hybrid intelligent android malware detection using evolving support vector machine based on genetic algorithm and particle swarm optimization," *IJCSNS*, vol. 19, no. 9, p. 15, 2019.

[25] A. Vishnoi, P. Mishra, C. Negi, and S. K. Peddoju, "Android malware detection techniques in traditional and cloud computing platforms: A state-of-the-art survey," *Int. J. Cloud Appl. Comput.*, vol. 11, no. 4, pp. 113–135, 2021. [Online]. Available: https://doi.org/10.4018/IJCAC.2021100107

[26] J. Gamba, Á. Feal, E. Blázquez, V. Bandara, A. Razaghpanah, J. Tapiador, and N. Vallina-Rodriguez, "Mules and permission laundering in android: Dissecting custom permissions in the wild," *IEEE Trans. Dependable Secur. Comput.*, vol. 21, no. 4, pp. 1801–1816, 2024. [Online]. Available: https://doi.org/10.1109/TDSC.2023.3288981

[27] J. C. Schlimmer and R. H. Granger, "Incremental learning from noisy data," *Mach. Learn.*, vol. 1, no. 3, pp. 317–354, 1986. [Online]. Available: https://doi.org/10.1023/A:1022810614389

[28] Q. Xiang, L. Zi, X. Cong, and Y. Wang, "Concept drift adaptation methods under the deep learning framework: A literature review," *Applied Sciences*, vol. 13, no. 11, p. 6515, 2023.

[29] —, "Activitymanager — api reference — android developers," https://developer.android.com/reference/android/app/ActivityManager, 3 2025.

[30] ——, "Privacy changes in android 10 — android developers," https://developer.android.com/about/versions/10/privacy/changes, 03 2025.

[31] J. Gama, I. Zliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, pp. 44:1–44:37, 2014. [Online]. Available: https://doi.org/10.1145/2523813

[32] Z. Wang and W. Wang, "Concept drift detection based on kolmogorov–smirnov test," in *Artificial Intelligence in China*. Singapore: Springer Singapore, 2020, pp. 273–280.

[33] 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, p. 102399, 2021. [Online]. Available: https://doi.org/10.1016/j.cose.2021.102399

[34] —, "Manifest.permission — android developers," https://developer.android.com/reference/android/Manifest.permission, 2025.

[35] Y. Zhauniarovich and O. Gadyatskaya, "Small changes, big changes: An updated view on the android permission system," in *RAID*, vol. 9854. Springer, 2016, pp. 346–367. [Online]. Available: https://doi.org/10.1007/978-3-319-45719-2_16

[36] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, "Permission evolution in the android ecosystem," in *ACSAC*. ACM, 2012, pp. 31–40. [Online]. Available: https://doi.org/10.1145/2420950.2420956

[37] K. W. Y. Au, Y. F. Zhou, Z. Huang, and D. Lie, "Pscout: analyzing the android permission specification," in *CCS*. ACM, 2012, pp. 217–228. [Online]. Available: https://doi.org/10.1145/2382196.2382222

[38] G. S. Tuncay, S. Demetriou, K. Ganju, and C. A. Gunter, "Resolving the predicament of android custom permissions," in *NDSS*. The Internet Society, 2018. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_08-4_Tuncay_paper.pdf

[39] E. Fragkaki, L. Bauer, L. Jia, and D. Swasey, "Modeling and enhancing android's permission system," in *ESORICS*, vol. 7459. Springer, 2012, pp. 1–18. [Online]. Available: https://doi.org/10.1007/978-3-642-33167-1_1

[40] A. Dawoud and S. Bugiel, "Droidcap: OS support for capability-based permissions in android," in *NDSS*. The Internet Society, 2019. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/droidcap-os-support-for-capability-based-permissions-in-android/

[41] H. Zhou, H. Wang, S. Wu, X. Luo, Y. Zhou, T. Chen, and T. Wang, "Finding the missing piece: Permission specification analysis for android NDK," in *ASE*. IEEE, 2021, pp. 505–516. [Online]. Available: https://doi.org/10.1109/ASE51524.2021.9678843

[42] R. Li, W. Diao, Z. Li, S. Yang, S. Li, and S. Guo, "Android custom permissions demystified: A comprehensive security evaluation," *IEEE Trans. Software Eng.*, vol. 48, no. 11, pp. 4465–4484, 2022. [Online]. Available: https://doi.org/10.1109/TSE.2021.3119980

[43] A. Aldoseri, D. F. Oswald, and R. Chiper, "A tale of four gates - privilege escalation and permission bypasses on android through app components," in *ESORICS*, vol. 13555. Springer, 2022, pp. 233–251. [Online]. Available: https://doi.org/10.1007/978-3-031-17146-8_12

[44] J. Reardon, Á. Feal, P. Wijesekera, A. E. B. On, N. Vallina-Rodriguez, and S. Egelman, "50 ways to leak your data: An exploration of apps' circumvention of the android permissions system," *login Usenix Mag.*, vol. 44, no. 4, 2019. [Online]. Available: https://www.usenix.org/publications/login/winter2019/reardon

[45] X. Wang, S. Shi, Y. Chen, and W. C. Lau, "Phyjacking: Physical input hijacking for zero-permission authorization attacks on

android," in *NDSS*. The Internet Society, 2022. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/auto-draft-187/

[46] X. Gao, D. Liu, H. Wang, and K. Sun, "Pmdroid: Permission supervision for android advertising," in *SRDS*. IEEE Computer Society, 2015, pp. 120–129. [Online]. Available: https://doi.org/10.1109/SRDS.2015.41

[47] J. Wu, Y. Wu, M. Yang, Z. Wu, T. Luo, and Y. Wang, "POSTER: bitheft: Stealing your secrets by bidirectional covert channel communication with zero-permission android application," in *SIGSAC*. ACM, 2015, pp. 1690–1692. [Online]. Available: https://doi.org/10.1145/2810103.2810108

[48] H. Bagheri, A. Sadeghi, J. Garcia, and S. Malek, "COVERT: compositional analysis of android inter-app permission leakage," *IEEE Trans. Software Eng.*, vol. 41, no. 9, pp. 866–886, 2015. [Online]. Available: https://doi.org/10.1109/TSE.2015.2419611

[49] Y. Zhang, M. Yang, Z. Yang, G. Gu, P. Ning, and B. Zang, "Permission use analysis for vetting undesirable behaviors in android apps," *IEEE Trans. Inf. Forensics Secur.*, vol. 9, no. 11, pp. 1828–1842, 2014. [Online]. Available: https://doi.org/10.1109/TIFS.2014.2347206

[50] W. Xu, F. Zhang, and S. Zhu, "Permlyzer: Analyzing permission usage in android applications," in *ISSRE*. IEEE Computer Society, 2013, pp. 400–410. [Online]. Available: https://doi.org/10.1109/ISSRE.2013.6698893

[51] Z. Fang, W. Han, D. Li, Z. Guo, D. Guo, X. S. Wang, Z. Qian, and H. Chen, "revdroid: Code analysis of the side effects after dynamic permission revocation of android apps," in *AsiaCCS*. ACM, 2016, pp. 747–758. [Online]. Available: https://doi.org/10.1145/2897845.2897914

[52] S. Wang, Y. Wang, X. Zhan, Y. Wang, Y. Liu, X. Luo, and S. Cheung, "APER: evolution-aware runtime permission misuse detection for android apps," in *ICSE*. ACM, 2022, pp. 125–137. [Online]. Available: https://doi.org/10.1145/3510003.3510074

[53] S. Yang, Z. Zeng, and W. Song, "Permdroid: automatically testing permission-related behaviour of android applications," in *ISSTA*. ACM, 2022, pp. 593–604. [Online]. Available: https://doi.org/10.1145/3533767.3534221

[54] A. Sadeghi, R. Jabbarvand, and S. Malek, "Patdroid: permission-aware GUI testing of android," in *ESEC/FSE*. ACM, 2017, pp. 220–232. [Online]. Available: https://doi.org/10.1145/3106237.3106250

[55] F. Hsu, N. Liu, Y. Hwang, C. Liu, C. Wang, and C. Chen, "DPC: A dynamic permission control mechanism for android third-party libraries," *IEEE Trans. Dependable Secur. Comput.*, vol. 18, no. 4, pp. 1751–1761, 2021. [Online]. Available: https://doi.org/10.1109/TDSC.2019.2937925

[56] 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. [Online]. Available: https://doi.org/10.1109/TIFS.2019.2950134

[57] P. Kumar and S. Singh, "Enhancing android application security: A novel approach using droidxgb for malware detection based on permission analysis," *Secur. Priv.*, vol. 7, no. 2, 2024. [Online]. Available: https://doi.org/10.1002/spy2.361

[58] G. Tao, Z. Zheng, Z. Guo, and M. R. Lyu, "Malpat: Mining patterns of malicious and benign android apps via permission-related apis," *IEEE Trans. Reliab.*, vol. 67, no. 1, pp. 355–369, 2018. [Online]. Available: https://doi.org/10.1109/TR.2017.2778147

[59] Q. Qiao, R. Feng, S. Chen, F. Zhang, and X. Li, "Multi-label classification for android malware based on active learning," *IEEE Transactions on Dependable and Secure Computing*, pp. 1–18, 2022.

[60] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," in *USENIX*. USENIX Association, 2017, pp. 625–642. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/jordaney

[61] F. Barbero, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Transcending TRANSCEND: revisiting malware classification in the presence of concept drift," in *43rd SP*. IEEE, 2022, pp. 805–823. [Online]. Available: https://doi.org/10.1109/SP46214.2022.9833659

[62] Y. He, J. Lei, Z. Qin, and K. Ren, "Dream: Combating concept drift with explanatory detection and adaptation in malware classification," 2024. [Online]. Available: https://arxiv.org/abs/2405.04095

[63] A. Abusnaina, A. Anwar, M. Saad, A. Alabduljabbar, R. Jang, S. Salem, and D. Mohaisen, "Exposing the limitations of machine learning for malware detection under concept drift," in *WISE*, vol. 15437. Springer, 2024, pp. 273–289. [Online]. Available: https://doi.org/10.1007/978-981-96-0567-5_20

[64] Y. Chen, Z. Ding, and D. A. Wagner, "Continuous learning for android malware detection," in *USENIX*. USENIX Association, 2023, pp. 1127–1144. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/chen-yizheng

[65] A. Mishra and M. Stamp, "Cluster analysis and concept drift detection in malware," *CoRR*, vol. abs/2502.14135, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2502.14135

[66] A. S. Li, A. Iyengar, A. Kundu, and E. Bertino, "Revisiting concept drift in windows malware detection: Adaptation to real drifted malware with minimal samples," in *NDSS*. ISOC, 2025.

[67] B. P. Gond and D. P. Mohapatra, "Deep learning-driven malware classification with api call sequence analysis and concept drift handling," 2025. [Online]. Available: https://arxiv.org/abs/2502.08679

**Ahmed Sabbah** received a Bachelor's degree in computer science from An-Najah National University, Palestine, in 2008 and a Master's degree in software engineering from Birzeit University in 2021. He is currently working toward a Ph.D. degree with the Department of Computer Science, Birzeit University. His research interests include security, machine learning, software engineering, and mobile malware analysis.

**Radi Jarrar** received his B.Sc. in Computer Information Technology from the Arab American University in 2007 and a Ph.D. in Computer Science from Monash University in 2012. Since 2015, he has been an assistant professor in the Department of Computer Science at Birzeit University, Ramallah, Palestine. His research interests include machine learning, computer vision, and data science, with applications in computer security.

**Samer Zein** received the M.Sc. degree in Software Engineering from Northumbria University, United Kingdom, in 2004, and the Ph.D. degree in Mobile Software Engineering from the International Islamic University Malaysia (IIUM) in 2016. He is currently an Associate Professor in the Department of Computer Science at Birzeit University. He has over 20 years of academic experience and has contributed as a software engineer to several management information system (MIS) projects since 2000. His research interests include mobile app software engineering, empirical software engineering, model-driven software development, and systematic literature reviews (SLRs). He has also conducted multiple qualitative studies involving contemporary industrial case studies.

**David Mohaisen** (Senior Member, IEEE) received the MSc and PhD degrees from the University of Minnesota in 2012. He is currently a full professor at the University of Central Florida, where he directs the Security and Analytics Lab. From 2015 to 2017, he was an assistant professor at SUNY Buffalo, and from 2012 to 2015, he was a senior research scientist with Verisign Labs. His research interests span networked systems security, online privacy, and measurements. He has been an associate editor for the IEEE Transactions on Mobile Computing, IEEE Transactions on Cloud Computing, IEEE Transactions on Parallel and Distributed Systems, and IEEE Transactions on Dependable and Secure Computing. He is a senior member of ACM (2018) and IEEE (2015), a distinguished speaker of the ACM, and a distinguished visitor of the IEEE Computer Society.