Journal of Computer Virology and Hacking Techniques (2023) 19:65–96
https://doi.org/10.1007/s11416-022-00432-3

ORIGINAL PAPER

Leveraging the first line of defense: a study on the evolution and usage of android security permissions for enhanced android malware detection

Alejandro Guerra-Manzanares[1] · **Hayretdin Bahsi**[1] · **Marcin Luckner**[2]

Received: 25 October 2021 / Accepted: 29 March 2022 / Published online: 26 August 2022

Abstract
Android security permissions are built-in security features that constrain what an app can do and access on the system, that is, its privileges. *Permissions* have been widely used for Android malware detection, mostly in combination with other relevant app attributes. The *available* set of permissions is dynamic, refined in every new Android OS version release. The refinement process adds new permissions and deprecates others. These changes directly impact the type and prevalence of permissions requested by malware and legitimate applications over time. Furthermore, malware trends and benign apps' inherent evolution influence their requested permissions. Therefore, the usage of these features in machine learning-based malware detection systems is prone to *concept drift* issues. Despite that, no previous study related to permissions has taken into account concept drift. In this study, we demonstrate that when concept drift is addressed, permissions can generate long-lasting and effective malware detection systems. Furthermore, the discriminatory capabilities of distinct set of features are tested. We found that the initial set of permissions, defined in Android 1.0 (API level 1), are sufficient to build an effective detection model, providing an average 0.93 F1 score in data that spans seven years. In addition, we explored and characterized permissions evolution using local and global interpretation methods. In this regard, the varying importance of individual permissions for malware and benign software recognition tasks over time are analyzed.

Keywords Android · Permission · Machine learning · Malware detection · Concept drift · Mobile security

1 Introduction

Android malware is ubiquitous and deceptive [1]. Malicious applications disguise in many forms and shapes, constantly adapting in an ever-evolving *sophistication* trend since the early years of Android operating system (OS) [2,3], the leading mobile OS [4]. The open nature of Android and its massive spread make the popular OS an attractive target for cyber attackers, thus posing end-users at constant risk [5]. Furthermore, as mobile devices are increasingly becoming more integrated into our daily routine, from leisure time activities to work-related tasks, mobile security emerges as a central element for individuals and companies to prevent cyber attacks and protect the wealth of sensitive data that mobile devices manage and store [6].

Security permissions constitute the *first line* of defense against malicious threats in Android devices. *Permissions* are a Linux kernel-based Android OS built-in security feature that enables the system to control what apps can do and access, that is, their *privileges*. In Android devices, the user grants or denies access to apps to data and resources from the system via permissions, thus determining the apps capabilities on the system. The original Android OS permissions-based security model implemented an *accept all-or-nothing* policy upon installation. However, due to its critical importance as the first-line security barrier, the security model evolved after the release of Android 6.0 *Marshmallow* (i.e., API level 23) in 2015. Since then, the *new* permissions model lets the user decide to grant or deny specific permissions, related to *sensitive* data, for each app at run-time and not upon installation [7]. This improvement may have led to increased risk awareness, greater flexibility and situational control over the privileges of apps on the sys-

✉ Alejandro Guerra-Manzanares
alejandro.guerra@taltech.ee

[1] Department of Software Science, Tallinn University of Technology, Tallinn, Estonia

[2] Faculty of Mathematics and Information Science, Warsaw University of Technology, Warsaw, Poland

tem, which in turn may have diminished the risk for some users, but it has certainly not eradicated the threat.

Furthermore, even though additional security mechanisms have been implemented at software [8] and hardware [9] level to overcome the traditional limitations of antivirus to detect malware in the mobile platform [10], malware authors have always found their way to bypass them [11]. Statistical figures show that the threat is not only alive but constantly evolving [12].

In recent years, machine learning (ML) techniques have been explored as a means to mitigate exposure to the threat, showing remarkable success even with evolved, *zero-day* and *obfuscated* malware samples [13]. ML-based malware detection systems use statistical properties of collected samples to *learn* about *known* data and make accurate predictions about *future* or *unknown* data (i.e., supervised learning). A wide variety of properties or *features* of malware samples have been used to build detection systems [14]. However, according to their nature, they are broadly categorized as *static* or *dynamic* features. Static features are directly extracted from the source code, without executing the app, whereas dynamic features require the execution of the app in a *live* environment to be acquired.

Permissions are the most used *static* features for Android malware detection purposes [14]. Despite the remarkable effectiveness reported by these detection systems, most of them are built on data belonging to *old* and short time frames within the whole Android history, thus neglecting the impact of time and malware evolution on data. For instance, *Drebin* [15], the most used data set in recent studies, provides data collected between 2010 and 2012. Therefore, this *old* data set becomes an *obsolete* representation of the actual threat landscape and it is not representative of the recent malware threats (e.g., the first ransomware for Android was discovered in 2013) [16].

In addition to that, most studies propose the usage of a reduced subset of features for optimal detection, obtained after applying feature selection methods to the *training* data used to build the detection model. These feature subsets may provide great discriminatory power on the training set timeframe but generate doubts about the generalization of such systems to future time frames, where significant changes may affect relevant data properties due to the natural evolution of malware and benign data, thus directly impacting the discriminatory power of features and, consequently, the detection performance of those systems [17]. As a result, these facts cast severe doubts about the long-lasting detection capabilities of systems built using old data to detect *recent* malware. Relevant features to detect malware effectively are prone to change as malware evolves, a phenomenon called *concept drift*. Neglecting concept drift has a potentially devastating impact on detection systems. Therefore, addressing the detrimental impact of concept drift becomes a critical issue to build effective and long-lasting machine learning-based malware detection systems.

This study aims to address these gaps by taking the impact of time into consideration when *permissions* are used as model features. Firstly, we modeled concept drift effectively by using a solution consisting of an ensemble of ML classifiers. Then, we applied *permutation feature importance*, a global *interpretation* method, to characterize the most relevant features per data period and understand the changes in the discriminatory power of permissions over time. Furthermore, a local interpretation method was used (i.e., Shapley values) to explain individual predictions and *locally* compare permissions evolution for specific malware families. As security permissions are inherently *interpretable* constructs, their characterization provides useful insights about Android malware intentions and its evolution. To the best of our knowledge, no previous study focusing on permissions performed any temporal characterization nor analysis of permissions' importance evolution.

A distinctive fact of our concept drift modeling and characterization is the comparative analysis of the impact of distinct *natural* feature subsets on the model's performance. These feature sets were formed considering the *natural* evolution of the permission set which is updated in almost every new API release. As a result, these feature sets were not selected using any feature selection method, thus not artificially generated nor optimized for a specific time frame.

Lastly, although permissions have been widely used in detection systems, they have been relegated to a *secondary* role, mainly used in conjunction with other static or dynamic features to enhance performance [18]. This study brings permissions back to the *primary* role by showing that using a reduced set of permissions and addressing concept drift, a long-lasting effective malware detection system can be built. This malware detection system showed consistent performance, averaging 0.93 F1 score, in a seven-year-long time frame.

The paper structure is as follows: Sect. 2 provides background information about Android permissions. Section 3 outlines the state of the art in Android malware detection systems using permissions. The methodology used in this research and main results are addressed in Sects. 4 and 5, respectively. Section 6 provides the main findings and highlights discussion points. Lastly, Sect. 7 describes the limitations of this work while Sect. 8 concludes the study and outlines future work.

2 Android security permissions

Everything in Android, from the contacts list to games, is an application and every application, for security reasons, runs in a restricted and isolated environment (i.e., sandbox).

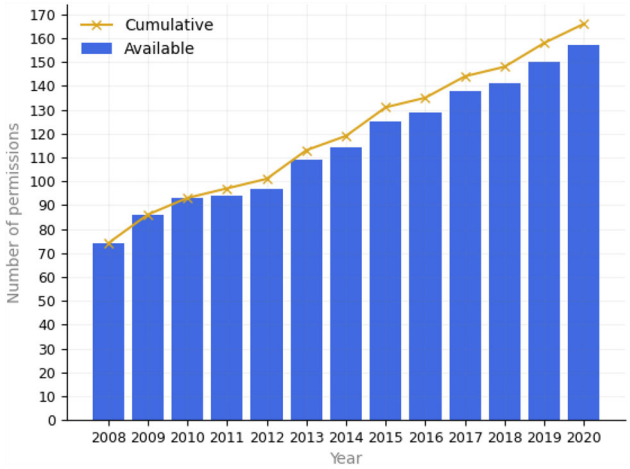


Fig. 1 Android permissions timeline evolution (based on data gathered from [21])

Therefore, if an app needs to access data or resources outside of its environment, it must ask for the necessary privileges to access them [19], requesting the appropriate permission.

In this regard, security permissions support user privacy by protecting access to *restricted data* (e.g., contacts information), preventing *restricted actions* (e.g., take pictures), and limiting interaction with other apps [20]. To be able to perform such actions or access sensitive data, the app must declare all the related permissions in the *AndroidManifest.xml*. The *manifest*, located in the root folder of the app's *.apk* archive, is the only mandatory file for every Android app and is used by the OS to get relevant information for the proper handling of the app.

The initial set of *standard* Android permissions available on the first Android release (i.e., API level 1) was composed of 74 permissions [21]. Since then, the set of available permissions has been modified, adding or deprecating permissions, on almost every new API release. This constant change has been in line with the increase in smartphone capabilities and the need for new security measures to handle them. The latest released version at the time of performing this research, Android 11 (i.e., API level 30), released in September 2020, has 157 available permissions. Thus, the permissions available suffered a two-fold increase in 13 years. More specifically, 166 permissions have been defined since API level 1 but 9 were deprecated in API updates. Figure 1 shows the evolution of Android permissions from 2008 to 2020, that is, from API level 1 (i.e., Android 1.0) to API level 30 (i.e., Android 11). The yellow line shows the cumulative number of permissions defined over time, while the blue bars indicate the number of *actually* available permissions for each API release/year (i.e., the usable set, without the deprecated permissions).

The dynamism of the phenomenon is evidenced in Fig. 1. The permissions set has increased more than two-fold since the first Android release and is constantly updated, especially in the most recent API releases. The updates have introduced new security permissions in response to new phone features (e.g., *USE_FINGERPRINT*, API level 23) or refine/extend existing permissions (e.g., *ACCESS_BACKGROUND_LOCATION*, API level 29, which refines *ACCESS_COARSE_LOCATION* and *ACCESS_FINE_LOCATION*) [21].

Besides, the usage of permissions is directly influenced by trends and behavioral changes in apps, affecting the prevalence of permissions over time. Therefore, the natural evolution of the feature set and the changes in prevalence over time make the usage of permissions for Android malware detection prone to concept drift issues.

Android security *standard* permissions are categorized into three risk or *protection* levels: *normal*, *signature*, and *dangerous* [21]. *Normal* permissions grant access to data and actions that pose a minimal risk to the user's privacy and the operation of other apps. *Signature* permissions are granted by the system to apps that declare a signature permission that another app has defined when both apps are signed with the same certificate. *Dangerous* permissions enable access to sensitive user data or actions that may affect the system and other apps. In addition to the *standard* permissions, developers can create their own permissions (i.e., *custom* permissions) which allow these apps to share their resources and data with other apps signed with the same certificate [22]. An additional category, outside of the scope of app developers are *special* permissions. These permissions are related to particular powerful app operations that only the platform and original equipment manufacturers can define [21].

Before Android 6.0 *Marshmallow*, all the permissions declared in the app manifest were granted automatically upon installation. So, if the user did not want to accept *all* the permissions the app declared, then the installation was not possible. This security paradigm changed in API level 23 with the inclusion of *run-time* permissions [23]. Since then, normal and signature permissions are granted automatically upon installation (i.e., *install-time* permissions), whereas dangerous permissions are requested for user acceptance during the app execution via an approval prompt (i.e., *run-time* permissions) [24]. Android 11 added further enhancements such as more granular permissions, *one-time* permissions, and the auto-reset of sensitive permissions for unused apps [25]. Recent Android releases have also focused on privacy issues and the update of the permission system [26], as evidenced by the refinement of the permission set and the addition of new permissions, as shown in Fig. 1. For instance, API level 29 introduced ten permissions and deprecated one, while API level 30 defined eight new permissions and removed one. These enhancements aim to provide more control to the users, transparency, and minimize data usage [20].

3 Related work

Android security permissions have been widely used in research since the early days of Android OS, becoming the most used static feature for Android malware detection [14]. Although static features are regarded as inherently *weak* against deception mechanisms such as encryption and obfuscation (i.e., especially API calls), Android permissions are relatively robust as without the required permission, the malicious behavior, obfuscated or not, might not be triggered [14]. Furthermore, permissions analysis enables direct detection on the device, upon download and without the need for app installation, execution [15] nor *root privileges* [27], featuring low computational cost and high efficiency [28]. Therefore, permissions constitute the first barrier to attackers, which could be leveraged for highly effective and efficient on-device malware detection.

Permissions have been widely used as input features to build Android malware detection solutions since the early days of Android OS. In this regard, they have been used alone, using all *available* permissions [27,29–31], selected subsets [17,32–34], or patterns and relationships between permissions [35–39], but also in combination with other static features extracted from the app manifest [18,40–42] and the decompiled source code [15,28,43–46]. Besides, so-called *hybrid* approaches have used permissions jointly with dynamic features such as system calls [47], run-time API calls/events [48,49], and network traffic [50,51].

However, despite the wide use of security permissions to build ML-based malware detection systems, no deep analysis of the feature evolution has ever been performed nor considered, thus neglecting the impact of the *time* variable on the learning models and its long-term evolution.

Among all the studies using permissions, only Hu et al. [46] considered concept drift. This study proposed a method to handle concept drift using a static feature set that includes permissions, API calls and actions (i.e., 405 features). Even though the method reports high accuracy, permissions are used in combination with other features, thus their analysis and impact on the overall performance of the detection model are not provided or explored. Besides, the data set used combines different data sets belonging to undefined and discontinued time frames (i.e., *Drebin* [15] and other unspecified sources), assuming the existence of sudden concept drift in the data. In this regard, a more *gradual* concept drift should be considered in a more realistic scenario where the threat landscape gradually evolves towards new threats based on old threats where sudden drifts might be possible but are less likely. In contrast, our study encompasses a long period in which a significant modification of the permissions set has been performed, thus capable of handling sudden and gradual drift, and our model only on permissions as input features, enabling us to assess and characterize using interpretability methods the impact of specific permissions and their evolution on the detection performance.

Besides, the vast majority of related research uses old, short, and static snapshots of Android malware historical data to build, validate and test their systems. For instance, *Drebin* [15] and *MalGenome* [52], the most used data sets for Android malware research, provide samples restricted just to the 2010–2012 time frame [53]. This fact poses serious concerns about the generalization capabilities of the proposed detection systems, built on old and non-representative data, to future and evolved malware.

Furthermore, in the studies that use data encompassing wider time frames to build ML-based detection systems (e.g., using two distinct data sets such as Drebin and Contagio [54]), the common practice is to merge the data and split the resulting data set randomly into disjoint sets of arbitrary proportions (i.e., the *training* and the *validation*/*testing* sets) [55]. This *typical* randomization procedure in machine learning, when applied to time-series data, introduces *temporal bias*, which has yielded not representative and overly inflated performances in Android malware detection research, not adjusted to the actual performance [56] due to the lack of *historical coherence* when data is split disrespecting the *historical timeline*. For proper validation, the testing data must belong to a *future* or *posterior* time frame regarding the training data [55,57]. An additional issue is that it is common that malware and benign samples used in the same data set do not belong to the same time frames, leading to an *historically* impossible configuration [44]. For instance, it is typical that benign data is collected at the time of research, whereas malware belongs to a well-known data set, such as Drebin, collected long before. This configuration generates biased detection systems and inflated performance as the features to describe apps belonging to distinct time frames might be too different (e.g., new permissions available, new requirements on permissions usage, etc.), thus generating an artificial scenario that does not reflect the real challenge of recognizing between malware and benign apps belonging to the same time frame.

These *common practices* provide unrealistic scenarios and neglect the impact of *time* on the input features and, consequently, its detrimental effect on the ML models over time (i.e.,*concept drift*). Therefore, neglecting concept drift provides biased and historically incoherent results, not adjusted to real scenarios [55–57].

Finally, the existing body of research has shown that the combination of permissions with other features may yield better performance than the usage of permissions alone [47,50,51,58], which has relegated permissions to a *secondary role*, as a complementary feature, mainly used in combination with other relevant features. Even though the combination may yield better detection performance, it does not provide any insights into the evolution of permissions,

their relevancy to the detection performance, or the changes in their discriminatory power over time.

This study focuses solely on the usage of permissions to detect Android malware, showing that when concept drift is addressed, permissions alone can provide and keep high-performance metrics over time. Notwithstanding that the combination with other features may yield increased performance, it is out of the scope of this research. To the best of our knowledge, no previous research has considered the concept drift issue for permissions, which has been explored for other features used in Android malware detection such as API calls [59,60] and system calls [61], nor performed a characterization of the phenomenon and assessed its impact on the detection performance over time.

4 Methodology

The following subsections detail the data set used in this research and concept drift basics, which enable the understanding of the approach used. After, the methodological workflow followed in this study, depicted in Fig. 2, is thoroughly explained.

4.1 Dataset and features

The data set used in this research is *KronoDroid* [53], a hybrid-featured data set that provides labeled samples of Android benign and malicious apps dating from 2008 to 2020, which makes it suitable for the analysis of the evolution of features and concept drift issues. Each sample in the data set is described by 489 features (i.e., 289 dynamic and 200 static). The dynamic features were collected using two distinct Android platforms, an emulator and a real device. Thus, the data set is originally split into two overlapping sub-datasets according to the source of the dynamic data. The data sets overlap in those samples that were able to provide dynamic data on both platforms but also contain distinct data samples, those acquired in only one of the sources. As in this research the interest lies just in the analysis of static features, disregarding the dynamic source, both data sets were merged to obtain the largest possible data set. The duplicated samples, caused by the overlap of the data sets, were removed. As a result, the data set used in this work is composed of 78,804 samples (i.e., 37,020 benign and 41,784 malware).

Regarding the input features, all the individual permissions-related features provided by the data set, the class label, and a timestamp were used. Therefore, each sample was described using 168 static features. More specifically, the permission-related features are composed of 166 categorical attributes, which are binary indicators of the presence of the permission in the app manifest (i.e., set as 1 if the permission is requested by the app and 0 if it is not). The data set provides permissions data until API level 30 (i.e., Android 11, released in September 2020). A total of 166 permissions were defined until API level 30 [21]. The timestamp used to locate each app within the Android historical timeline was the *last modification* timestamp. This timestamp is reported as the most reliable and accurate regarding the *historical context* from the four alternative timestamps provided by the data set. The last modification timestamp dates the app to the most recent timestamp retrieved from any of the app archive inner files [53]. Lastly, the class label (i.e., malware or benign) was obtained for each sample.

4.2 Concept drift

Static detection models assume that the statistical properties of the target distribution are relatively fixed, not changing over time. However, the malware threat landscape is dynamic, making the underlying distribution change over time (i.e., $P_t(\boldsymbol{X}) \neq P_{t+1}(\boldsymbol{X})$), referred to as *covariate shift*. When this shift affects the decision boundary of the classifier, and consequently, the class estimation (i.e., $P_t(\boldsymbol{Y}|\boldsymbol{X}) \neq P_{t+1}(\boldsymbol{Y}|\boldsymbol{X})$), the performance of the model is significantly harmed over time. Besides that, concept drift may also occur if the class estimation changes while $P_t(\boldsymbol{X})$ remains unchanged [62]. According to the speed of the changes, *incremental* drift, *gradual* drift, *sudden* drift, and *re-ocurring* drift have been proposed as concept drift typologies. The appearance of any of these typologies requires the update of the learning model [62]. Therefore, an effective malware detection model should be able to update its knowledge considering the ever-evolving threat landscape, keeping high detection performance in a non-stationary input context.

In this regard, to define concept drift for continuous analytics, let us define a new observation as $c_i = (x_i, y_i)$, where $x_i = (x_i^1, x_i^2, ..., x_i^n) \in \mathbf{X}$ is the feature vector and $y_i \in \mathbf{Y}$ is the target label. The incoming observations $c_i, \ldots, c_{i+k}$ are aggregated into sets of the same size k called *chunks*. To avoid the integration of extended periods in a single chunk, which may harm the detection of concept drift, temporal restrictions were imposed (i.e., a maximum of three months of data per chunk). In such a case, observation c_i is additionally described by a timestamp t_i and each chunk is a set $P_{[t_{min},t_{max})} = \{c_i : t_{min} \leq t_i < t_{max}\}$, where t_{min}, t_{max} define the temporal borders of the given chunk.

Next, let us assume that features from two chunks are given by distributions F and F'. *Feature drift* is defined if the null hypothesis H_0 that F and F' are identical can be rejected [63]. Feature drift is categorized as concept drift if and only if the change in the distribution leads to a change in $\hat{y}$ estimation, which corresponds to a change in class estimation by the model in the described malware detection case. Therefore, concept drift, which relates to a change in the model's detection performance, is the focus of our analysis.

4.3 Workflow

The main objective of this work is to analyze the changes and evolution in permissions usage in Android apps and, more specifically, in malware (i.e., characterization). The focus lies on the impact of *time* on the detection capabilities of permission-based malware detection models (i.e., concept drift). Therefore, the optimization of classification quality is not a primary objective of this research. Nevertheless, for a meaningful analysis, the malware detector used during the analytic process must obtain high and preferably stable quality. The required stability of detection quality is challenged by the concept drift phenomenon observed in malware data [46,64], especially for long-term observations. The workflow performed in this research is provided in Fig. 2 and explained in the following subsections.

4.3.1 Data preprocessing

The feature vector defined for each app was composed of 166 permissions binary indicators, a timestamp, and the class label. As when some features were not available at the time, the vectors were filled with 0 values, this may have introduced noise in the learning models and provide biased results. To explore this issue, models with distinct permission set sizes were compared in this work. More specifically, the initial feature set from API level 1 (i.e., 74 permissions) and the whole feature set from API level 30 (i.e., 166 permissions) were evaluated. In the former case, the model was restricted to using the features from the initial set during the whole analysis period, whereas in the latter case, the model was free to select what feature sets provided better performance for each specific period.

However, before any analysis is performed, the feature set should be clear of redundant and irrelevant attributes that may affect the performance of the detection model and distort the characterization results. To address these issues, two sequential steps were performed. Firstly, sample variance was calculated for all features. The features with variance equal to zero were removed (i.e., constant or zero-valued) as they were irrelevant to building the discriminatory model. Next, highly correlated features were removed. The removal of strongly correlated features enabled us to eliminate redundant data as well as improve the quality of the data for the characterization step using the permutation importance technique [65]. To analyze the correlation between individual permissions, which are categorical features, *Kendall's rank correlation* [66] was used and calculated pairwise for all features. Last, as depicted in Fig. 2, in the final preprocessing step, data samples were divided into n sequential chunks defined by timestamps $t_1, \ldots, t_{n+1}$ as $P_{[t_1,t_2)}, P_{[t_2,t_3)}, \ldots, P_{[t_n,t_{n+1})}$.

4.3.2 Concept drift handling detection model

After the data preprocessing phase, the resulting data set was suitable to be processed by the malware detection model. The data set was analyzed chunk by chunk, simulating a realistic batch processing model for data streams [67]. Therefore, an algorithm that addresses the concept drift issue for stream data could be used to tackle the Android malware detection concept drift. In our solution, an existing method that uses a pool of classifiers was implemented [68]. More specifically, during the data stream processing, the algorithm dynamically selects the best ensemble of classifiers from an existing pool of classifiers to forecast the labels of the samples of the new data chunk. In our implementation, all the classifiers in the pool were random forest models, which yielded high performance in related studies [47,61,69,70]. Each classifier of the pool of classifiers is trained on a different historical/previous data chunk. This enables the pool to contain varied models and to expand its knowledge with data belonging to different time windows. This previous knowledge is used to predict the new data chunk. An updating mechanism is implemented by the algorithm every time a new data chunk is processed, introducing a new classifier trained on the newest chunk and removing the worst classifier (i.e., *aging* model), which causes the pool to be constantly modified after each chunk. The update mechanism aims to handle and address concept drift in the data. The original algorithm [68] was modified to address Android data particularities as described in [61]. More precisely, based on the implementation proposed by Guerra-Manzanares et al. [61], the following changes were performed:

1. The algorithm started with a full pool of classifiers and kept the pool size static during the whole analytic process. This change minimizes the differences in the quality of the detection process in its initial phases.
2. The pool of binary classifiers was supported by an anomaly detection model trained only with benign data. This modification improves the recognition of benign software, which was underrepresented in most data chunks and showed more consistent features over time (i.e., less *drift* in the data).

Malware detection can be defined as a binary classification problem, where *TP* (i.e., true positive) refers to the number of correctly recognized malware among all test instances. *TN* (i.e., true negative) reflects the number of correctly recognized benign software among all test data. *FP* (i.e., false positive) provides the number of actual benign samples incorrectly recognized as malware among all test samples and *FN* (i.e., false negative) the number of actual malware samples incorrectly identified as benign data by the classifier in the test set.

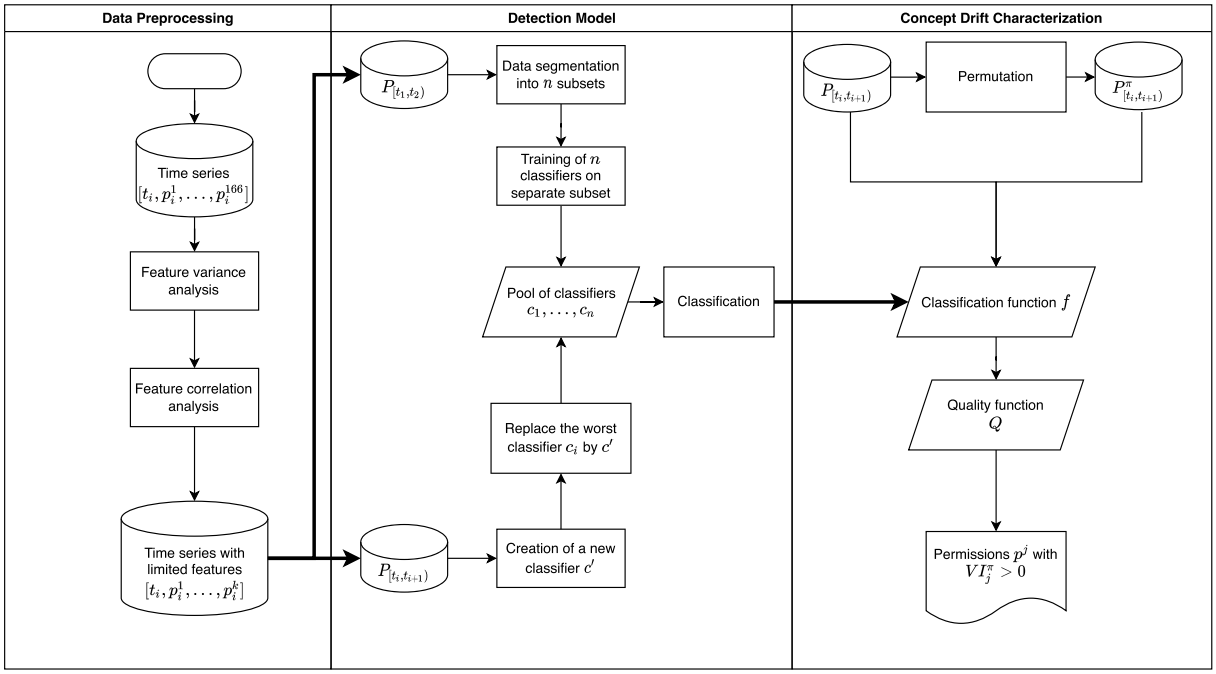


Fig. 2 Workflow of this study

Based on these concepts, the *F1 score* is used as a comprehensive metric for malware detection performance on imbalanced data sets. F1 score can be formulated as

$$F1 = \frac{2TP}{2TP + FP + FN} \tag{1}$$

In our experimental setup, the F1 score metric was used to analyze $\hat{y}$ estimation performance of the permissions-based malware detection model for each chunk separately.

4.3.3 Data analysis and characterization

The F1 score gives a global picture of the detection performance of the proposed model. However, for a deeper analysis of permissions as a malware detection source, it is relevant to evaluate the influence of each permission separately on the detection process. As the detection model built was based on an ensemble of Random Forest classifiers, the built-in variable importance estimation [71] could be used for this task. In such a case, the *importance* calculation is tied to the optimization functions, which are generally *accuracy* or *F1* measures. However, for a better understanding of the analyzed data, it is more relevant to observe the influence of specific permissions on more specialized functions such as *specificity* and *recall*.

Specificity or True Negative Rate (TNR) is a measure of benign software acceptance (i.e., negative label recognition) and is calculated as:

$$TNR = \frac{TN}{TN + FP} \tag{2}$$

Recall or True Positive Rate (TRP) is a measure of the quality of malware detection (i.e., positive label recognition), defined as:

$$TPR = \frac{TP}{TP + FN}. \tag{3}$$

A preferred approach for analyzing features' importance using these specialized metrics is *permutation feature importance* analysis [72]. Permutation feature importance is a model-agnostic variable importance evaluation technique that is not tied to any particular metric or data set, suitable for analyzing the importance of model features for different quality metrics on the training and testing data sets. In this study, it was used as a tool to analyze the feature importance changes among distinct data chunks. More specifically, the permutation feature importance method evaluates how the random permutation of a feature influences the quality function calculated for the given decision model. For a matrix of feature values $\mathbf{X}$ with rows $\mathbf{x}_i$, given each of N observations and corresponding response y_i, $\mathbf{x}_i^{\pi,j}$ is a vector achieved by randomly permuting the jth column of $\mathbf{X}$, where π refers to a random permutation. According to [73], the permutation process should be repeated at least 50 times to obtain stable results. Originally, the method is defined for a loss function

L, but it can be easily adapted for the quality function Q with range [0, 1] using $L(.) = 1 - Q(.)$. Then, the importance VI_j of the jth feature is calculated as the difference between the value of L for pseudo-randomized values and the value of L for the original data.

The proposed analytic tool uses the classification function $f_{[t_l,t_{l+1})}$ induced on data from chunk $P_{[t_l,t_{l+1})}$, where t_l and t_{l+1} define the temporal boundaries of a time period. The analysis observations X are taken from the corresponding chunk, $P_{[t_l,t_{l+1})}$. The procedure is summarized by the following equation:

$$VI_j^{\pi}(P_{[t_l,t_{l+1})}) = \frac{1}{N} \sum_{\substack{i=1, \\ x_i \in P_{[t_l,t_{l+1})}}}^{N} Q(y_i, x_i) - Q\left(y_i, f_{[t_l,t_{l+1})}(\mathbf{x}_i^{\pi,j})\right) \quad (4)$$

To analyze the importance of individual permissions for the recognition tasks, all features with $VI_j^{\pi}(.) > 0$ (i.e., positive importance) were retrieved for the analyzed data chunk.

Based on Eq. 4, the importance of features on the training set for the generated classifier is retrieved. The analysis of the important features for each new classifier's training set enables us to determine the relevant set of features for each specific time frame. When performed on consecutive data chunks, it allows the observer to analyze the changes in permissions' importance over time.

The permutation feature importance technique is a *global interpretability method* as it reports the average behavior or expected value of an ML model. Therefore, global interpretability methods are particularly useful to understand the general mechanisms in the data [74], which, in our case, enables us to study concept drift behavior. However, this wide-scope analysis may fail to grasp the particularities behind individual decisions. In such a case, *local interpretability* methods might provide a better answer. Local interpretability methods allow explaining the reasoning behind individual predictions. In our research, for the analysis of specific model predictions, *Shapley values* were used. Shapley values enable us to fairly attribute the prediction output among the relevant features, and it is the only local explanation method with a solid theoretical foundation [75]. The theoretical background of Shapley values lies in *coalitional game theory* [76]. The method aims to assess the contribution or importance of each feature in particular decisions (i.e., each feature is a *player* in a cooperative game or coalition where the prediction is the payout) [75]. As reflected in Eq. 5, the Shapley value of a feature (ϕ_i) is its contribution to the payout, weighted and summed over all possible feature combinations. It is calculated as:

Table 1 Data preprocessing results

Variance analysis	26 zero-valued
Correlation analysis	18 high-correlated
Final feature set	122 permissions

$$\phi_i(v) = \frac{1}{|N|} \sum_{S \subseteq N\{i\}} \binom{|N|-1}{|S|}^{-1} (v(S \cup \{i\}) - v(S)), \quad (5)$$

where N is the number of features, S is a subset of these features, i is the vector of feature values of the instance to be explained, and the function v calculates the payout for any subset/combination of features.

5 Results

The main results of the workflow followed in this research are described in the following subsections.

5.1 Data preprocessing

After the application of each sequential preprocessing step, the results reported in Table 1 were obtained.

From the initial set of 166 individual permissions, variance analysis showed that 26 were constant or null-valued for all data samples, so they were removed from the feature set as they did not provide any relevant information. Furthermore, correlation analysis reported that 18 features were highly correlated with at least another feature (i.e., Kendall's $|\tau| > 0.80$), evidencing redundancy in the data. These 18 features were removed. As a result, the final permissions feature set was composed of 122 permissions. This feature set is further referenced in this work as the *extended* or *full* feature set.

To test permissions evolution and explore the bias caused by the zero-filled values for the unavailable permissions, a reduced feature set was formed using only the permissions defined for API level 1 that remained available until API level 30. After the preprocessing steps, this *reduced* feature set was composed of 60 permissions.

Therefore, two feature sets were used in this research. The *extended* or *full* feature set includes all permissions defined during the whole Android history until API level 30 (i.e., 122), and the *reduced* or *initial* feature set is composed of the permissions defined in API level 1 that remained available until API level 30 (i.e., 60).

5.2 Detection model

The dynamic model described in Section 4.3.2 was used to classify and analyze malware and benign app samples described by permissions. The input data was described using both feature sets. The data samples were aggregated for each quarter of the 2011-2018 period (i.e., three months data chunks). This temporal constraint was found experimentally as the optimal for concept drift handling using these data. F1 score was calculated for each period using a dynamic ensemble selection from a pool of n classifiers as detailed in Section 4.3.2. In our experiment, $n = 12$ provided the best results.

Figure 3 illustrates the changes in the F1 score quality measure for malware detection among periods. The solid blue line shows the F1 score per chunk using the extended feature set, whereas the dashed blue line provides the same information for the reduced feature set. The horizontal yellow lines provide the average values for both cases.

As can be observed in Fig. 3, the performance of the proposed method to deal with Android data concept drift is relatively stable. Except for some periods (i.e., 5 chunks out of 28), the obtained results exceed 0.91 F1 score in all periods. The results obtained for the extended feature set and the reduced set are very similar, with a visible advantage of the reduced vector in the first period and a slight improvement of the full vector in the last periods. Furthermore, F1 score mean values are almost identical (i.e., ~ 0.93).

To formally compare both feature sets of input data, the non-parametric *Kolmogorov–Smirnov* test was used to analyze the equality of both probability distributions (i.e., *two-sample K–S test*). The results of the statistical analysis, depicted in Fig. 15 and further described in Appendix A, confirmed that there is no significant difference in detection performance between the compared feature sets.

A deeper inspection of Fig. 3 enables us to observe that, for both feature vectors, there are three visible quality drops in performance: in the initial period (i.e., 2011-Q3), 2015-Q4 and 2016-Q3. The nature of these drops is deeply analyzed and discussed in Section 5.3. In any case, despite these incidental dips, the system provided an average 0.93 F1 score in the analyzed period, which spans seven years of historical Android data.

In conclusion, based on the experimental results and the statistical analysis performed, depicted in Figs. 3 and 15, respectively, the discriminatory power of the *initial* feature set (i.e., 60 permissions) is comparable to the extended feature set (i.e., 122 permissions). This fact allows concluding that the set of initial permissions has a more significant role in malware detection than the later added permissions in the analyzed period. More features did not provide a better performance, emphasizing the goodness of a *small* subset of features to provide consistent and high malware detection performance over time, even in the presence of *evolving* data. Therefore, when concept drift is addressed, a high-performance malware detection solution can be built using just a small number of permissions as input features without needing to constantly update the feature set when new permissions are defined. Even though the importance of features changed over time (see Sect. 5.3), the initial feature set was found of critical relevancy to generate a long-lasting and robust permissions-based Android malware detection system.

5.3 Data analysis and characterization

The following subsections provide a characterization of the concept drift observed using interpretation methods. More precisely, Section 5.3.1. analyzes the evolution of the importance of individual permissions over time for recall and specificity tasks. Section 5.3.2. matches the class-based performance with the characterization analysis, and Section 5.3.3. performs a thorough analysis of the malware families' distribution over time and its impact on the learning models and concept drift.

5.3.1 Permissions' importance evolution

This subsection presents a thorough analysis of permissions' importance evolution over time. Features' importance was calculated using *specificity* and *recall* as quality (Q) functions. In this regard, importance calculated for specificity informs about which permissions are essential to recognize benign/legitimate applications. In the case of recall, it reports permissions important for the malware detection task. Permutation feature importance was calculated for the full and reduced feature sets separately for each quarter. For the sake of the stability of the results, the permutation feature importance procedure was performed four times with 2000 permutations per iteration.

Figures 4 and 6 provide the set of important features and their relative importance values calculated for the full feature set using specificity and recall, respectively. More specifically, each bar corresponds to a specific period (i.e., year quarter) where the color refers to a specific feature, as indicated by the graph legend. The bar range goes from 0 to 1, meaning that it contains the total feature importance for the period. The colored areas provide the relative feature importance of each permission regarding the total feature importance for that period. The vertical length of the areas is equated to the percentage or proportion of the total importance provided by each specific feature. For instance, in 2016-Q1 in Fig. 4, five colored areas are observed which relate to five different permissions found important, in varying proportions, to recognize benign apps in that specific period. Namely, SEND_SMS,

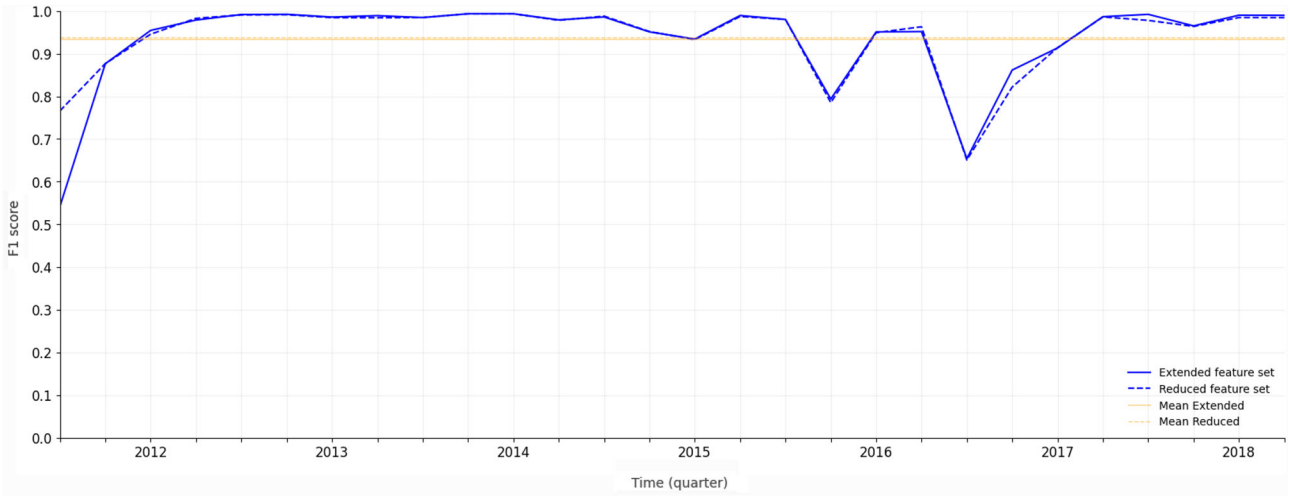


Fig. 3 Evolution of the detection performance using different feature sets

INTERNET, READ_PHONE_STATS, GET_TASKS and MOUNT_UNMOUNT_FILESYSTEMS permissions are represented by the purple, light green, dark blue, dark green and light blue areas, respectively. The grey regions refer to the importance related to features not present in the reduced feature set as, for the sake of interpretability and comparison between feature sets performance, in both Fig. 4 and 6, only the features belonging to the reduced feature set are depicted with colors and the ones belonging just to the extended feature set are reported in grey.

The white line provides the value of the maximum importance found for a feature in each specific period. For instance, in 2016-Q1 in Fig. 4, the most important feature, SEND_SMS, reports average feature importance of 0.35 (i.e., the value of the white line for that bar), meaning that when this feature was randomly shuffled, the specificity metric dropped, on average, 0.35 or 35%. Therefore, combining the information of the colored bars (i.e., relative importance) with the white line values (i.e., maximum absolute importance) provides a better depiction of the important features and their impact on the specific recognition tasks.

Figures 5 and 7 provide the same illustration of changes in feature importance for the reduced permission set. As can be noticed, no grey areas are found on these graphs. For the sake of comparison and interpretation of the results, in Figs. 4, 6, 5 and 7 characterization graphs, only the same set of permissions is depicted and with the same colors (i.e., only features belonging to the reduced set). Overall, as can be observed, the contribution of the later added permissions to the total importance (i.e., grey areas in Figs. 4 and 6) is less significant than the reduced permissions set. This observation aligns with the results reported in Figs. 3 and 15.

In the case of the extended feature set, 85 out of 122 features obtained average positive importance in at least one quarter, whereas for the reduced feature set, this value is 48 out of 60. Of these 48 features, 44 are present in both feature sets. The larger number of features important for the extended set causes a reduction of the importance value of particular features, as evidenced by the grey areas found in the extended set graphs (i.e., Figs. 4 and 6). However, even though the presence of these features does not allow the bars to be completely color-filled (i.e., except for 2015-Q3 in Fig. 6), the importance of the initial set of features is remarkably more significant in all periods, as they are responsible for more than 85% of the total importance of the whole time frame analyzed (i.e., 2011–2018).

For the specificity task, the distribution of the essential features is similar for both feature vectors, as depicted in Figs. 4 and 5. The most important permission in a single quarter is MOUNT_UNMOUNT_FILESYSTEMS (i.e., 2017-Q2). The four permissions with the largest overall importance (i.e., SEND_SMS, READ_PHONE_STATE, MOUNT_UNMOUNT_FILESYSTEMS, and INTERNET) are the same for both feature vectors. These results might be expected as they are permissions related to common mobile phone functions such as communications, phone calls, and access operations to storage file systems. The contribution to the importance of the subset of permissions only belonging to the extended permissions set is visibly lower than the importance of the permissions included in the reduced feature set. Even though the impact of the extended features has fluctuated over time, the common subset of features explains most of the feature importance in the whole analyzed time frame, thus being critical for effective benign software detection over time. Furthermore, the absolute importance values (i.e., white line) reflect that the importance of the most important feature per quarter is extremely high for this recognition task, averaging 26.9% for the extended feature set and 33.1% for the reduced feature set.

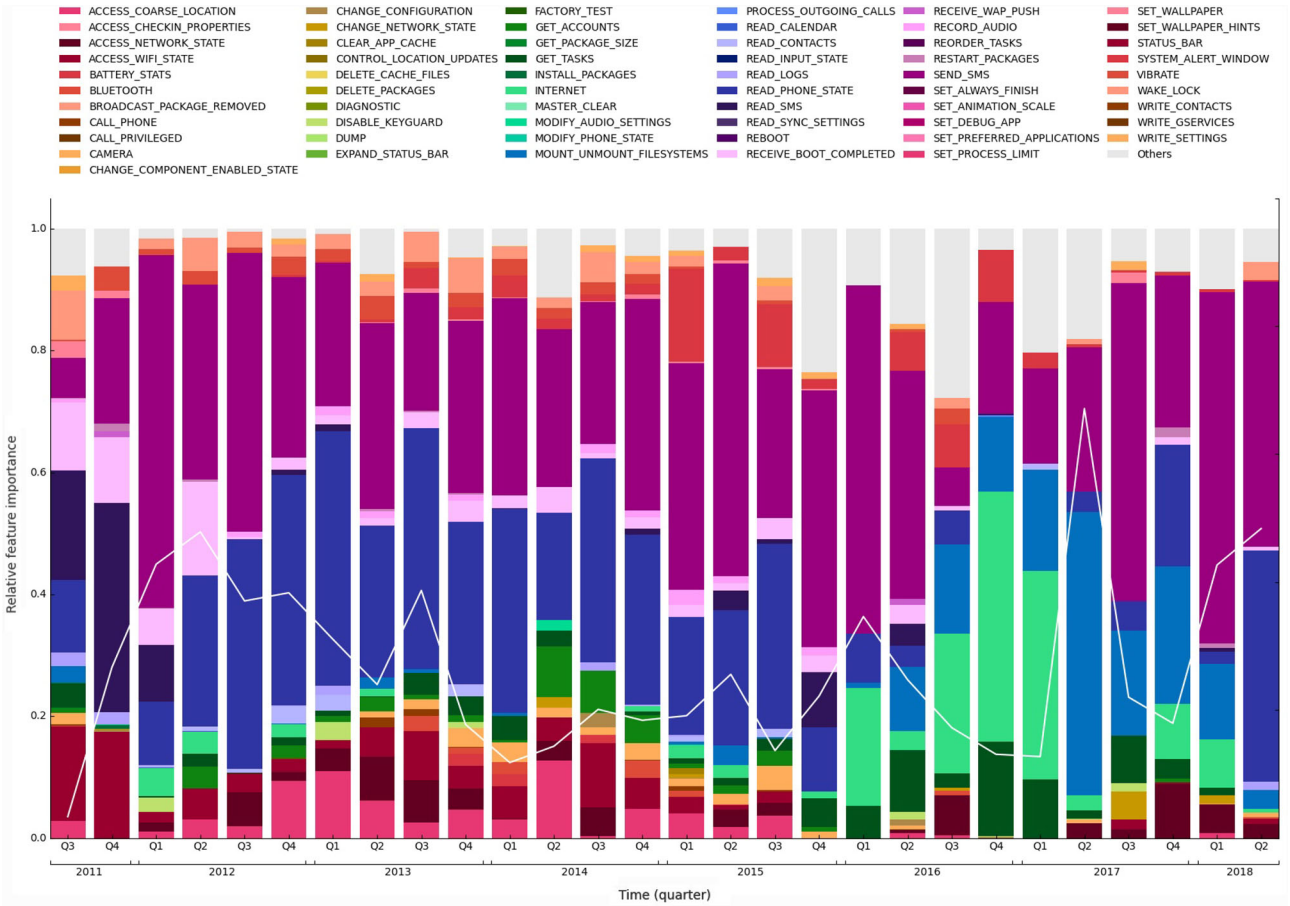


Fig. 4 Importance for specificity, calculated for the extended feature set

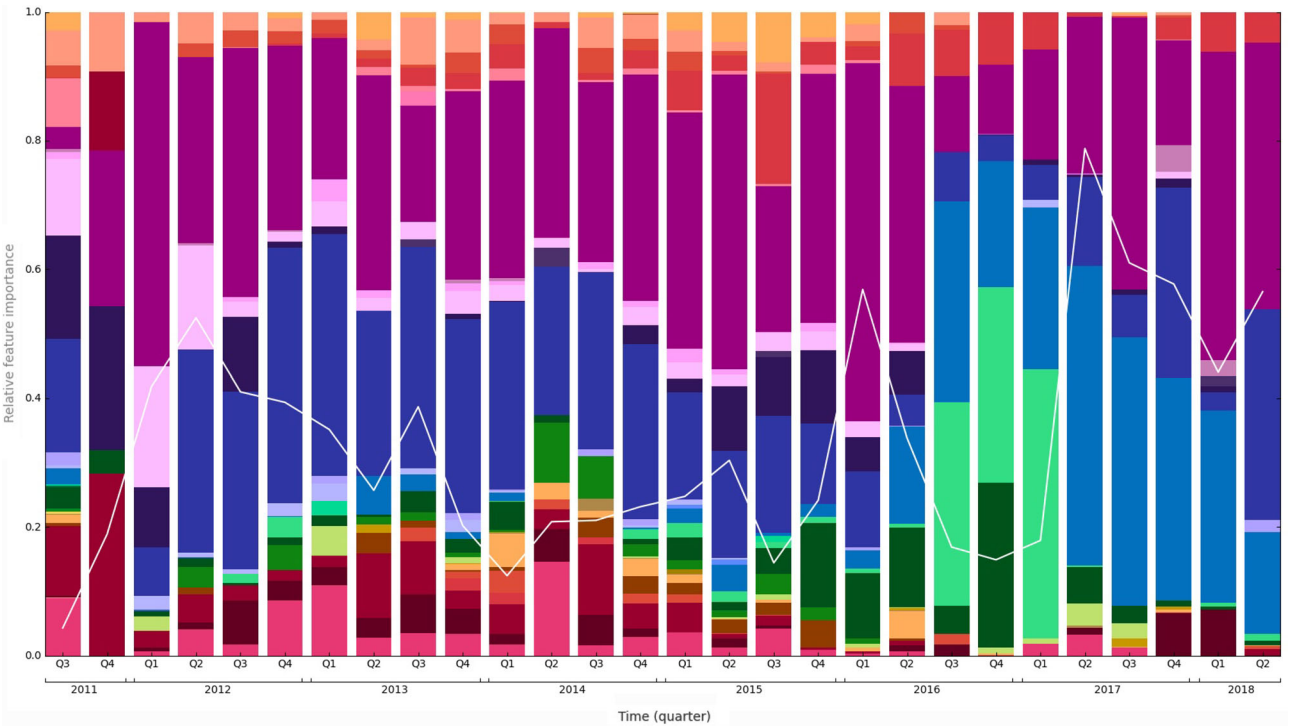


Fig. 5 Importance for specificity, calculated for the reduced feature set

Figure 5 clearly illustrates the existence of concept drift in benign data. As can be observed, READ_PHONE_STATE permission (i.e., dark blue), which was very relevant in the period from 2012-Q2 to 2016-Q1, lost its significant role in the period encompassing from 2016-Q2 to 2017-Q3, being outperformed by MOUNT_UNMOUNT_FILESYSTEMS (i.e., light blue) in that specific time frame. More precisely, MOUNT_UNMOUNT_FILESYSTEMS emerged as the most important feature for that period and kept its significant role after it. Furthermore, for a few quarters, INTERNET permission (i.e., light green) surged as a locally important feature. Beyond 2017-Q3, READ_PHONE_STATE importance role surged back again after the weakening period. These *sudden* changes in the importance of features may have contributed to the sharp decrease in performance observed in 2016-Q3, as shown in Fig. 3). Besides, the sudden change is correlated with a decrease in the maximum absolute importance (i.e., a major dip in the white line), coincidental with the local dominance of the INTERNET permission.

For the malware recognition task, depicted in Figs. 6 and 7 for the extended and reduced feature sets, respectively, the situation is significantly different. In this case, the importance of features changed dramatically over time, suggesting a more complex and abrupt *drift* in the data. More specifically, READ_PHONE_STATE and SEND_SMS were the most important features in the 2012-2013 time frame. After it, SEND_SMS rarely became significant, whereas READ_PHONE_STATE kept its significant role for almost all the analyzed time frame. In the period from 2014-Q1 to 2014-Q3, a sudden change in the important permissions is observed, especially in 2014-Q2 when permissions not included in the reduced set emerged as critically important (i.e., 50% of total importance). A fact that was repeated in 2017-Q1. From 2014-Q4, READ_PHONE_STATE surged back as the most important feature, showing decay over time and finally being of minimal importance from 2017. Thus, the opposite situation to the specificity case is observed. Lastly, from 2017-Q2, a similar pattern is spotted on both recall graphs, with a larger number of features sharing the status of important features and similar relative importance values.

A remarkable fact in the recall case is that, as opposed to specificity, the absolute magnitude of the most important feature per quarter is small, never surpassing 0.25 or 25%, and with an average value of around 4.8% in both feature sets.

The observed concept drift for recall when the reduced feature set is used is similar to the extended feature set case but more evident in the last periods, as shown in Fig. 7. More accurately, until 2017-Q1, READ_PHONE_STATE permission was consistently the most critical feature. After that period, this feature was marginalized, emerging as relatively important in just two periods, in quarters dominated by SYSTEM_ALERT_WINDOW and RECEIVE_BOOT_COMPLETED permissions, among others. Despite that, the overall trends and most important permissions are consistent between both feature sets but show different overall importance orderings (i.e., READ_PHONE_STATE, SEND_SMS, WAKE_LOCK, ACCESS_WIFI_STATE, RECEIVE_BOOT_COMPLETED and ACCESS_NETWORK_STATE). However, the influence of the extended features for recall appears to be much more significant in specific periods than in the specificity case. For instance, in 2014-Q2 and 2017-Q1, the importance is almost evenly split between the reduced set features and the extended set features. A fact not observed in the specificity case, with smaller grey areas in Fig. 4 compared to Fig. 6.

Overall, when both detection tasks are compared, more features appear to be important for the malware recognition task, with the most important feature per quarter reporting lower absolute importance for the recall task. Therefore, the complexity of the recall task is deemed as more challenging and varied, less stable over time, and more susceptible to sudden concept drift. The observed changes in the last periods demonstrate that malware is more unpredictable than benign software in permissions usage.

To further explore these differences in important features, the results obtained for specificity and recall were compared for each quarter using *Wilcoxon signed-rank* test [77]. The statistical comparison, detailed in Appendix B, confirmed that the important features for the specificity task are not equally relevant for the recall task.

For the sake of completeness of the characterization analysis, Figs. 8 and 9 unveil the important features behind the grey areas in Figs. 4 and 6, respectively. More specifically, Fig. 8 provides the features belonging to the extended feature set that were found relevant in the analyzed time frame for the specificity task, while Fig. 9 provides the same information for the recall task. In these graphs, the features belonging to the reduced feature set are hidden behind the grey areas. Even though the importance of these features has been demonstrated to be significantly lower than the ones included in the reduced feature set, their analysis enables us to fully characterize the evolution of the importance of permissions for the whole analyzed time frame, from 2011 until 2018.

In the specificity case, depicted in Fig. 8, only 21 features added in later releases of Android OS were found relevant at some specific period in the analyzed time frame. As can be noted in Appendix C, the available permission set was refined in almost every Android release, extending the *available* permissions set to 157 permissions at the time of writing. More specifically, WRITE_EXTERNAL_STORAGE, BIND_DEVICE_ADMIN, and READ_EXTERNAL_STORAGE were identified as the most relevant features for specificity from the later additions to the feature set. Besides, Fig. 8 evidences locally emerging concept drift as some spe-

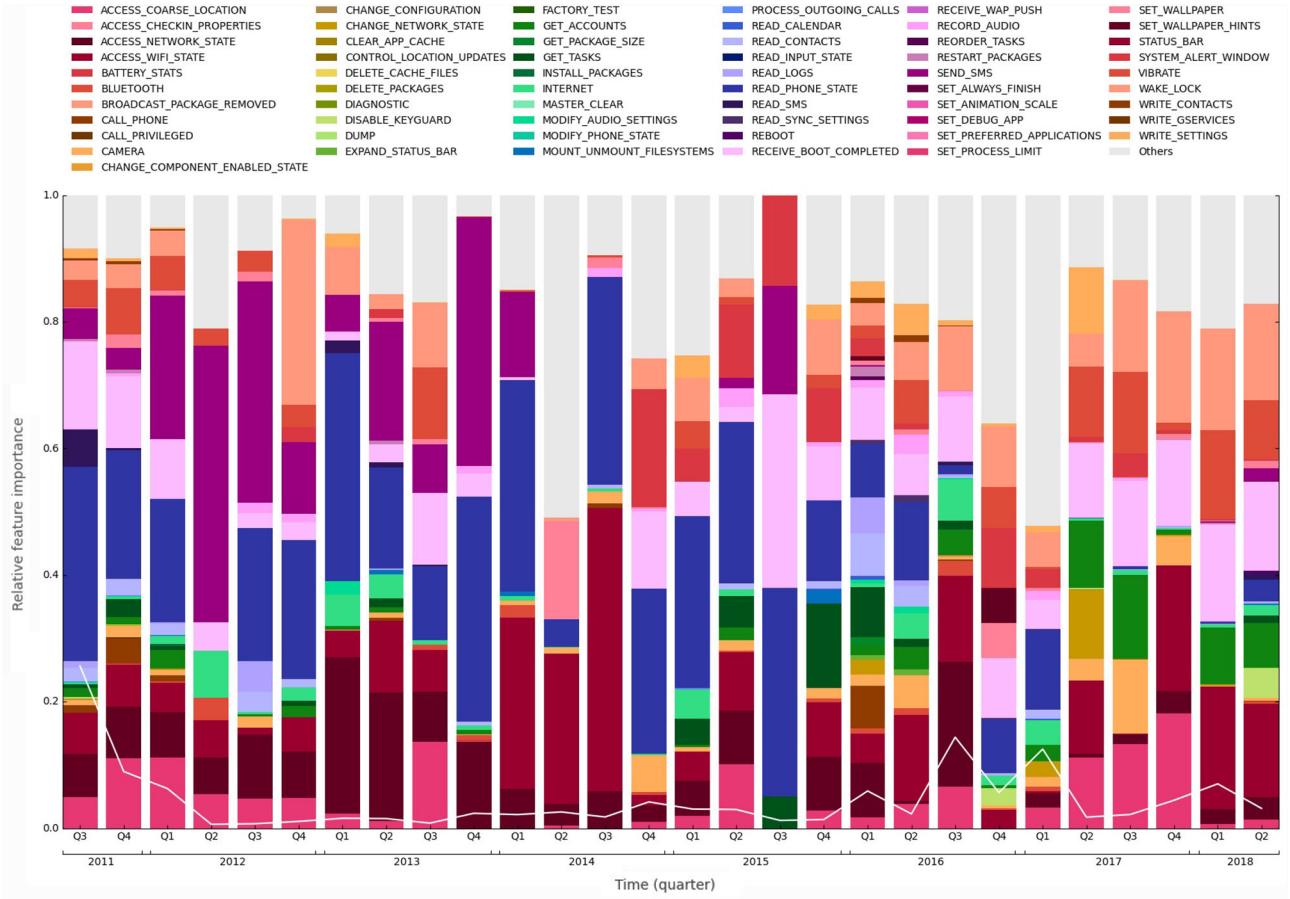


Fig. 6 Importance for recall, calculated for the extended feature set

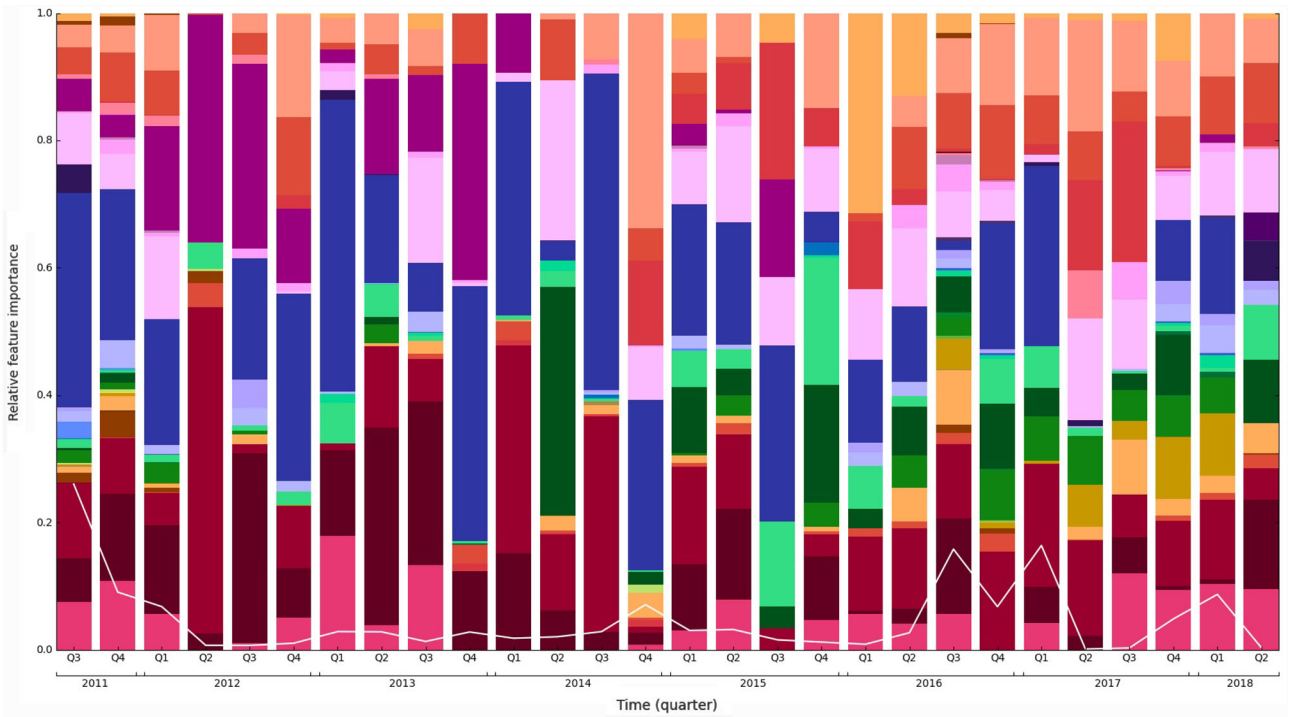


Fig. 7 Importance for recall, calculated for the reduced feature set

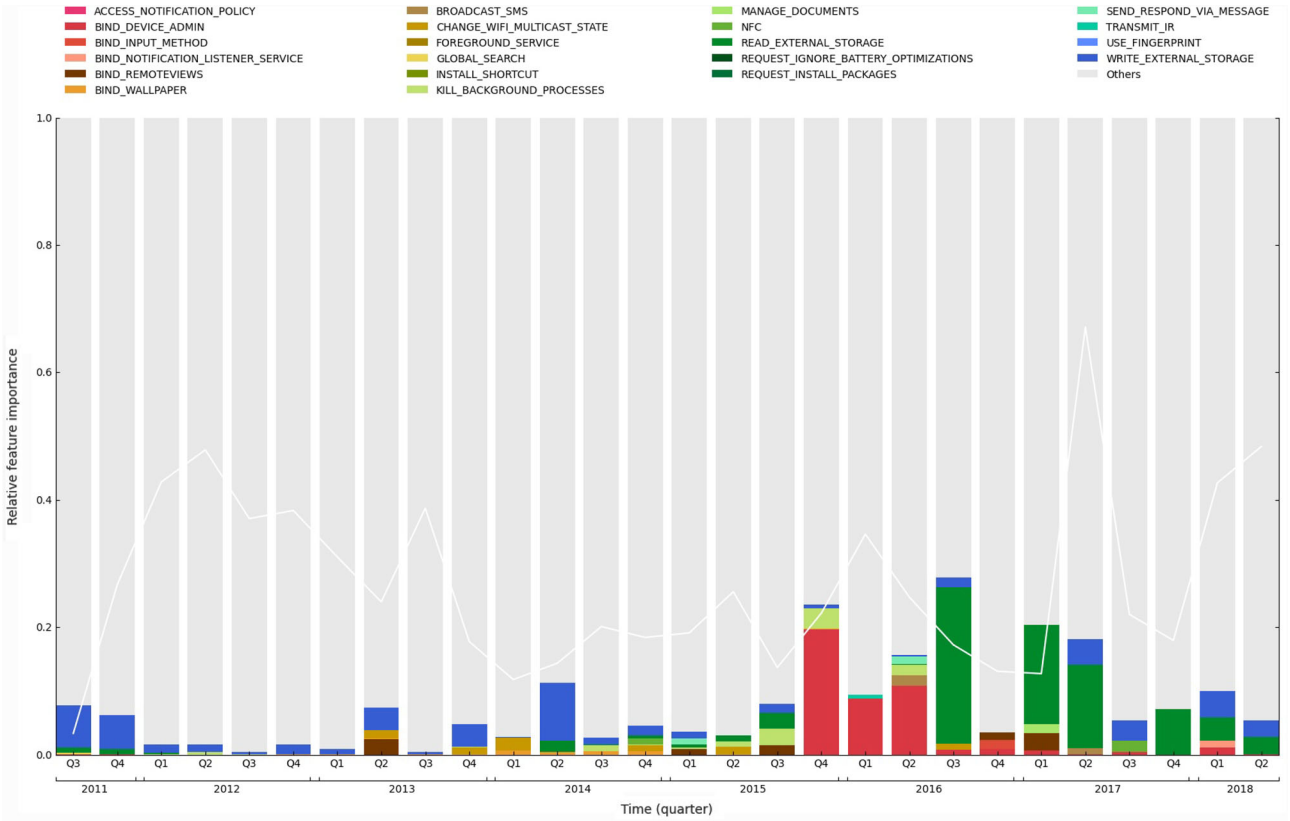


Fig. 8 Importance for specificity, showing only the extended features

cific features are relevant in some quarters but not globally (e.g., BIND_DEVICE_ADMIN from 2015-Q4 to 2016-Q2).

For the recall task, 27 features belonging to the extended feature set were found important at some point in the analyzed period, as provided in Fig. 9. More interestingly, in some cases, such as 2014-Q2 and 2017-Q1, an *extended* feature became the most important for the specific quarter. This fact was not observed for specificity and demonstrates the greater complexity of the malware recognition task, which is characterized by more sudden and, consequently, complex to handle concept drifts. In addition to the relevant features found for the specificity case, KILL_BACKGROUND_PROCESSES and BIND_WALLPAPER were of remarkable importance for the malware recognition task in specific quarters. Besides, WRITE_EXTERNAL_STORAGE showed an even more significant impact on recall, present in almost all quarters with distinctive relative importance.

Lastly, when Figs. 8 and 9 are compared, the observed concept drifts provide an interesting observation. For some prominent permissions, the features were found relevant for a task during a specific time frame, and then, after losing their importance for that task, they became relevant for the other task. For instance, BIND_DEVICE_ADMIN was important firstly for specificity (i.e., 2015-Q4 to 2016-Q2), and immediately after its importance vanished for this task, it emerged as an important feature for recall (i.e., 2016-Q3 to 2017-Q1). The opposite is observed for KILL_BACKGROUND_PROCESSES, becoming first relevant for recall and later for specificity.

In summary, the characterization analysis performed evidenced the critical importance of the initial set of permissions to build an effective recognition system, the lower relevancy for such a purpose of the later added permissions, and that even though concept drift issues were found in benign and malware data, the former shows relative stability with gradual changes being relatively easy to address, whereas the latter is characterized by more sudden, complex concept drifts dominated by specific features, making it harder to handle. Besides, the set and degree of importance of features differ for both tasks. Therefore, the analysis performed in this section evidences the dynamism and constantly evolving nature of the malware threat landscape and emphasizes the critical requirement to address concept drift for any solution aiming to provide long-lasting effective malware detection and adapt, updating its knowledge in an ever-evolving threat landscape. This constantly changing nature has been overlooked by all the proposed detection solutions using permissions in the related literature.

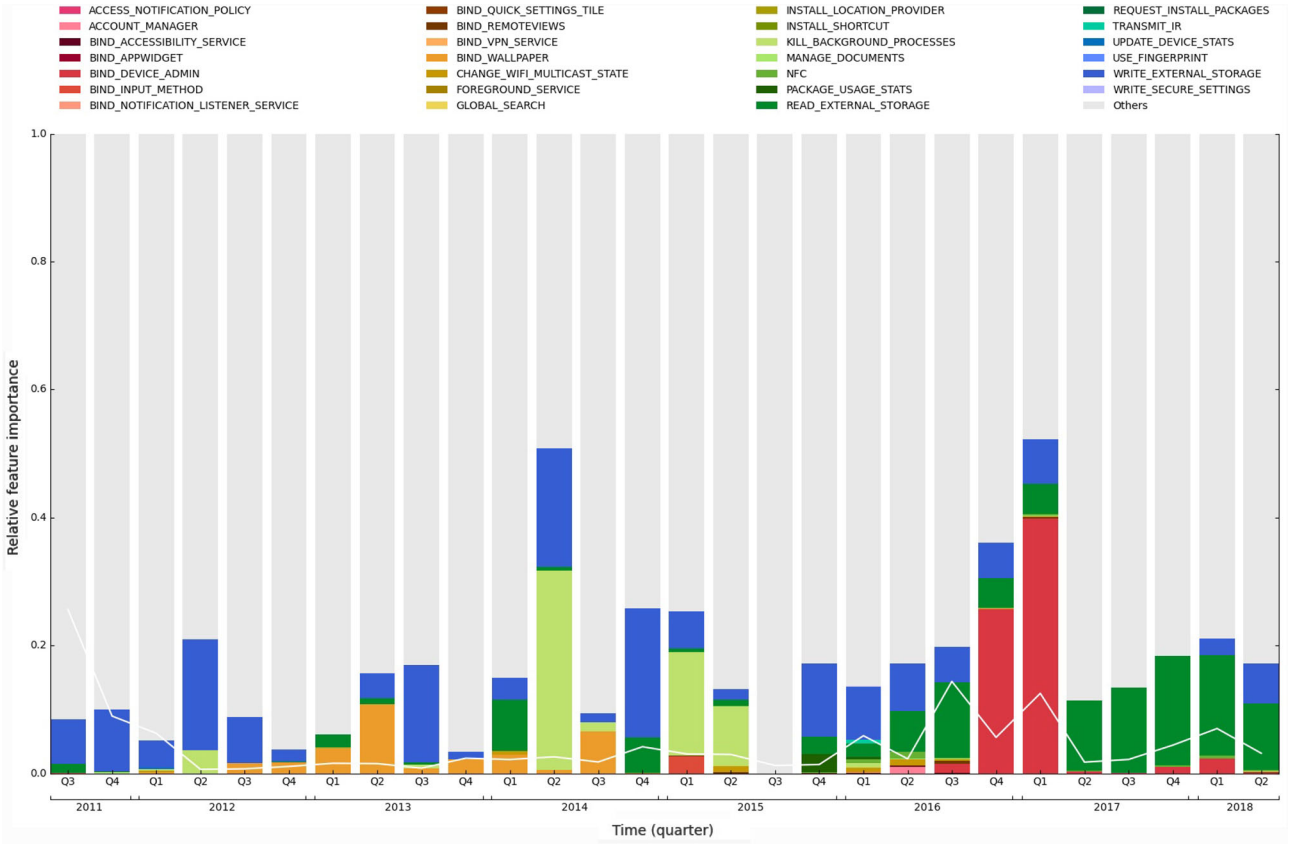


Fig. 9 Importance for recall, showing only the extended features

5.3.2 Class-based recognition performance

The F1 score metric, reported in Fig. 3, is a relevant performance metric to assess the effectiveness of malware detection solutions. However, class-based recognition comes in handy to assess the effectiveness of the detection solution to detect either of the classes and evaluate the degree of precision in the forecast of benign data.

The effectiveness of the model to detect each of the classes is provided in Fig. 10. This graph compares recall, f1 score, and specificity metrics for the reduced and extended feature sets.

More specifically, the red lines provide quarterly recall information for the full (i.e., red solid line) and reduced feature sets (i.e., red dashed line). The horizontal red lines provide the average recall performance for the whole time frame for both feature sets. Similarly, the green and light grey lines provide the same information for specificity and F1 score, respectively.

As can be noticed in Fig. 10, the model provided consistently high specificity quality (i.e., over 0.8) using either of the feature sets. This fact emphasizes the effectiveness of the permission-based model to recognize benign apps effectively over time and is consistent with the *smooth* concept drift that characterized these data.

However, the situation is notably different for the recall metric. Even though the average recall performance of the system shows an accuracy over 0.90, even reaching 0.99 in 8 periods, the malware recognition performance dips in two specific time frames, as evidenced by the F1 score and recall performance metrics in Fig. 10. The system failed to identify new malware samples effectively in those specific chunks but kept high benign software recognition performance. Figure 10 shows that the reduced feature set is more suitable for long-term accurate malware detection than the full feature set, as the average line for the full feature set is below the average line for the reduced feature set. More precisely, the average recall value for the reduced feature set is 91.7%, whereas for the extended feature set is 89.9%. Besides, the third dip was less severe when the reduced feature set was used. Therefore, for malware detection purposes, the reduced feature set is preferred.

The opposite situation happens in the case of benign software recognition, where the extended feature set provides better average performance than the reduced feature set. More specifically, an average of 91% specificity is obtained with the extended set, whereas for the reduced feature set,

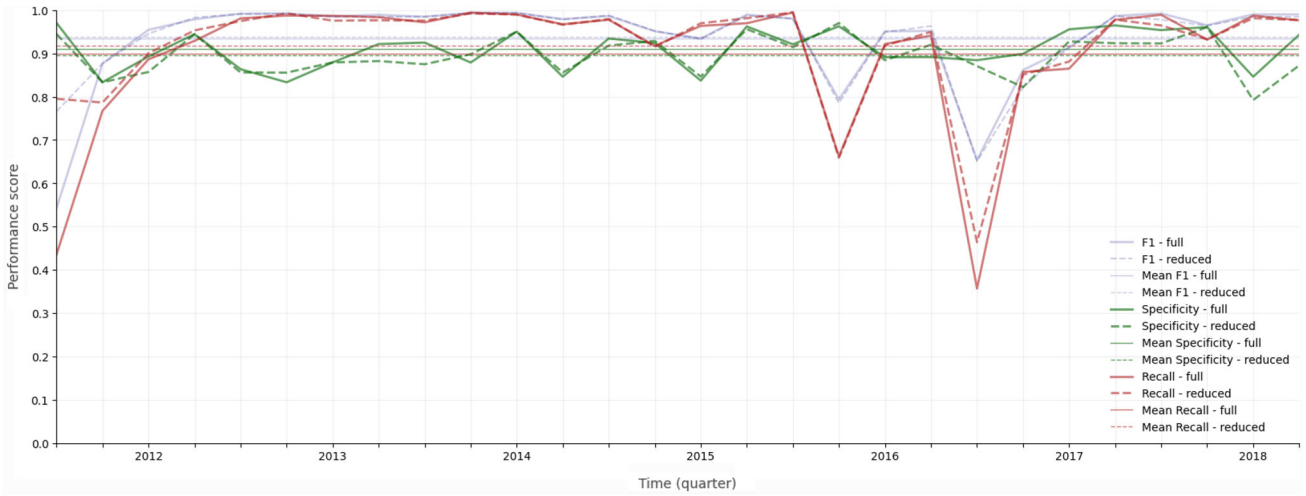


Fig. 10 Comparison of model's performance metrics using both feature sets

the average value stands at 89.5%. This observation is consistent with the findings in [53], where benign data samples were found to use a smaller but more varied set of permissions than malware apps. Consequently, the extended feature set, which includes more permissions, provides better overall performance for this task.

Despite the overall high performance of the detection system on both tasks, the system provided diminished malware detection performance, with different degrees of severity, at three specific time frames, namely, 2011-Q3 (initial period), 2015-Q4, and 2016-Q3. The cause and related insights behind these exceptional cases are described in the following paragraphs.

The first dip happens in the initial period, 2011-Q3. Even though this should not be considered a dip in performance, it is worth analyzing its cause. In the initialization phase of the system, where the model has access to a limited amount of data, the performance of the system strictly depends on the generalization capabilities of the initial data chunk regarding the following chunks/quarters. Therefore, the initial performance directly relates to the quality of the data in the initial chunk and not to the learning capabilities of the system, as there is no previous reference of detection performance. In our case, the initial chunk was split into n ordered data chunks, where n refers to the number of classifiers in the pool. The system was initialized in this initial quarter, building a distinct model with each data chunk and incorporating it into the pool. As a result, the low initial performance is likely caused by an insufficient variety of initial data to capture the phenomenon accurately. In this regard, the initial chunk was significantly dominated by benign apps (i.e., 97.25%), thus limiting the learning capabilities for an effective malware detection performance. The specificity performance tops in this initial quarter. However, even in this challenging learning situation, where the proportion of malware was very small (i.e., 3.75%), the system provided acceptable performance when the reduced feature set was used (i.e., 0.77 F1 score). The malware detection performance increased in the subsequent quarters, demonstrating the capabilities of the model to learn and adapt over time, even when a distinct combination of features emerged as important in close quarters (e.g., 2012-Q2). In the early quarters, the reduced set of permissions showed prominent importance, as evidenced by the high dependency of the extended model on the reduced set of permissions in the first 11 quarters, with importance of over 80% in all of them, as shown in Fig. 6.

The second and third performance drops correspond with actual concept drift.

The second performance drop, which is less severe than the third, is located in the fourth quarter of 2015. At first glance, the feature importance pattern looks similar to the previous data chunks. However, the distribution of 2015-Q4 changes significantly from the previous four chunks, with more features sharing the *important* status and a significant decrease in the domination of READ_PHONE_STATE, as can be observed in Figs. 6 and 7. In addition, the absolute importance value reaches a local minimum in that quarter, which emphasizes the fact that more features are important for the model, and none of them dominates significantly. This also includes the extended features (i.e., Fig. 9). An additional contributing factor might have been the special characteristics of the previous chunk, which shows a radically different importance distribution, dominated by a small number of features from the reduced feature set and no *extended* feature showing importance. Thus, all these changes in feature importance distribution evidence a light drift in this quarter's data concerning previous malware data, which can be seen as promoting factors behind this moderated performance dip.

This malware data shift is coincidental with the specificity score reaching its maximum value and being affected

by a decrease in the importance of previously important features, as shown in Fig. 4, and the outbreak of BIND_DEVICE_ADMIN as an important feature. This situation, where the model had to learn about significant changes in benign software, might have also impacted the recall performance at the expense of boosting specificity. Despite all these learning challenges, the F1 score of the model showed an acceptable performance (i.e., over 0.80), improving in the subsequent chunks.

The third drop is located in the third quarter of 2016. In this quarter, recall dips dramatically, especially for the extended feature set. At first sight, this period shows remarkably different characteristics from the previous ones. Firstly, it shows a significantly different feature distribution. In this period, 35 features were found important for the extended feature set (i.e., 16 belonging to the reduced set), whereas, in the previous quarters, fewer than 20 features were found important, on average. This relates to a change in data complexity, as the model had to learn and rely on more features for proper detection. Secondly, the dominant feature in the previous chunks, READ_PHONE_STATE, was relegated to a marginal role, and two features emerged as dominant features, ACCESS_NETWORK_STATE and ACCESS_WIFI_STATE. Besides, an increase in READ_EXTERNAL_STORAGE, belonging to the extended feature set, was also observed. The emergence of new important features correlated with an increase in absolute importance value, reaching a local maximum, as depicted by the overlay line. Thirdly, BIND_DEVICE_ADMIN diminished its importance significantly for specificity and emerged as important for recall for the first time, as shown in Fig. 9. Lastly, the sudden concept drift in recall is combined with a significant drift in specificity as new features irrupt as relevant with remarkable importance values in the reduced feature set (i.e., INTERNET and MOUNT_UNMOUNT_FILESYSTEMS) and the extended feature set (i.e., READ_EXTERNAL_STORAGE), as shown in Figs. 4 and 8, respectively.

In this case, for both recall and specificity, the changes were too sudden, unexpected, and distinct from previous data patterns for the classification pool to be able to deal with the new data characteristics properly. However, after this sudden and severe decrease in the recall performance, where the system still kept acceptable recognition capabilities for benign software, the pool adapted and learned from the new data, improving and recovering past performance levels in the subsequent chunks. No significant dip in performance was observed again.

In summary, the analysis performed in this subsection enabled us to correlate the quarterly characterization performed in the previous subsection with class-based performance results and find the rationale behind the decrease in performance in some specific quarters. The analysis evidenced that concept drift handling of benign data is an easier task due to the smoother feature variability in these data provoking a gradual concept drift and that sudden drifts in feature importance observed for malware data correlate with performance dips. The more dramatic and unexpected the changes, the more severe the decrease in performance observed. This fact is augmented when the extended feature set is used. The reduced feature set shows improved performance and less severe dips for malware data. Regarding specificity, even though both feature sets work well, the more varied usage of features by benign applications makes the extended feature set occasionally outperform the reduced feature set.

5.3.3 Malware family evolution analysis

This subsection aims to explore the third dip in detail, seeking the etiology of the unprecedented complexity observed in this quarter and the significant decrease in performance that occurred.

Android malware is constantly evolving, and this is reflected in the prevalence of malware families over time. Figure 11 shows the distribution of the top 10 malware families per quarter in the time frame from 2011-Q3 to 2018-Q2. The graph shows the prevalence of 54 malware families over time, indicated using specific colors for each one. Besides, the graph reports the number of malware families per period (i.e., white stars in the middle of each bar, whose values are related to the right vertical axis). The F1 performance of the detection model is provided with a white dashed overlay line.

As can be noticed, the major dip happens in a quarter dominated by the *Slocker* malware family, the first and most relevant Android ransomware family (i.e., 68% of the samples in 2016-Q3). This fact suggests that the dip might have been directly caused by diminished ransomware detection capabilities. However, it is worth noticing that ransomware was also dominant in 2015-Q3 (i.e., 53% of the samples), and the detection model provided over 98% F1 score and higher recall, as shown in Fig. 10.

To better explore the phenomenon, one-class anomaly detection models were built for the most prevalent families. Instead of using the label as the concept of class to build the one-class anomaly models, the malware family was leveraged as the class construct. The main idea behind these *one-class* or, in our case, *one-family* anomaly models is as follows. If an effective one-class anomaly detector for a specific family can be built in a specific quarter (i.e., high performance on the training data), when tested with samples of the same malware family in subsequent chunks, it should be able to detect them properly, reporting high accuracy. As a result, if the family samples change their character (i.e., malware family evolution), the anomaly model would reflect it as a performance decrease. Otherwise, if malware does not change significantly, high accuracy is expected.

Fig. 11 Distribution of malware families per period

Local Outlier Factor (LOF) algorithm was used to build the *one-family* anomaly detection models. LOF allows detecting *dissimilar* data samples using local density deviation of data points with respect to their neighbors [78]. Figure 12 provides *one-family* anomaly models for 12 predominant families in the time frame ranging from 2011-Q3 to 2018-Q2. The initial models for each family were induced in different quarters when the malware family produced an outbreak, and they were tested with subsequent chunks until the malware family's presence vanished.

Figure 12 provides relevant insights about malware families' evolution, which are directly related to the performance dips and concept drift observed. Firstly, most malware families showed similar or the same character over time. This is especially evident in the first half of the analyzed time frame, where almost all malware families showed the same features over time, as evidenced by the high accuracy values provided by each initial anomaly model tested with the subsequent quarters. It implies that most malware families do not evolve much regarding permissions over time, thus making these features powerful discriminators. Secondly, and more interestingly, the *Slocker* family did not follow that pattern. The initial model built for the *Slocker* family dips significantly in 2016-Q3, showing a dramatic and sudden change in characteristics concerning the initial model samples. The experimental findings are confirmed by [79], which reported that in the second half of 2016, over thrice Slocker variants were detected in comparison with the same period in 2015, and by [80], which reported about a recursive ransomware outbreak characterized by evolution into a more sophisticated and diverse malware family. This increase in variety and sophistication was already suggested by the more complex and diverse quarter characterization depicted in Fig. 7.

Additional evidence of the diversification of the *Slocker* ransomware family is provided in Fig. 13. Figure 13a provides individual predictions' *explanations* for samples belonging to the *Slocker* family in 2015-Q3, whereas Fig. 13b provides the same information for *Slocker* samples belonging to 2016-Q3. The comparison between the 2015-Q3 and 2016-Q3 samples is significant due to their similarity with regards to the large dominance of this family in these quarters and the fact that they yielded notably distinct detection performance. *Shapley values* are leveraged in these graphs to explain the reasoning behind individual predictions by the

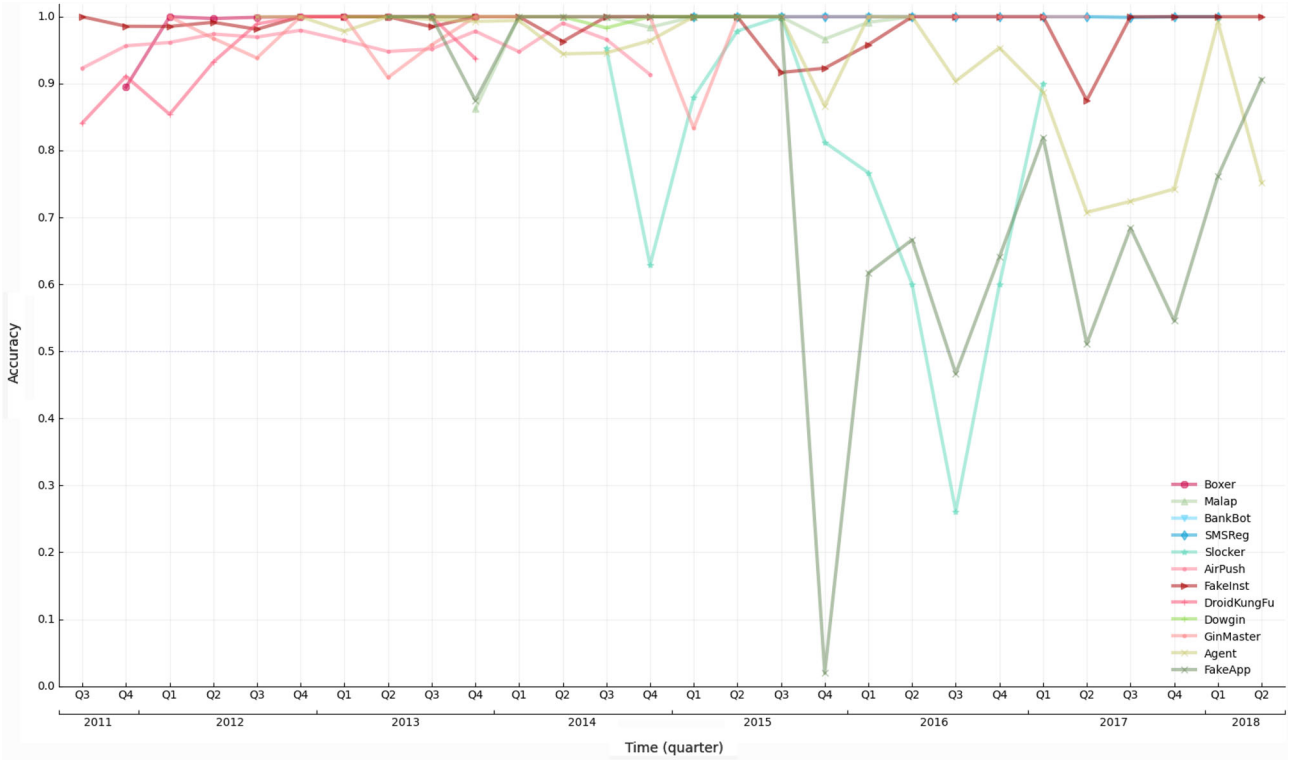
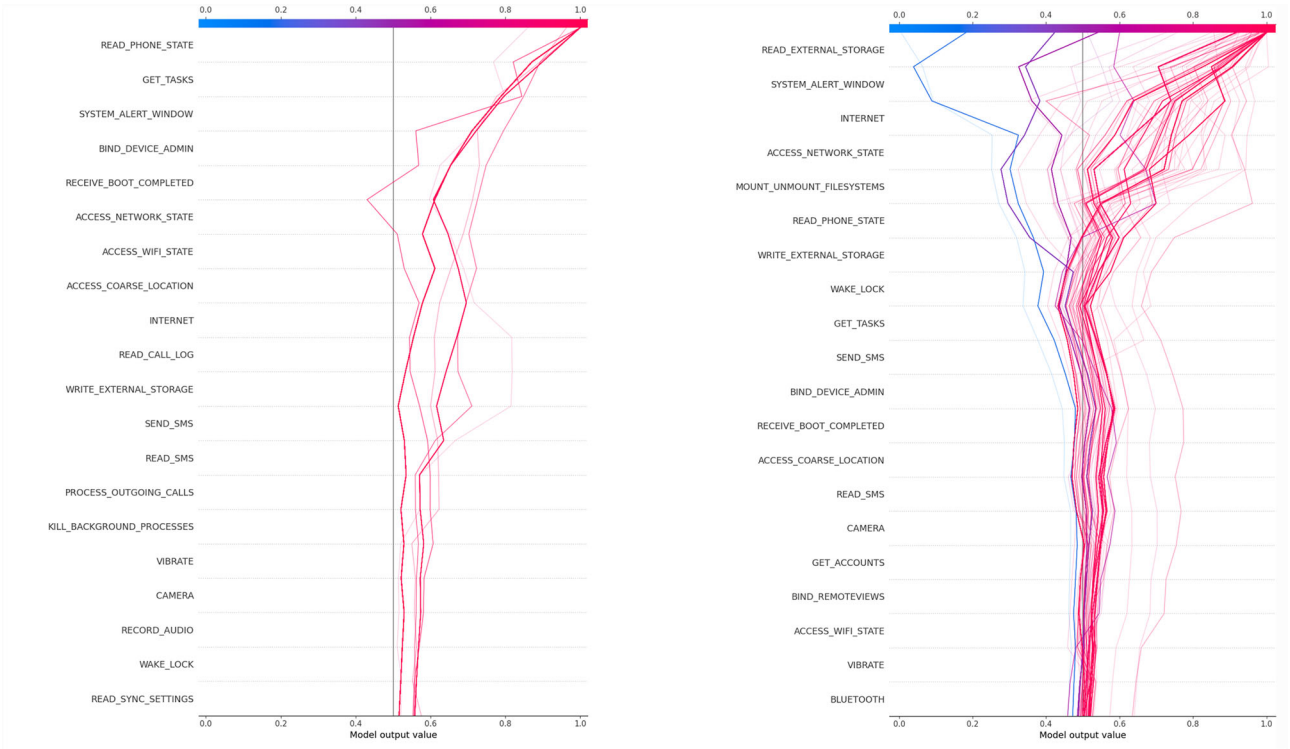


Fig. 12 Performance of *one-family* anomaly detection models



(a) 2015-Q3 Slocker predictions decision paths

(b) 2016-Q3 Slocker predictions decision paths

Fig. 13 *Slocker* family 2015-Q3 and 2016-Q3 predictions' decision paths

model. The graphs should be interpreted as follows. The initial probability of a sample belonging to the malware class is 0.5. As the data set is balanced, it reflects the expected value. On the left vertical axis, and from bottom to top, the features are ordered in increasing importance for the final decision made by the classifier, specified as prediction probability on the top part of the graph. The range, from 0 to 1, refers to the malware class probability. The lines or *decision paths* starting from the bottom, and reaching a prediction probability in the top part, give a notion of the impact of the features to classify each sample regarding the final probability. As all the data samples used to construct these graphs were Slocker samples, the expected classification result was over 0.5 in all cases (i.e., the model identified it as malware). The color of the line is based on the final prediction, highlighting the paths leading to the *malware* prediction (i.e., red) or the *no malware* prediction (i.e., blue). The more intense the line color, the larger the number of samples that followed that decision path. Both graphs use the extended feature set models as they enable us to fully understand the differences between the important features in each case.

As can be seen in Fig. 13a, all ransomware samples were correctly classified by the model, and the explanations are similar for all samples, with just six decision paths explaining all of them. Therefore, the ransomware samples in this quarter show similar characteristics, and consequently, similar features are used to classify them. On the contrary, Fig. 13b shows numerous paths used to classify the ransomware samples, with varying features importance, and many misclassified samples reach the no malware outcome. Therefore, Slocker samples from 2015-Q3 are significantly distinct from 2016-Q3 samples. The samples from 2016-Q3 are more varied, showing many different characters, and pose a more complex threat than the initial Slocker samples. In this regard, the differences in important feature sets on the explanations provide useful insights into the changes in behavior and main characteristics of these samples. For instance, the most important permission to discriminate 2016-Q3 ransomware was READ_EXTERNAL_STORAGE, which provides read access to files outside the app-specific directory and the media store (i.e., probably to access sensitive information and send it to the C&C server), whereas, in 2015-Q3, the ransomware samples were more concerned about the running tasks and phone general information. These plots evidence the significant change in the Slocker family, disguised in many forms to avoid detection [80], and, consequently, the added complexity for effective discrimination in the 2016-Q3 quarter.

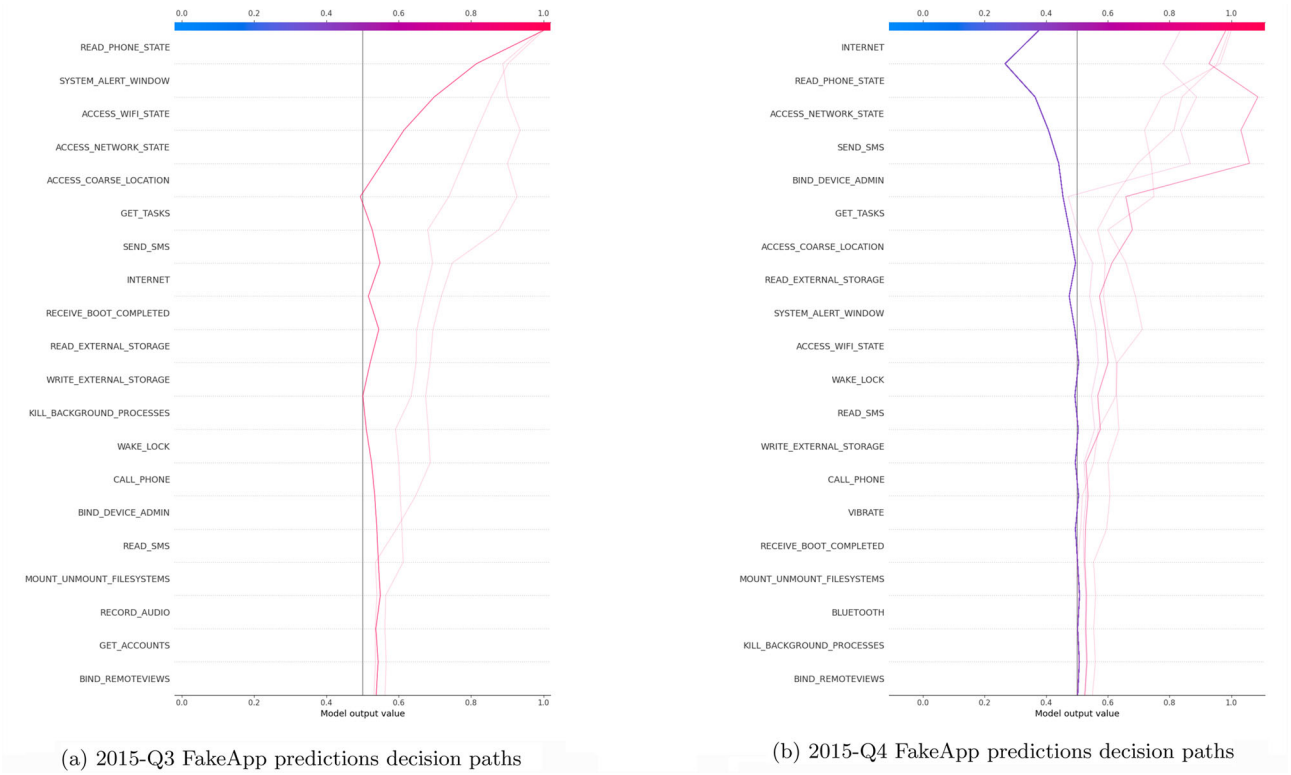
Lastly, Fig. 12 also provides relevant insights about the reasons behind the second performance dip, located in 2015-Q4. As can be observed, the quarterly malware distribution graph, depicted in Fig. 11, shows a significant increase and domination of the *FakeApp* family (i.e., 29.73%) in this period. At the same time, the related *one-family* anomaly detection model for that family shows that almost all samples for that quarter were detected as *anomalies*, thus indicating an *extreme* change in this malware family.

Similar to the ransomware case, Fig. 14 shows the decision paths for the *FakeApp* samples belonging to 2015-Q3 and 2015-Q4. More precisely, Fig. 14a provides explanations for *FakeApp* samples belonging to 2015-Q3, whereas Fig. 14b provides the same information for the *FakeApp* samples from 2016-Q4. As can be observed, a similar but more extreme outcome than in the ransomware case is depicted in Fig. 14, where the majority of data samples belonging to 2015-Q4 were misclassified. However, they showed consistent characteristics, and therefore only a few decision paths are observed.

The malware family analysis performed in this subsection enabled us to compare the performance dips and the changes in the malware threat landscape. It has been shown that malware family evolution is behind the performance dips and that sudden outbreaks and evolution of specific malware families explain the results observed in previous sections.

6 Discussion

The present study focused on the analysis of permissions evolution over time and its significant impact on malware classifiers built using these data. The proposed method, built exclusively using permissions and an ensemble of classifiers selected from a pool, allowed it to *learn* and adapt to concept drift, providing high performance over an extended period. Despite the decrease in performance observed in 2015-Q4 and 2016-Q3, an average F1 score of 0.93 was achieved in the 2011-Q3 to 2018-Q2 time frame. This consistent high performance emphasizes the effectiveness of the proposed solution to deal with Android malware concept drift using permissions as sample descriptors. No prior solution using just permissions has addressed the concept drift issue. Therefore, this study, which directly tackles this issue, constitutes a novelty in the field with no comparative reference. The studies in the literature that used permissions before, achieving high-performance metrics, were built and tested using small and *outdated* snapshots of data from Android history. Only [17] included long-term data (i.e., 2010-2016) achieving over 0.90 F1 score. However, the usage of random selection to generate the training/testing sets, thus neglecting concept drift, poses severe concerns about the generalization capabilities of this detection system. Our study shows that if the concept drift issue is addressed, permissions alone can provide long-term effective malware discriminatory capabilities. This allows leveraging their security-related implications to enhance and *understand* the detection systems. Notwithstanding that, the usage of permissions with



(a) 2015-Q3 FakeApp predictions decision paths

(b) 2015-Q4 FakeApp predictions decision paths

Fig. 14 *FakeApp* family 2015-Q3 and 2015-Q4 predictions' decision paths

additional dynamic and static features may enhance the detection capabilities of the proposed system, especially in those time frames where permissions alone may struggle to detect effectively under sudden concept drift (e.g., 2016-Q3). This constitutes part of our future work.

The experimentation performed in this research has shown that the initial set of permissions, released with the first version of Android OS, is currently the most relevant feature set to discriminate malware apps. The later additions to the permissions set, while showing relatively high local importance in specific periods, do not show the powerful and long-term consistent discriminatory capabilities provided by the initial permission feature set. This fact has remarkable implications in the generation of permissions-based effective discriminatory models as there is no *actual* need to update the feature set after every release, so the maintenance requirements of the detection system diminish significantly. Even though this may change in the future, as the initial set of permissions covers core security aspects (e.g., access to sensitive data), if they are not redefined or substituted in future API releases, they might retain their discriminatory power and be an essential part of the arsenal for effective malware detection. Besides, the *curse of dimensionality*, a degenerative performance issue related to ever-growing feature sets in machine learning models, is avoided when a small feature set is used. Furthermore, permissions are the easiest static features to collect and are inherently *interpretable* as they have security-related implications behind them. This fact makes the machine learning-based detection models based on permissions computationally efficient, and their outcomes easily explainable and traced using *post-hoc* explainers or interpretable machine-learning models.

The analysis of the importance of features and their relation to malware distribution has provided relevant insights to explain the dips in performance of the proposed permissions-based detection system. Permutation feature importance and Shapley values have been used to provide permissions characterization over time and a better understanding of the decision outcomes, respectively, thus providing useful insights about malware evolution for its effective detection. None of the previous permission-based proposed solutions took into account concept drift in their models nor provided any thorough characterization of permissions evolution over time.

7 Limitations and threats to validity

The limitations that may challenge the generalization of the results are centered around the data set, model performance, and interpretation methods used.

The quality of the data is critical to building effective machine learning-based detection systems. Thus, for an accurate malware detection solution to perform effectively, the data used should be representative of the actual situation. This is especially critical for production setups. The data set used, *KronoDroid*, provides a good approximation to the malware threat landscape in the period studied but may have limitations regarding the temporal accuracy, which is critical for proper concept drift study, and the quantity of data. These facts may threaten the external validity of the experimentation. However, no other data set for Android research includes timestamps nor suits for the evaluation of long-term evolution. Furthermore, the naming for malware families is not homogeneous among vendors, which may impact the labeling and, consequently, the range of distinct behaviors included under the same family label. Despite that, the findings using *KronoDroid* data have shown a correlation with the actual Android historical timeline events, thus showing the goodness and reliability of the data set.

Regarding the model performance, the time frame used to split the data (i.e., quarters) was found experimentally, based on the characteristics of the data set. The solution provided high performance for an extended period. However, two dips in performance were observed due to sudden concept drift events. In a real-world setup, where data is constantly flowing, the time constraints should be narrowed down (i.e., experimentally find the optimal chunk size/length), and the sudden concept drift would likely be handled more efficiently as *smaller* data chunks should minimize and smooth out the concept drift impact. Despite these limitations, the solution provided high specificity, recall, and F1 metrics for a seven-year-long time frame. They can be used as a reference of the capabilities of the system for production setups.

Finally, the interpretation methods used to characterize the concept drift and explain the individual decisions, which were permutation feature importance and Shapley values, have inherent limitations, especially when correlated features are included [75]. To minimize this issue, the correlation between features was analyzed in the data pre-processing step, and correlated features were removed. Despite that, no machine learning interpretation method is free from limitations, and the quality of the data used can also significantly impact the outcome [75]. Therefore, any interpretation method output should be carefully considered. In our case, the global interpretability method (i.e., permutation feature importance) and the local interpretability method (i.e., Shapley values) provided similar insights about the data, supporting our findings.

8 Conclusions

Permissions are *first-line* security constructs within the Android ecosystem. The analysis of their evolution and usage can provide relevant highlights about the malware scene, especially when analyzed long-term.

We leveraged all these features in this study. Our results show that when concept drift is addressed, permissions alone can be used to build a long-lasting, effective malware detection solution. An average F1 performance of over 0.93 was achieved in data that span seven years. Furthermore, permissions, which have been relegated to a secondary role in Android malware detection systems, can provide powerful insights about the behavior of apps, as they are inherently *comprehensible* features, a property that can be used to enhance detection systems and malware knowledge.

This research shows that a reduced number of features can be used to build an effective system, especially for the specific malware recognition task, and that there is no need to update the feature set after every Android release. The analysis of the evolution of permissions shows that malware is a complex phenomenon in constant evolution, so concept drift issues must be considered to keep high detection performance. However, clear patterns in permissions data are observed, especially for the benign recognition task, which can be leveraged to build more effective systems.

Family-based analysis of permissions evolution and its characterization is part of our future work.

Appendix A. Two-sample Kolmogorov-Smirnov statistical test

To formally compare the performance of both feature sets (i.e., extended vs. reduced), the non-parametric *Kolmogorov–Smirnov* test was used to analyze the equality of both probability distributions (i.e., two-sample K–S test). The test statistic quantifies the distance between the empirical distribution functions (EDF) of two data samples. In our case, the distribution of F1 score data is compared for the reduced and extended feature sets. The results show that with a p-value = 0.7634, the test cannot be the basis for H_0 rejection that the F1 score distributions come from the same distribution. Moreover, the K–S statistic value, which reflects the maximum distance between the EDFs, is relatively small (i.e., K–S = 0.04). The test cumulative density function (ECDF) is illustrated in Fig. 15. Thus, it can be concluded that there is no significant difference in detection performance between the compared feature sets.

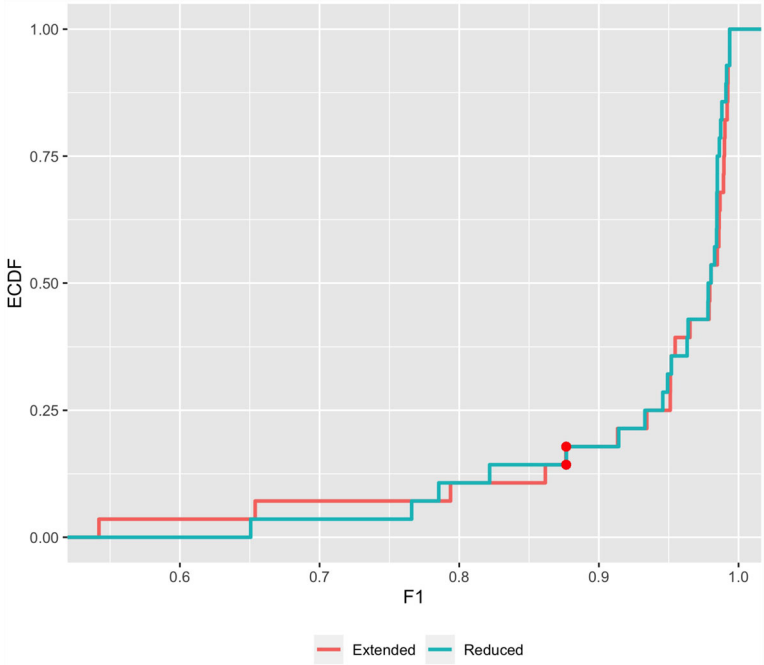


Fig. 15 Kolmogorov-Smirnov test to compare F1 for the reduced and extended feature sets. The maximum distance between EDFs is 0.04 (marked in the graph with red dots).

Appendix B. Wilcoxon signed-rank test

Wilcoxon signed-rank test is a non-parametric statistical hypothesis test (i.e., no normality distribution is assumed) that enables the comparison of two data distributions (i.e., specificity and recall in this case). However, the test requires the same amount of data on both distributions and, in our particular case, as not all features were found important in all periods, there were missing values in quarters regarding specific features. The quantity of missing values is critical. The data for the reduced vector showed nearly 70% missing values while reaching 83% for the extended vector. Therefore, the statistical analysis was performed only on the reduced feature set data. In this regard, as the negative values of function (4) are unknown but are needed for the test, a solution could be replacing missing values with zero values. Yet, the large missing value ratio might bias the comparison results, groundlessly increasing the similarity of the vector with a large number of missing values. Therefore, a better approach is to replace the missing values with the mean importance of the feature, calculated for the given data set and taken as a negative value. Thus, vectors are compared using means when the number of missing values is high and using the distribution of importance otherwise.

The results of the Wilcoxon test are reported in Table 2, calculated for the reduced feature set, and used to distinguish permissions important to optimize specificity and recall tasks. The table reports the features with a p-value less than 0.005, which suggests a highly significant difference between the compared vectors. The occurrence refers to the number of not missing values in the compared vectors (i.e., the number of times the feature was found important in recall or specificity). The maximum and total importance are provided for each feature. The maximum importance reflects the maximum value of importance the feature had in any quarter, while the total importance is the cumulative importance of the specific feature in all quarters. The features are ordered based on the completeness of the data vectors (i.e., from a larger number of occurrences to a smaller number).

As reported in Table 2, among the 44 shared features found important for both recognition tasks, 29 showed statistically significant differences with a p-value < 0.005. Among these important features showing significantly distinct importance distributions for malware detection and benign software recognition, there are relevant concept drift-related features such as READ_PHONE_STATE, SEND_SMS, and MOUNT_UNMOUNT_FILESYSTEMS. Apart from the large total importance, all these features have high occurrence and the largest maximum importance values. In all cases, the obtained importance is significantly higher for specificity. Therefore, based on this table, it can be concluded that important features for benign app recognition (i.e., specificity) are not equally relevant for the malware recognition task (i.e., recall).

Appendix C. Android permission set evolution

Table 3 provides the modifications that changed the available set of Android security permissions over time. This table gives a notion of the dynamic character of the permission set

Table 2 Significantly different ($p < 0.005$) features used to maximize recall and specificity for the reduced feature set

	Permission	Importance		Occurrences
		Total	Max	
1	READ_PHONE_STATE	5.0541	0.5246	51
2	SEND_SMS	7.7330	0.6100	44
3	GET_TASKS	0.8402	0.1256	43
4	SYSTEM_ALERT_WINDOW	0.6956	0.1083	42
5	READ_SMS	0.8039	0.1488	28
6	MOUNT_UNMOUNT_FILESYSTEMS	3.1416	0.7879	25
7	READ_SYNC_SETTINGS	0.0510	0.0191	19
8	MODIFY_AUDIO_SETTINGS	0.0310	0.0214	16
9	BATTERY_STATS	0.0280	0.0126	14
10	READ_CALENDAR	0.0075	0.0032	13
11	CHANGE_CONFIGURATION	0.0215	0.0141	12
12	INSTALL_PACKAGES	0.0090	0.0052	12
13	PROCESS_OUTGOING_CALLS	0.0208	0.0193	10
14	GET_PACKAGE_SIZE	0.0114	0.0091	9
15	CALL_PRIVILEGED	0.0008	0.0008	8
16	CLEAR_APP_CACHE	0.0047	0.0046	6
17	WRITE_CONTACTS	0.0192	0.0085	6
18	ACCESS_CHECKIN_PROPERTIES	0.0003	0.0003	4
19	MODIFY_PHONE_STATE	0.0009	0.0006	4
20	REBOOT	0.0010	0.0010	4
21	RECEIVE_WAP_PUSH	0.0050	0.0047	4
22	SET_WALLPAPER_HINTS	0.0022	0.0021	4
23	DELETE_CACHE_FILES	0.0010	0.0009	3
24	DELETE_PACKAGES	0.0001	0.0001	3
25	EXPAND_STATUS_BAR	0.0053	0.0048	3
26	SET_PREFERRED_APPLICATIONS	0.0250	0.0250	3
27	READ_INPUT_STATE	0.0000	0.0000	1
28	SET_DEBUG_APP	0.0000	0.0000	1
29	STATUS_BAR	0.0823	0.0823	1

and its evolution throughout the whole Android lifetime. The permissions set was first defined for API level 1 (i.e., the first release of Android) and has been constantly modified since then. Table 3 provides the evolution of the *available permission set* from API level 1 to API level 30. The table is ordered chronologically based on API level release (i.e., from the oldest to the latest), and it provides API level-related information such as release date (i.e., *Date*) and *OS version* name. For each API level (i.e., rows), the added and deprecated permissions in the corresponding API level are provided in the *Added Permissions* and *Deprecated Permissions* columns. Furthermore, for each added permission, the protection level is provided in the column *Type*. Three protection levels are defined: *dangerous*, *normal* and *others*, reported as D, N and O, respectively. The *others* category refers to permissions that can be requested by third-party apps and that do not belong to the dangerous or normal category, as defined in the official documentation [21]. If the permission cannot be used by third-party apps, it is referenced with a hyphen (-). For each deprecated permission, the API level that introduced the permission is provided within parenthesis. Lastly, the column *Set* reports the number of available permissions (i.e., excluding deprecated) in each API level.

Table 3 Android permission feature set evolution

API	Date	OS version	Added permission	Type	Deprecated permission	Set
1	09-2008	1.0	ACCESS_COARSE_LOCATION	D		74
			ACCESS_FINE_LOCATION	D		
			ACCESS_CHECKIN_PROPERTIES	–		
			ACCESS_LOCATION_EXTRA_COMMANDS	N		
			ACCESS_NETWORK_STATE	N		
			ACCESS_WIFI_STATE	N		
			BATTERY_STATS	O		
			BLUETOOTH	N		
			BLUETOOTH_ADMIN	N		
			BROADCAST_PACKAGE_REMOVED	–		
			BROADCAST_STICKY	N		
			CALL_PHONE	D		
			CALL_PRIVILEGED	–		
			CAMERA	D		
			CHANGE_COMPONENT_ENABLED_STATE	–		
			CHANGE_CONFIGURATION	O		
			CHANGE_NETWORK_STATE	N		
			CHANGE_WIFI_STATE	N		
			CLEAR_APP_ACHE	O		
			CONTROL_LOCATION_UPDATES	–		
			DELETE_CACHE_FILES	O		
			DELETE_PACKAGES	–		
			DIAGNOSTIC	–		
			DISABLE_KEYGUARD	N		
			DUMP	–		
			EXPAND_STATUS_BAR	N		
			FACTORY_TEST	–		
			GET_ACCOUNTS	D		
			GET_PACKAGE_SIZE	N		
			GET_TASKS	N		
			INSTALL_PACKAGES	–		
			INTERNET	N		

Table 3 continued

API	Date	OS version	Added permission	Type	Deprecated permission	Set
			MASTER_CLEAR	–		
			MODIFY_AUDIO_SETTINGS	N		
			MODIFY_PHONE_STATE	–		
			MOUNT_UNMOUNT_FILESYSTEMS	–		
			PERSISTENT_ACTIVITY	D		
			PROCESS_OUTGOING_CALLS	D		
			READ_CALENDAR	D		
			READ_CONTACTS	D		
			READ_INPUT_STATE	–		
			READ_LOGS	–		
			READ_PHONE_STATE	D		
			READ_SMS	D		
			READ_SYNC_SETTINGS	N		
			READ_SYNC_STATS	N		
			REBOOT	–		
			RECEIVE_BOOT_COMPLETED	N		
			RECEIVE_MMS	D		
			RECEIVE_SMS	D		
			RECEIVE_WAP_PUSH	D		
			RECORD_AUDIO	D		
			REORDER_TASKS	N		
			RESTART_PACKAGES	D		
			SEND_SMS	D		
			SET_ALWAYS_FINISH	–		
			SET_ANIMATION_SCALE	–		
			SET_DEBUG_APP	–		
			SET_PREFERRED_APPLICATIONS	D		
			SET_PROCESS_LIMIT	–		

Table 3 continued

API	Date	OS version	Added permission	Type	Deprecated permission	Set
			SET_TIME_ZONE	–		
			SET_WALLPAPER	N		
			SET_WALLPAPER_HINTS	N		
			SIGNAL_PERSISTENT_PROCESSES	–		
			STATUS_BAR	–		
			SYSTEM_ALERT_WINDOW	O		
			VIBRATE	N		
			WAKE_LOCK	N		
			WRITE_APN_SETTINGS	–		
			WRITE_CALENDAR	D		
			WRITE_CONTACTS	D		
			WRITE_GSERVICES	–		
			WRITE_SETTINGS	O		
			WRITE_SYNC_SETTINGS	N		
2	02-2009	1.1 Petit Four	BROADCAST_SMS	–		76
			BROADCAST_WAP_PUSH	–		
3	04-2009	1.5 Cupcake	BIND_APPWIDGET	–		81
			BIND_INPUT_METHOD	O		
			MOUNT_FORMAT_FILESYSTEMS	–		
			UPDATE_DEVICE_STATS	–		
			WRITE_SECURE_SETTINGS	–		
4	09-2009	1.6 Donut	CHANGE_WIFI_MULTICAST_STATE	N		85
			GLOBAL_SEARCH	O		
			INSTALL_LOCATION_PROVIDER	–		
			WRITE_EXTERNAL_STORAGE	D		
5	10-2009	2.0 Eclair	ACCOUNT_MANAGER	–		86
6	12-2009	2.0.1 Eclair				86
7	01-2010	2.1 Eclair				86
8	05-2010	2.2 Froyo	BIND_DEVICE_ADMIN	O		90
			BIND_WALLPAPER	O		
			KILL_BACKGROUND_PROCESSES	N		
			SET_TIME	–		

Table 3 continued

API	Date	OS version	Added permission	Type	Deprecated permission	Set
9	12-2010	2.3 Gingerbread	NFC	N		93
			SET_ALARM	N		
			USE_SIP	D		
10	02-2011	2.3.3 Gingerbread				93
11	02-2011	3.0 Honeycomb	BIND_REMOTEVIEWS	O		94
12	05-2011	3.1 Honeycomb				94
13	07-2011	3.2 Honeycomb				94
14	10-2011	4.0 Ice Cream Sandwich	ADD_VOICEMAIL	D		97
			BIND_TEXT_SERVICE	O		
			BIND_VPN_SERVICE	O		
15	12-2011	4.0.3 Ice Cream Sandwich			PERSISTENT_ACTIVITY (1)	94
					RESTART_PACKAGES (1)	
					SET_PREFERRED_APPLICATIONS (1)	
16	07-2012	4.1 Jelly Bean	BIND_ACCESSIBILITY_SERVICE	O		97
		READ_CALL_LOG	D			
		READ_EXTERNAL_STORAGE	D			
		WRITE_CALL_LOG	D			
				READ_INPUT_STATE (1)		
17	11-2012	4.2 Jelly Bean				97
18	07-2013	4.3 Jelly Bean	BIND_NOTIFICATION_LISTENER_SERVICE	O		100
			LOCATION_HARDWARE	–		
			SEND_RESPOND_VIA_MESSAGE	–		
19	10-2013	4.4 KitKat	BIND_NFC_SERVICE	O		109
			BIND_PRINT_SERVICE	O		
			BLUETOOTH_PRIVILEGED	–		
			CAPTURE_AUDIO_OUTPUT	–		
			INSTALL_SHORTCUT	N		
			MANAGE_DOCUMENTS	–		
			MEDIA_CONTENT_CONTROL	–		
			TRANSMIT_IR	N		
			UNINSTALL_SHORTCUT	–		
20	06-2014	4.4W KitKat	BODY_SENSORS	D		110

Table 3 continued

API	Date	OS version	Added permission	Type	Deprecated permission	Set
21	11-2014	5.0 Lollipop	BIND_DREAM_SERVICE	O		114
			BIND_TV_INPUT	O		
			BIND_VOICE_INTERACTION	O		
			READ_VOICEMAIL	O		
			WRITE_VOICEMAIL	O		
					GET_TASKS (1)	
22	03-2015	5.1 Lollipop	BIND_CARRIER_MESSAGING_SERVICE	O		115
23	10-2015	6.0 Marshmallow	ACCESS_NOTIFICATION_POLICY	N		125
			BIND_CARRIER_SERVICES	O		
			BIND_CHOOSER_TARGET_SERVICE	O		
			BIND_INCALL_SERVICE	O		
			BIND_MIDI_DEVICE_SERVICE	O		
			BIND_TELECOM_CONNECTION_SERVICE	O		
			GET_ACCOUNTS_PRIVILEGED	O		
			PACKAGE_USAGE_STATS	O		
			REQUEST_IGNORE_BATTERY_OPTIMIZATIONS	N		
			REQUEST_INSTALL_PACKAGES	O		
			USE_FINGERPRINT	N		
					BIND_CARRIER_MESSAGING_SERVICE (22)	
24	08-2016	7.0 Nougat	BIND_CONDITION_PROVIDER_SERVICE	O		129
			BIND_QUICK_SETTINGS_TILE	–		
			BIND_SCREENING_SERVICE	O		
			BIND_VR_LISTENER_SERVICE	O		
25	10-2016	7.1 Nougat				129
26	08-2017	8.0 Oreo	ANSWER_PHONE_CALLS	D		138
			BIND_AUTOFILL_SERVICE	O		
			BIND_VISUAL_VOICEMAIL_SERVICE	O		
			INSTANT_APP_FOREGROUND_SERVICE	O		
			MANAGE_OWN_CALLS	N		
			READ_PHONE_NUMBERS	D		
			REQUEST_COMPANION_RUN_IN_BACKGROUND	N		
			REQUEST_COMPANION_USE_DATA_IN_BACKGROUND	N		
			REQUEST_DELETE_PACKAGES	N		
27	12-2017	8.1 Oreo				138

Table 3 continued

API	Date	OS version	Added permission	Type	Deprecated permission	Set
28	08-2018	9.0 Pie	ACCEPT_HANDOVER	D		141
			FOREGROUND_SERVICE	N		
			NFC_TRANSACTION_EVENT	N		
			USE_BIOMETRIC	N		
					USE_FINGERPRINT (23)	
29	09-2019	10 Quince Tart	ACCESS_BACKGROUND_LOCATION	D		150
			ACCESS_MEDIA_LOCATION	D		
			ACTIVITY_RECOGNITION	D		
			BIND_CALL_REDIRECTION_SERVICE	O		
			BIND_CARRIER_MESSAGING_CLIENT_SERVICE	O		
			CALL_COMPANION_APP	N		
			REQUEST_PASSWORD_COMPLEXITY	N		
			SMS_FINANCIAL_TRANSACTIONS	O		
			START_VIEW_PERMISSION_USAGE	O		
			USE_FULL_SCREEN_INTENT	N		
					PROCESS_OUTGOING_CALLS (1)	
30	09-2020	11 Red Velvet Cake	BIND_CONTROLS	–		157
			BIND_QUICK_ACCESS_WALLET_SERVICE	O		
			INTERACT_ACROSS_PROFILES	O		
			LOADER_USAGE_STATS	O		
			MANAGE_EXTERNAL_STORAGE	O		
			NFC_PREFERRED_PAYMENT_INFO	N		
			QUERY_ALL_PACKAGES	N		
			READ_PRECISE_PHONE_STATE	O		
					BIND_CHOOSER_TARGET_SERVICE (23)	

References

1. Hautala, L.: Android malware tries to trick you. Here's how to spot it. https://www.cnet.com/tech/services-and-software/android-malware-tries-to-trick-you-heres-how-to-spot-it/ (2021)
2. Palmer, D.: Sophisticated android malware spies on smartphones users and runs up their phone bill too. https://www.zdnet.com/article/sophisticated-android-malware-spies-on-smartphones-users-and-runs-up-their-phone-bill-too/ (2018)
3. Yaswant, A.: New advanced android malware posing as "ystem update". https://blog.zimperium.com/new-advanced-android-malware-posing-as-system-update/ (2021)
4. O'Dea, S.: Mobile operating systems' market share worldwide from January 2012 to June 2021. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/ (2021)
5. Kaspersky: Can you get viruses on android? every android user is at risk. https://www.kaspersky.com/resource-center/preemptive-safety/android-malware-risk (2021)
6. Velzian, B.: Calling all threat hunters—mobile malware to look out for in 2021. https://www.wandera.com/calling-all-threat-hunters-mobile-malware-to-look-out-for-in-2021/ (2021)
7. Android: App permissions best practices. https://developer.android.com/training/permissions/usage-notes (2021)
8. Google: Google play protect. https://developers.google.com/android/play-protect (2021)
9. Samsung: This is protection, samsung knox. https://www.samsungknox.com/en/secured-by-knox (2021)
10. Withwam, R.: Android antivirus apps are useless—here's what to do instead. https://www.extremetech.com/computing/104827-android-antivirus-apps-are-useless-heres-what-to-do-instead (2020)
11. Lakshmanan, R.: Joker malware apps once again bypass Google's security to spread via play store. https://thehackernews.com/2020/07/joker-android-mobile-virus.html (2020)
12. Chebyshev, V.: Mobile malware evolution 2020. https://securelist.com/mobile-malware-evolution-2020/101029 (2021)
13. Faruki, P., Ganmoor, V., Laxmi, V., Gaur, M.S., Bharmal, A.: Androsimilar: robust statistical feature signature for android malware detection. In: Proceedings of the 6th International Conference on Security of Information and Networks, pp. 152–159 (2013)
14. Feizollah, A., Anuar, N.B., Salleh, R., Wahab, A.W.A.: A review on feature selection in mobile malware detection. Digit. Investig. **13**, 22–37 (2015)
15. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C.: Drebin: effective and explainable detection of android malware in your pocket. In: Ndss, vol. 14, pp. 23–26 (2014)
16. Lipovský, R., Štefanko, L., Braniša, G.: The rise of android ransomware. https://www.welivesecurity.com/wp-content/uploads/2016/02/Rise_of_Android_Ransomware.pdf (2016)
17. Mathur, A., Podila, L.M., Kulkarni, K., Niyaz, Q., Javaid, A.Y.: Naticusdroid: a malware detection framework for android using native and custom permissions. J. Inf. Secur. Appl. **58**, 102696 (2021)
18. Khariwal, K., Singh, J., Arora, A.: Ipdroid: android malware detection using intents and permissions. In: 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), pp. 197–202. IEEE (2020)
19. Android: Request app permissions. https://developer.android.com/training/permissions/requesting (2021)
20. Android: Permissions on android. https://developer.android.com/guide/topics/permissions/overview (2021)
21. Android: Manifest.permission., https://developer.android.com/reference/android/Manifest.permission (2021)
22. Android: Define a custom app permission. https://developer.android.com/guide/topics/permissions/defining (2021)
23. Codepath: Understanding app permissions. https://guides.codepath.com/android/Understanding-App-Permissions (2021)
24. Android: App permissions best practices. https://developer.android.com/training/permissions/usage-notes (2021)
25. Android: Permissions updates in android 11. https://developer.android.com/about/versions/11/privacy/permissions (2021)
26. JR, R.: Android versions: A living history from 1.0 to 12. https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html (2021)
27. Milosevic, N., Dehghantanha, A., Choo, K.-K.R.: Machine learning aided android malware classification. Comput. Electr. Eng. **61**, 266–274 (2017)
28. Zhu, H.-J., You, Z.-H., Zhu, Z.-X., Shi, W.-L., Chen, X., Cheng, L.: Droiddet: effective and robust detection of android malware using static analysis along with rotation forest model. Neurocomputing **272**, 638–646 (2018)
29. Talha, K.A., Alper, D.I., Aydin, C.: Apk auditor: permission-based android malware detection system. Digit. Investig. **13**, 1–14 (2015)
30. Rovelli, P., Vigfússon, Ý.: Pmds: permission-based malware detection system. In: International Conference on Information Systems Security. Springer, pp. 338–357 (2014)
31. Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Bringas, P.G., Álvarez, G.: Puma: permission usage to detect malware in android. In: International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions, pp. 289–298. Springer (2013)
32. Zarni Aung, W.Z.: Permission-based android malware detection. Int. J. Sci. Technol. Res. **2**, 228–234 (2013)
33. Wang, W., Wang, X., Feng, D., Liu, J., Han, Z., Zhang, X.: Exploring permission-induced risk in android applications for malicious application detection. IEEE Trans. Inf. Forensics Secur. **9**, 1869–1882 (2014)
34. Ghasempour, A., Sani, N.F.M., Abari, O.J.: Permission extraction framework for android malware detection. Int. J. Adv. Comput. Sci. Appl. **11**(11) (2020)
35. Arora, A., Peddoju, S.K., Conti, M.: Permpair: android malware detection using permission pairs. IEEE Trans. Inf. Forensics Secur. **15**, 1968–1982 (2019)
36. Liu, X., Liu, J.: A two-layered permission-based android malware detection scheme. In: 2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering, pp. 142–148 (2014). https://doi.org/10.1109/MobileCloud.2014.22
37. Moonsamy, V., Rong, J., Liu, S.: Mining permission patterns for contrasting clean and malicious android applications. Futur. Gener. Comput. Syst. **36**, 122–132 (2014)
38. Sokolova, K., Perez, C., Lemercier, M.: Android application classification and anomaly detection with graph-based permission patterns. Decis. Support Syst. **93**, 62–76 (2017)
39. Wang, C., Xu, Q., Lin, X., Liu, S.: Research on data mining of permissions mode for android malware detection. Clust. Comput. **22**, 13337–13350 (2019)
40. Idrees, F., Rajarajan, M., Conti, M., Chen, T.M., Rahulamathavan, Y.: Pindroid: a novel android malware detection system using ensemble learning methods. Comput. Secur. **68**, 36–46 (2017)
41. Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Nieves, J., Bringas, P.G., Álvarez Marañón, G.: Mama: manifest analysis for malware detection in android. Cybern. Syst. **44**, 469–488 (2013)
42. Arslan, R.S., Ölmez, E., Er, O.: Afwdroid: deep feature extraction and weighting for android malware detection. Dicle üniversitesi MÜhendislik Fakültesi Mühendislik Dergisi **12**, 237–245 (2021)
43. Alazab, M., Alazab, M., Shalaginov, A., Mesleh, A., Awajan, A.: Intelligent mobile malware detection using permission requests and API calls. Futur. Gener. Comput. Syst. **107**, 509–521 (2020)

44. Tao, G., Zheng, Z., Guo, Z., Lyu, M.R.: Malpat: mining patterns of malicious and benign android apps via permission-related APIs. IEEE Trans. Reliab. **67**, 355–369 (2017)
45. Kim, T., Kang, B., Rho, M., Sezer, S., Im, E.G.: A multimodal deep learning method for android malware detection using various features. IEEE Trans. Inf. Forensics Secur. **14**, 773–788 (2018)
46. Hu, D., Ma, Z., Zhang, X., Li, P., Ye, D., Ling, B.: The concept drift problem in android malware detection and its solution. Secur. Commun. Netw. **2017** (2017)
47. Guerra-Manzanares, A., Nomm, S., Bahsi, H.: In-depth feature selection and ranking for automated detection of mobile malware. In: ICISSP, pp. 274–283 (2019)
48. Zhou, Y., Wang, Z., Zhou, W., Jiang, X.: Hey, you, get off of my market: detecting malicious apps in official and alternative android markets. In: NDSS, vol. 25, pp. 50–52 (2012)
49. Lindorfer, M., Neugschwandtner, M., Platzer, C.: Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis. In: IEEE 39th Annual Computer Software and Applications Conference, vol. 2, pp. 422–433. IEEE (2015)
50. Arora, A., Peddoju, S.K.: Ntpdroid: a hybrid android malware detector using network traffic and system permissions. In: 17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/12th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE), pp. 808–813. IEEE (2018)
51. Arora, A., Peddoju, S.K., Chouhan, V., Chaudhary, A.: Hybrid android malware detection by combining supervised and unsupervised learning. In: Proceedings of the 24th Annual International Conference on Mobile Computing and Networking, pp. 798–800 (2018)
52. Zhou, Y., Jiang, X.: Dissecting android malware: characterization and evolution. In: IEEE Symposium on Security and Privacy, pp. 95–109. IEEE (2012)
53. Guerra-Manzanares, A., Bahsi, H., Nõmm, S.: Kronodroid: time-based hybrid-featured dataset for effective android malware detection and characterization. Comput. Secur. **110**, 102399 (2021)
54. Mila: Contagio mobile. http://contagiominidump.blogspot.com/ (2018)
55. Arp, D., Quiring, E., Pendlebury, F., Warnecke, A., Pierazzi, F., Wressnegger, C., Cavallaro, L., Rieck, K.: Dos and don'ts of machine learning in computer security (2020). arXiv preprint arXiv:2010.09470
56. Pendlebury, F., Pierazzi, F., Jordaney, R., Kinder, J., Cavallaro, L.: {TESSERACT}: eliminating experimental bias in malware classification across space and time. In: 28th {USENIX} Security Symposium ({USENIX} Security 19), pp. 729–746 (2019)
57. Allix, K., Bissyandé, T.F., Klein, J., Le Traon, Y.: Are your training datasets yet relevant? In: International Symposium on Engineering Secure Software and Systems, pp. 51–67. Springer (2015)
58. Cen, L., Gates, C.S., Si, L., Li, N.: A probabilistic discriminative model for android malware detection with decompiled source code. IEEE Trans. Dependable Secure Comput. **12**, 400–412 (2015)
59. Xu, K., Li, Y., Deng, R., Chen, K., Xu, J.: Droidevolver: self-evolving android malware detection system. In: IEEE European Symposium on Security and Privacy (EuroS&P), pp. 47–62. IEEE (2019)
60. Lei, T., Qin, Z., Wang, Z., Li, Q., Ye, D.: Evedroid: event-aware android malware detection against model degrading for ioT devices. IEEE Internet Things J. **6**, 6668–6680 (2019)
61. Guerra-Manzanares, A., Luckner, M., Bahsi, H.: Android malware concept drift using system calls: detection, characterization and challenges. Expert Syst. Appl. **117200**, 117200 (2022)
62. Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., Zhang, G.: Learning under concept drift: a review. IEEE Trans. Knowl. Data Eng. **31**, 2346–2363 (2018)
63. Lu, N., Zhang, G., Lu, J.: Concept drift detection via competence models. Artif. Intell. **209**, 11–28 (2014)
64. Jordaney, R., Sharad, K., Dash, S.K., Wang, Z., Papini, D., Nouretdinov, I., Cavallaro, L.: TranscEnd: detecting concept drift in malware classification models. In: Proceedings of the 26th USENIX Security Symposium, pp. 625–642 (2017)
65. Hooker, G., Mentch, L.: Please stop permuting features: an explanation and alternatives (2019). arXiv preprint arXiv:1905.03151
66. Samara, B., Randles, R.H.: A test for correlation based on Kendall's tau. Commun. Stat. Theory Methods **17**, 3191–3205 (1988)
67. Aggarwal, C.C.: Data Mining: The Textbook. Springer, Berlin (2015)
68. Zyblewski, P., Sabourin, R., Woźniak, M.: Preprocessed dynamic classifier ensemble selection for highly imbalanced drifted data streams. Inf. Fusion **66**, 138–154 (2021)
69. Guerra-Manzanares, A., Nõmm, S., Bahsi, H.: Time-frame analysis of system calls behavior in machine learning-based mobile malware detection. In: 2019 International Conference on Cyber Security for Emerging Technologies (CSET), pp. 1–8 (2019). https://doi.org/10.1109/CSET.2019.8904908
70. Guerra-Manzanares, A., Bahsi, H., Nõmm, S.: Differences in android behavior between real device and emulator: a malware detection perspective. In: 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS), pp. 399–404 (2019). https://doi.org/10.1109/IOTSMS48152.2019.8939268
71. Maimon, O., Rokach, L. (eds.): Data Mining and Knowledge Discovery Handbook. A Complete Guide for Practitioners and Researchers. Springer, San Francisco (2005)
72. Breiman, L.: Random forests. Mach. Learn. **45**, 5–32 (2001)
73. Altmann, A., Toloşi, L., Sander, O., Lengauer, T.: Permutation importance: a corrected feature importance measure. Bioinformatics **26**, 1340–1347 (2010)
74. Biecek, P., Burzykowski, T.: Explanatory Model Analysis. Chapman and Hall, New York (2021)
75. Molnar, C.: Interpretable machine learning. https://christophm.github.io/interpretable-ml-book/ (2019)
76. Shapley, L.S.: 17. A Value for n-person Games. Princeton University Press, Princeton (2016)
77. Japkowicz, N., Shah, M.: Evaluating Learning Algorithms: A Classification Perspective. Cambridge University Press, New York (2011)
78. Breunig, M.M., Kriegel, H.-P., Ng, R.T., Sander, J.: Lof: identifying density-based local outliers. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, pp. 93–104 (2000)
79. Wu, L.: Android mobile ransomware: bigger, badder, better? https://www.trendmicro.com/en_us/research/17/h/android-mobile-ransomware-evolution.html (2017)
80. Seals, T.: Slocker android ransomware resurfaces in undetectable form. https://www.infosecurity-magazine.com/news/slocker-android-ransomware/ (2017)