Computers & Security 122 (2022) 102835

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

On the relativity of time: Implications and challenges of data drift on long-term effective android malware detection

Alejandro Guerra-Manzanares*, Hayretdin Bahsi

Department of Software Science, Tallinn University of Technology, Estonia

ARTICLE INFO

Article history:
Received 24 March 2022
Revised 29 June 2022
Accepted 12 July 2022
Available online 16 July 2022

Keywords:
Android malware
Machine learning
Timestamp
Malware detection
Concept drift
Malware evolution
Data drift

ABSTRACT

The vast body of research in the Android malware detection domain has demonstrated that machine learning can provide high performance for mobile malware detection. However, the learning models have been usually evaluated with data sets encompassing short time frames, generating doubts about the feasibility of these models in operational settings that deal with the ever-evolving malware threat landscape. Although a limited number of studies have developed concept drift resilient models for handling data drift, they have never considered the impact of different timestamps on the detection solutions. Timestamps are critical to locating the data samples within the historical timeline. Different timestamping approaches may locate samples differently, which, in turn, can significantly impact the performance of the model and, consequently, the adaptive capabilities of the system to concept drift. In this study, we conducted a comprehensive benchmarking that compares the detection performance of six distinct timestamping approaches for static and dynamic feature sets. Our experiments have demonstrated that timestamp selection is an important decision that has a significant impact on concept drift modeling and the long-term performance of the model regardless of the feature type used for model construction.

1. Introduction

Mobile devices have continuously penetrated every aspect of our lives, including personal and business-related. The vibrant ecosystem of mobile app development has accompanied the progress in the areas of mobile hardware and OS, leading to the very rapid uptake of technology. Considering the increasing involvement of mobile apps in the management of IoT devices, mobile security threats may cause economic, societal, or even physical damage. In this threat category, malicious applications pose a significant attack vector due to their evolving nature. Despite the countermeasures applied by device and OS vendors (Google, 2021; Samsung, 2021), the threat still remains. Therefore, it is imperative to timely detect and isolate the apps showing malicious behavior. This study focuses on Android malware, as Android is the dominant OS in the market (72% as of September 2021 (Statista, 2021)) and the target of most mobile malware (Islam, 2021).

Machine learning (ML) methods have provided promising detection results in Android malware detection (Sharma and Rattan, 2021). However, the general validation approach in these studies is limited to the utilization of data sets that encompass specific time frames without paying attention to the detection performance against the evolving threat landscape that includes new malware types and variants of existing ones.

Here, it is important to underline the general expectation of the cyber security domain about the ML methods in detecting malicious behavior. The current signature-based solutions supported by the cyber threat intelligence ecosystem constitute a working solution despite its various well-known shortcomings. ML methods extend this capability by detecting malware not seen before (Buczak and Guven, 2015). However, *temporal experimental bias* arises as a significant issue when the models are not tested with samples that belong to later times than the training samples (Pendlebury et al., 2019). The existence of such bias creates confusion about whether the model can detect new and evolved malware and whether the operational environments can sustain their discriminatory capability for a long time against data shifts in malware or benign data. Therefore, it is necessary to model the change in the threat landscape (i.e., concept drift) by considering longer time frames and determining the life cycle of the detection models (e.g., creation and update). We even argue that handling the concept drift should be an integral part of any ML-based malware detection solution to demonstrate its viability in operational settings.

Regardless of the utilized features or modeling approaches, timestamps are the central element for concept drift analysis and

* Corresponding author.
E-mail addresses: alejandro.guerra@taltech.ee (A. Guerra-Manzanares), hayretdin.bahsi@taltech.ee (H. Bahsi).

https://doi.org/10.1016/j.cose.2022.102835
0167-4048/

its proper handling as they provide the grounds of the *historical* context. They enable us to locate every app within the Android historical timeline according to their values. In this regard, the misplacement of a large number of samples may distort the data distribution and feature space representation. As a result, an inaccurate *timestamping* approach may prevent the model grasp the natural evolution of malware, which is vital for proper detection. However, despite the critical role of timestamps, there is no unambiguous approach to determining an app's temporal location with complete reliability and accuracy. A mobile app includes many files with temporal metadata that can be used as a timestamp for the app. External information sources such as *VirusTotal* can also provide the temporal context regarding the appearance of malware *in the wild*. It is hard to define the exact timing of the appearance of specific malware and malware families as they usually evolve from older variants or reuse code from other malware types.

Despite their importance, timestamping methods and their impact on learning models have not been thoroughly investigated. In the literature, only Guerra-Manzanares et al. (2022b) used two timestamps to locate samples in the system calls feature space analyzing cross-platform detection issues. In contrast, we conducted a benchmarking study that investigates the impact of six distinct temporal approaches on concept drift-based models induced by using a dynamic feature set, system calls, and two static feature sets, API calls and permissions. As a data set, we utilized *KronoDroid*, which covers a wide period (i.e., 2008–2020), in addition to providing various timestamp alternatives (Guerra-Manzanares et al., 2021). First, we performed a comprehensive statistical analysis of malicious and benign apps to explore the *availability* and *validity* of the temporal data obtained by each timestamping approach. In this part, we also developed an approximation method to compare the accuracy of the obtained data. Then, we formulated a concept drift-handling method that uses a *pool* composed of classifiers to compare the detection rates of each timestamping approach for each feature set.

Our results indicate that the selection of the timestamping method has a significant impact on the detection accuracy of the learning models that encompass long time frames regardless of the feature set nature (i.e., dynamic or static). Thus, it is imperative to consider the collection of relevant temporal values besides the comprehensiveness of the data set. Note that our work does not aim to optimize the performance of the concept drift-handling method. Instead, it uses a working solution to compare the performance of temporal data for each feature set.

Our study provides a unique contribution to the literature as a detailed evaluation regarding the impact of timestamping alternatives on the performance of concept drift models has not been addressed for mobile malware detection.

The outline of the paper is as follows: Section 2 defines concept drift and its consequences, while Section 3 surveys the related literature. Methods and analytical approaches used in the study are presented in Section 4. Section 5 provides the results of our experiments. The key findings, contributions and limitations of our study are given in Section 6. Section 7 concludes the paper.

2. Concept drift

Most learning models are *static*, meaning that they assume stationary data distributions, which are consistent over time. Thus, the training and testing data are assumed to be *similar*. However, non-stationary data distributions might be found in many problem domains, such as in Android malware detection, where data evolves over time (e.g., the emergence of new malware families), adding additional complexity to the data modeling process that, if not addressed, can severely impact the generalization of the detection model to future (evolved) data leading to model obsolescence, a phenomenon called *concept drift*.

Concept drift can be defined as the phenomenon in which the statistical properties of data change over time in an unpredictable manner (Lu et al., 2018). Formally, concept drift can be defined as follows.

Given a bounded period of time $[t_0, t_1]$ and a set of examples from that period $S_{t_0,t_1} = \{s_{t_0}, \ldots, s_{t_1}\}$, where $s_i = (x_i, y_i)$ relates to a single observation, $x_i = (x_i^1, x_i^2, \ldots, x_i^n) \in \mathbf{X}$ defines the feature vector, $y_i \in \mathbf{Y}$ corresponds to the target label, and S_{t_0,t_1} follows a specific distribution $F_{t_0,t_1}(\mathbf{X}, \mathbf{Y})$. The phenomenon of concept drift is observed at t_2 if $F_{t_0,t_1}(\mathbf{X}, \mathbf{Y}) \neq F_{t_2,\infty}(\mathbf{X}, \mathbf{Y})$, and may be denoted as $\exists t : P_t(\mathbf{X}, \mathbf{Y}) \neq P_{t+1}(\mathbf{X}, \mathbf{Y})$ (Lu et al., 2018). Following this definition, concept drift at time period t_i can be equated as the change in the joint probability of $\mathbf{X}$ and $\mathbf{Y}$ at time t_i, expressed as $P_{t_i}(\mathbf{X}, \mathbf{Y})$. As $P_{t_i}(\mathbf{X}, \mathbf{Y}) = P_{t_i}(\mathbf{X}) \times P_{t_i}(\mathbf{Y} \mid \mathbf{X})$, concept drift can arise from three primary sources (Lu et al., 2018):

1) $P_t(\mathbf{X}) \neq P_{t+1}(\mathbf{X})$ and $P_t(\mathbf{Y} \mid \mathbf{X}) = P_{t+1}(\mathbf{Y} \mid \mathbf{X})$. This shows a change in the input data distribution, $P_t(\mathbf{X})$, that has no impact on the posterior probability of the learning model, $P_t(\mathbf{Y} \mid \mathbf{X})$, thus not affecting its decision boundary. This phenomenon is named *virtual drift*.
2) $P_t(\mathbf{X}) = P_{t+1}(\mathbf{X})$ and $P_t(\mathbf{Y} \mid \mathbf{X}) \neq P_{t+1}(\mathbf{Y} \mid \mathbf{X})$. In such conditions, the input data distribution does not change, but the changes in the posterior probability modify the decision boundary, leading to a decrease in the learning accuracy, which reflects *real concept drift*.
3) $P_t(\mathbf{X}) \neq P_{t+1}(\mathbf{X})$ and $P_t(\mathbf{Y} \mid \mathbf{X}) \neq P_{t+1}(\mathbf{Y} \mid \mathbf{X})$. In such a scenario, the shift in the feature data distribution coincides with a change in the decision boundary, defining a *real concept drift*.

As can be noted from the previous definitions, only the *real* concept drift changes the decision boundary of the model, which affects the generalization capabilities of the model and leads to model obsolescence. More precisely, *real concept drift* relates to changes in the model's decision boundary, defined by $P(\mathbf{Y} \mid \mathbf{X})$, that might coexist with *covariate shift* (Gama et al., 2014). *Virtual drift*, also named *feature space drift* or *covariate shift*, reflects a change in the underlying data distribution that does not affect the model decision boundary, that is, $P(\mathbf{Y} \mid \mathbf{X})$. In this regard, from a predictive perspective, only the changes that affect the decision boundary, that is, the *class* prediction made by the model, require the implementation of adaptive measures (Gama et al., 2014).

3. Related work

3.1. Concept drift in android malware detection

The vast majority of literature regarding Android malware detection neglects the existence of concept drift. The models and proposed solutions are generally trained and validated on static *snapshots* of Android historical data, usually with the same *well-known* data sets (Abderrahmane et al., 2019; Afonso et al., 2015; Ahsan-Ul-Haque et al., 2018; Alzaylaee et al., 2017; 2020; Amin et al., 2016; Bhatia and Kaushal, 2017; Burguera et al., 2011; Canfora et al., 2015; Casolare et al., 2021; Da et al., 2016; Dimjašević et al., 2016; Feng et al., 2018; Ferrante et al., 2016; Frenklach et al., 2021; Guerra-Manzanares et al., 2019a; 2019b; 2019c; Hei et al., 2021; Hou et al., 2016; Isohara et al., 2011; Jaiswal et al., 2018; Jang et al., 2014; Kapratwar et al., 2017; Lin et al., 2013; Lindorfer et al., 2015; Liu et al., 2021; Mahindru and Sangal, 2021; Malik and Khatter, 2016; McLaughlin et al., 2017; Naval et al., 2015; Rathore et al., 2021; Saif et al., 2018; Saracino et al., 2018; Sasidharan and Thomas, 2021; Sihag et al., 2021; Singh and Hofmann, 2017; Surendran et al., 2020; Tchakounté and Dayang, 2013; Tong and Yan, 2017; Vidal et al., 2017; Vinod et al., 2019;

Wahanggara and Prayudi, 2015; Wang and Li, 2021; Xiao et al., 2015; 2016; 2019; Yu et al., 2013; Yuan et al., 2014). In this regard, *MalGenome* (Zhou and Jiang, 2012) and *Drebin* (Arp et al., 2014) are the most used data sets for Android malware research. Despite their small size and *outdated* data (i.e., they were collected between 2009–2012), they are still widely used as the primary source of malware samples in research studies (Rathore et al., 2021; Sasidharan and Thomas, 2021).

Even though some recent studies (Cai et al., 2021a; 2021b; Gao et al., 2021) complement their data with more recent and larger data sets, such as the *Android Malware Dataset* (AMD) (Wei et al., 2017), to mitigate data-related issues (e.g., Drebin duplication issue (Irolla and Dey, 2018)), they still rely on *partial*, relatively *old* (e.g., AMD's most recent sample dates back to 2016 and includes just 71 malware families), and *short* snapshots of data within the whole Android historical timeline (i.e., 2008–2022). Furthermore, when the proposed solutions for Android malware detection are generated, it is common to randomize the data set order, thus neglecting its *natural* ordering, and split it into two disjoint sets (i.e., train/test sets) or several train/test folds (i.e., cross-validation). This common procedure, which can be applied to any ML task where the data does not follow any natural order or evolution issues (e.g., image classification), can significantly harm the performance of solutions built for *naturally* ordered sequential data, such as natural language processing tasks or data streams. The randomization of train/test splits neglects apps' location in the historical timeline, undermining *historical coherence* and yielding *significantly biased* and *historically incoherent* results (Allix et al., 2015; Pendlebury et al., 2019). As a result, the training data may contain samples belonging to *future* time frames, and the testing data may include *past* samples contemporaneous to the training data. Thus the model is trained with an *impossible configuration*, with data that is not typically available in practice and providing biased performance, a phenomenon called *data snooping* (Arp et al., 2020). Consequently, these practices pose serious doubts about the *generalization* capabilities and effectiveness of these solutions for detecting evolved and recent malware.

Only a reduced number of the referred studies considered the usage of distinct snapshots of historical data for the train/test split. However, they included significant time gaps between them (Guerra-Manzanares et al., 2019a; 2019b; 2019c). Therefore, *concept drift* and its degenerative impact on the performance of the ML-based models were neglected.

As a result, the *time* variable and malware evolution have been neglected by the vast majority of Android malware research. Only a few studies dealing with Android malware detection have considered the concept drift issue and proposed ML solutions that *adapt* to changes in the data and aim to minimize its detrimental impact over time. In this regard, general frameworks have been proposed to detect emerging *drift* in Android data (Barbero et al., 2020; Jordaney et al., 2017; Pendlebury et al., 2019), although the vast majority of proposed solutions are based on API calls (Cai, 2020; Lei et al., 2019; Narayanan et al., 2016; Onwuzurike et al., 2019; Xu et al., 2019; Zhang et al., 2020).

3.2. Timestamps: when time matters

The essential constructs underlying effective concept drift handling are *timestamps*. Timestamps enable the temporal allocation of apps, aiming to provide a reliable temporal context. However, due to the nature of the malware generation and discovery process, the reliability of timestamps is questionable.

The studies that tackled concept drift-related issues in Android malware detection have not discussed the issue of timestamp selection. Although some studies did not provide any information about it (Onwuzurike et al., 2019), most studies used a similar approach, the *compilation date*. The compilation date, referred to using different names in the literature, is an *internal* timestamp that reports the creation or compilation time of the app archive (i.e., *apk*). Despite being pointed out as the best timestamp (Pendlebury et al., 2019) and used in related research (Barbero et al., 2020; Cai et al., 2020; Pendlebury et al., 2019; Xu et al., 2019), it has become an unusable approach as most of the apps released nowadays have it set in *1980* (du Luxembourg, 2021). Another *internal* timestamp, proposed more recently, is the last modification timestamp, which refers to the most recent modification *datetime* found in any of the inner files of the *apk* archive (Guerra-Manzanares et al., 2021). This feature was introduced in Guerra-Manzanares et al. (2021), which discussed the feasibility of four distinct timestamp approaches for Android malware detection.

Even though the internal timestamps, collected from apps' inner files, could be deemed *accurate*, they are prone to manipulation, which may cause invalid timestamps or temporal misplacement. A more robust temporal context can be obtained using *external* timestamps. An external timestamp is not retrieved from the app files but provided by an external entity. In this regard, Virustotal's *first seen*, also named *appearance* or *submission* time in the studies, reports the *datetime* at which a sample was first received by the VirusTotal scanning service. It has been used in research studies (Cai et al., 2019; Lei et al., 2019; Zhang et al., 2020) as it is based on third-party, reliable services, making the temporal assignment more robust to alterations. However, due to its proactive nature, this timestamp is prone to significant delay and temporal displacement (i.e., a user must submit the file to generate the timestamp).

Consequently, the timestamp attribute and its selection emerge as critical elements to handle concept drift effectively for long-term Android malware detection. Despite its criticality, it has been neglected by the concept drift-related studies in the related literature. In this research, we address this significant research gap by thoroughly analyzing, comparing, and evaluating different timestamps for effective concept drift handling.

4. Methodology

4.1. Data set

The data set used in this study is *KronoDroid* (Guerra-Manzanares et al., 2021). This data set provides timestamped and labeled Android app samples covering the whole 2008–2020 period. Each data sample is described using dynamic and static features. More precisely, it is composed of 489 features, of which 289 are dynamic features and 200 static features. For this research, system calls and permissions features were used along with the provided timestamps (i.e., four timestamps), class labels, and other relevant metadata. As Kronodroid is composed of two data sets (i.e., according to the source of the dynamic features), in this study, the *real device dataset* was preferred due to its larger size for the same historical time frame (i.e., 78,137 samples). Furthermore, the selection of real device-collected data prevents that any behavioral differences could be influenced by malware *anti-sandbox* techniques. Therefore, the data set used in this research is composed of 41,382 malware samples belonging to 240 malware families and 36,755 benign apps, encompassing from 2008 to 2020.

To explore the impact of the time variable and the concept drift issue extensively, additional features were collected for this study. More specifically, for each data sample, Android API calls were statically collected using *Androguard* (Desnos et al., 2018) and two additional timestamps from the inner *apk* files (i.e., *dex date* and *manifest date*). The usage of all these data features enables us to observe and analyze the *time* impact on distinct feature spaces for the same set of data within the same problem do-

Table 1
Feature spaces.

Feature space	Dimensions	Type
System calls	288	Numeric
Permissions	166	Binary
API calls	53,523	Binary

main (i.e., system calls, permissions, and API calls for malware detection) from six different temporal perspectives (i.e., timestamps). The sample descriptors (i.e., features) are representative of widely used dynamic and static approaches for Android malware detection (Feizollah et al., 2015).

The feature sets used in this research, their dimensions, and data type are summarized in Table 1. The temporal perspectives used to date the apps within the Android historical timeline and their description are outlined in Table 2.

As can be observed in Table 1, three feature spaces are analyzed in this research for the same data set, providing complementary perspectives for the same phenomenon (i.e., concept drift). Furthermore, as the feature spaces are defined on varying dimensions and data types, they enable us to explore and analyze the issue from a wide perspective. In addition, the temporal dimensions segment and transform the feature spaces distinctively, thus providing extensive exploration of the studied phenomenon. In this study, six distinct timestamps are used and thoroughly compared. KronoDroid data set provides four possible timestamps per sample: *last modification, earliest modification, first seen,* and *first seen in the wild*. The nature of their temporal attribution is described in Table 2. Besides these timestamps, the *dex date* and *manifest date* timestamps were extracted in this research for all the *apk*s that compose the original data set. These timestamps were collected by inspection of two important app inner files: *classes.dex* and *AndroidManifest.xml*. The *dex date* timestamp, not included in the *Kronodroid* data set due to the inaccurate value reported for recent samples (Guerra-Manzanares et al., 2021; du Luxembourg, 2021), has been widely used in previous related research to locate apps within the Android timeline, thus being relevant for the comparison. The *dex date* timestamp is also referred to in the literature as *compilation* or *creation* time, referring to the *classes.dex* timestamp, which is essentially the compiled source code. The *manifest date* reports the timestamp of the *AndroidManifest.xml* or *app manifest* file. The addition of the *manifest date* as a temporal approach is based on the critical relevancy of this file for any Android app. In this regard, the *manifest* is the only compulsory file for any Android app as it provides the essential information about the app to the Android build tools, the operating system, and Google Play (Android, 2021a). Therefore, the relevancy of this critical file and its *omnipresent* characteristic are leveraged as the grounds for its inclusion in a complete analysis.

4.2. Workflow

4.2.1. Data collection

The initial stage of this research involved the collection of additional features to describe every sample in the selected data set.

API *external* methods were extracted from every apk file using *Androguard* (Desnos et al., 2018). As not all *external* methods are Android API calls, the results were filtered to select only the methods relative to the existing Android API families (Android, 2021b). These *Android Platform API* families include *android, dalvik, java, javax, junit, apache, json, dom*, and *xml*. A similar approach was used in API call-related studies (Onwuzurike et al., 2019; Xu et al., 2019). Each class method defined by the apps in the data set was included as a feature. Therefore, all samples within the data set were described using a binary vector that indicates the presence/absence (i.e., 1/0) of each API call for each data sample.

The additional timestamps used to describe every data sample (i.e., *dex date* and *manifest date*) were retrieved from the inner files of the *apk* archive by inspecting the metadata extracted from the *classes.dex* and *AndroidManifest.xml* files. A *Python* script was used for that purpose.

The collected data were processed and structured in *CSV* format, where each row referred to a data sample and each column to a feature. As a result, each sample was described using a vector concatenating the data for the three independent feature spaces (i.e., syscalls, permissions, and API calls), the six timestamps, the class label, malware family, and the *sha-256* hash value that uniquely identifies each data sample.

The feature spaces explored in this research are representative of the most common attributes used for Android malware detection purposes. *System calls* are the most widely used dynamic features, whereas *permissions* and *API calls* are the features used in the vast majority of *static* approaches for malware detection (Feizollah et al., 2015).

4.2.2. Timestamp analysis

The main focus of this research is the temporal dimension of the data and its impact on concept drift representation, analysis, and handling. Therefore, in this second stage, a thorough comparative analysis of the timestamps was performed.

The usage of a timestamp to locate applications across the Android historical timeline is subject to *availability* and *reliability* issues. The former refers to the *accessibility* of the timestamp for its collection, while the latter regards the temporal precision of the timestamp concerning the ground-truth location of the app within the historical timeline. As the ground-truth temporal location is hardly achievable in the vast majority of cases (i.e., it is not possible to know with absolute certainty when the sample was released), the timestamp approaches main aim is to provide a *good approximation* to the given phenomenon in the absence of ground-truth data. In our approach, a good approximation would minimize the amount of error for the majority of the samples and enable the handling of concept *drifts* effectively. In this regard, due to the absence of a ground-truth temporal reference, our assumption is that an effective concept drift-handling solution may provide relatively more stable and smoother performance over time using an accurate timestamp than with an inaccurate timestamp, as data should usually evolve through shifting gradually over time, introducing new elements and discarding others in a relatively smooth transition. Sudden data drifts may happen over time, but their prevalence should not be significant (e.g., completely new malware outbreaks). Otherwise, concept drift could hardly be modeled, and keeping high performance over time would be a utopia.

At this stage, the following steps were performed to analyze and compare the timestamps using their intrinsic properties, whereas the next stage, described in Section 4.2.3, explores their suitability for concept drift handling.

The sequential steps of the comparative analysis are described as follows:

1) *Prevalence*: the prevalence of timestamps is a term related to *availability* that informs about the accessibility of the timestamp, that is, if the timestamp can be successfully collected or retrieved from the samples. For each timestamp, the number of set timestamps (i.e., properly defined) and not set timestamps (i.e., *missing* or undefined) was analyzed.
2) *Validity*: the validity of a timestamp is an indicator of whether the timestamp is comprised within the Android historical time frame (i.e., from the 22nd of October 2008 (Google, 2008) to the present day). It is not an indicator of accuracy but discards

Table 2
Timestamps.

Timestamp name	Description
Last modification	The most recent timestamp retrieved from any file inside the *apk* archive
Earliest modification	The *oldest* timestamp retrieved from any file inside the *apk* archive
First seen	Date and time of the first submission of the sample to *VirusTotal*
First seen in the wild	Date and time of the first time the sample was seen anywhere on internet
Dex date	Timestamp retrieved from the *classes.dex* file, located inside the *apk* archive (i.e., compilation time)
Manifest date	Timestamp retrieved from the *AndroidManifest.xml* file, located inside the *apk* archive

all the timestamps not comprised within the *valid* Android timeline range (i.e., before the 22nd of October 2008 or in the future).

3) *Suitability*: the suitability of a timestamp combines the previous concepts positively. Thus a *suitable* timestamp is *available* and *valid*. Consequently, an *unsuitable* timestamp is *available* but *invalid*.
4) *Distribution and statistical analysis*: data distributions for each timestamp and each class were analyzed and compared using statistical measures. Histograms were used to visualize the data distributions and as input to statistical tests and techniques for similarity assessment. Two statistical methods for measuring the similarity between data distributions were used:
   - *Jensen–Shannon distance*: a distance metric that is the square root of the *Jensen–Shannon divergence*. The Jensen–Shannon divergence is computed using the *Kullback Leibler (KL) divergence*, but it has more desirable properties than the KL divergence, such as being always finite, symmetrical, and smooth, thus preferred when some probability values are small or zero. The Jensen–Shannon divergence is computed as:

     $$JSD(P||Q) = \frac{D_{KL}(P||M)}{2} + \frac{D_{KL}(Q||M)}{2},$$

     where P and Q refer to two probability distributions, M is the point-wise mean of P and Q calculated as $\frac{1}{2}(P+Q)$ and D_{KL} refers to KL divergence values calculated for each pair of distributions. The KL divergence, also named *relative entropy*, between two discrete distributions is calculated as:

     $$D_{KL}(P||Q) = \sum_{x \in X} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

     The KL divergence quantifies the difference between two probability distributions, which is leveraged by the Jensen–Shannon divergence to provide a more comprehensive, smooth, and well-defined distance metric (i.e., the square root of JSD) to measure the similarity between two probability distributions. The JSD distance for two probability distributions is bounded between [0, 1] when the logarithm to the base 2 is used for computations. The general interpretation is that the higher the value (i.e., closer to one), the greater the difference between the distributions.
   - *Kolmogorov–Smirnov two-sample test*: a non-parametric statistical hypothesis test to assess the equality of one-dimensional probability distributions. It enables us to assess the probability that two collections of samples (i.e., $F(x)$ and $G(x)$) could have been drawn from the same probability distribution, that is, if they are statistically similar. The null hypothesis H_0 for the test is that the two distributions are identical (i.e., $F(x) = G(x), \forall x$), whereas the alternative hypothesis H_1 is that they are not identical. The Kolmogorov-Smirnov (KS) test answers this hypothesis test by analyzing the maximum difference between the two experimental cumulative frequency distribution functions. The KS statistic is calculated as:

     $$D_{m,n} = \sup_x \mid F_n(x) - G_m(x) \mid,$$

     where $F_n(x)$ and $G_m(x)$ refer to the empirical distribution functions of the two data samples, of sizes m and n, respectively, and *sup* is the *supremum* function. For large samples, the null hypothesis is rejected at significance level α if

     $$D_{m,n} > c(\alpha)\sqrt{\frac{m+n}{mn}},$$

     where m and n are the sizes of the distributions and the value of $c(\alpha)$ is a parameter calculated as $c(\alpha) = \sqrt{-ln(\frac{\alpha}{2})\frac{1}{2}}$.

   Therefore, different combinations of data distributions based on the timestamps were analyzed regarding their similarity using both measures. As they evaluate *similarity* using different approaches, the usage of both techniques provides a better overall perspective of the differences between the analyzed sets.
5) *Accuracy*: an approximation of the accuracy of the timestamps was explored to assess their reliability. The evaluation of timestamp accuracy (i.e., the precise location of the sample within the Android historical timeline) is a significant challenge due to the absence of an exact ground-truth timestamp (i.e., it is not possible to surely know when the malware was first released). Thus only approximations to the ground-truth timestamp can be targeted. For this purpose, we used open-source intelligence feeds such as new malware family discovery news by antivirus vendors and media sources to establish an approximate *discovery time* of a specific malware family. After that, a time frame around the date was established (i.e., ± 6 months), and statistics were retrieved from each timestamp data distribution for each family. The rationale behind this analysis is that, if the timestamp is accurate, it should place samples around that time frame (i.e., *discovery time* ±6 months) and after it, implying that the malware family might be located accurately and also its evolution (i.e., samples dated after this range). If a significant amount of samples is placed outside of this time frame, the timestamp might be deemed relatively inaccurate. This approach is naive and has notable limitations as it relies on the precision of the malware family labels and also the relative accuracy of the news feed. Nevertheless, it provides a good notion of the accuracy of the timestamps, especially when the results are compared.

4.2.3. Concept drift handling

The concept drift problem is usually identified with large data streams, as the meaningful properties and descriptors of data are prone to change over time (Aggarwal, 2015; Margara and Rabl, 2018). Android malware detection, which can be considered a constant flow of data, faces concept drift issues. The impact of concept drift on detection classifiers is a substantial decrease in performance over time until the detection model becomes obsolete. Thus, concept drift issues should be taken into account to design high-performance and long-lasting detection systems that are resilient to data changes over time.

In this study, an algorithm designed to handle concept drift in data streams was used to explore the impact of the timestamps to model and handle emerging data drift effectively. In Zyblewski et al. (2021), the incoming data stream is split into *chunks* and processed sequentially. The method uses a pool of classifiers trained on past data chunks to predict new data samples. During the process, the best ensemble of classifiers is dynamically selected to perform accurate predictions. Furthermore, the pool is updated after each new incoming data chunk by introducing classifiers trained on the new data and removing low-performance, *aging* models. This last step makes the system resilient to concept drift, keeping high performance over time by updating the pool of classifiers with new, evolved data.

The original algorithm by Zyblewski et al. (2021) was slightly modified for our scenario as described in the proposed methodology by Guerra-Manzanares et al. (2022a). More specifically, the initial data chunk was split to generate a full pool from the beginning. This fact avoids the need of waiting to n chunks to gradually fill the classifier pool (i.e., the n hyper-parameter refers to the fixed pool size), as the original solution proposes. This enhanced classification methodology to deal with Android data concept drift, as detailed in Guerra-Manzanares et al. (2022a), was used to analyze the impact of different timestamps on concept drift handling in distinct feature spaces within the same problem domain.

The performance of the classification method was evaluated using the $F1$ score metric. The $F1$ performance metric provides a notion of the accuracy of the classifier in detecting malware instances. It is a comprehensive metric that is calculated as the harmonic mean of *precision* and *recall* performance metrics as defined by the following equation:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{1}$$

In this regard, the *precision* of a classification model informs about the fraction of actual malware data points that the model correctly classified as *malware* among all malware predictions and is calculated as:

$$Precision = \frac{TP}{TP + FP}, \tag{2}$$

where TP refers to *True Positive*, that is, the number of actual malware that was predicted as malware by the model, and FP relates to *False Positive* or the number of benign instances incorrectly predicted as malware by the model.

The *recall* metric reports the fraction of samples classified as malware (i.e., positive) among the total number of actual malware samples included in the testing set. It is calculated as:

$$Recall = \frac{TP}{TP + FN}, \tag{3}$$

where TP relates to *True Positive* and FN refers to *False Negative*, that is, the number of actual malware instances incorrectly predicted as benign by the classification model.

In the case of imbalanced data scenarios, such as the one explored in this study, the F1 score reports better the positive class detection performance than overall accuracy, thus being a preferred performance metric for our analysis.

5. Results

5.1. Data collection

The original data set was composed of 41,382 malware samples and 36,755 benign apps. Each sample was described by 288 system call features (i.e., counts), 166 permission-related features (i.e., binary indicators), 4 timestamps (i.e., *earliest modification*, *last modification*, *first seen,* and *first seen in the wild*), the hash value, and the class and family labels. To extend the set of feature spaces evaluated, API calls were collected. The API calls features extraction was successful for 99.82% of the original data set, with the final data set composition described in Table 3.

Table 3
Final data set composition.

Class	Original data set size	Error	Final data set size
Benign	36,755	38	36,717
Malware	41,382	104	41,278
Total	78,137	142	77,995

As can be observed in Table 3, the API call collection process was not successful for 104 malware apps and 38 benign apps from the original data set due to corrupted file headers in particular *apk* archives (i.e., *zip* files), which prevented the extraction tool from parsing the compressed bundle and retrieving the data (i.e., failed *magic number* verification provoked a critical *Bad Zip File* error). As the comparative analysis required the same data set to be used in all the experimental setups, the final data set used was composed of 36,717 benign samples and 41,278 malware samples. There were no issues retrieving the *dex date* and *manifest date* timestamps for any of the samples.

Therefore, the final data set samples are described by 3 distinct multi-dimensional feature spaces (i.e., *system calls, permissions,* and *API calls*) and 6 timestamps (i.e., *last modification, earliest modification, first seen, first seen in the wild, dex date,* and *manifest date*).

5.2. Timestamps analysis and relations

The following sections describe and provide a comparative analysis of the timestamps from different perspectives regarding *availability* and *reliability* measures. This is the essential part of this study that aims to compare and evaluate distinct timestamp approaches and their suitability to build concept drift resilient machine learning-based Android malware detection models.

It is worth noticing that of the set of six timestamps analyzed in this research, two correspond to *external* timestamps, not extracted from the apk archive metadata, which were set in this case by VirusTotal scanning reports. These timestamps are the *first seen* and *first seen in the wild*. An external timestamp is less prone to be manipulated by perpetrators as it is not in the immediate scope of the attacker. However, they can be prone to delays as they depend on users' proactive behavior (i.e., user submission to VirusTotal's service) and processing errors. Besides, the *first seen in the wild* timestamp, defined as the first time the app was seen anywhere on the internet, might not be set for benign applications.

The remaining four timestamps are *internal* timestamps, collected from the inner files of the app bundle. Therefore, they can be manipulated or removed by a motivated attacker.

5.2.1. Prevalence of timestamp data

The prevalence of timestamps data provides a notion of data availability, which is critical to building effective learning systems. If the timestamp cannot be retrieved, the sample cannot be located in the historical context and is, consequently, *unusable*. Based on this, Fig. 1 conveys graphically the *prevalence* or *availability* property for every timestamp and the whole data set. The horizontal axis provides the timestamps, referenced in abbreviated form. *EM* refers to the earliest modification timestamp, *LM* to the last modification timestamp, *FS* is used for *first seen, FSW* for *first seen in the wild, DD* for *dex date,* and *MD* for *manifest date*. For every timestamp, two vertical bars are defined, which inform about the relative frequency or *percentage* of data samples that had the timestamp *available*, meaning that it was defined or properly set. The colored areas refer to class-wise proportions (i.e., red for malware,

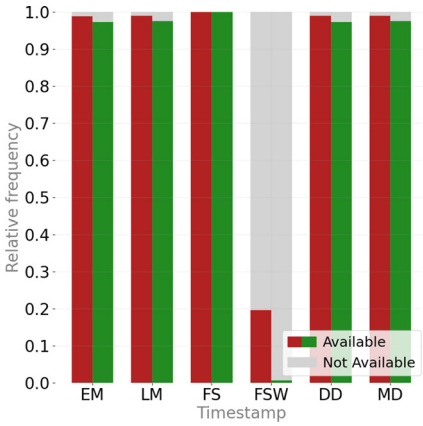


Fig. 1. *Availability* of timestamps.

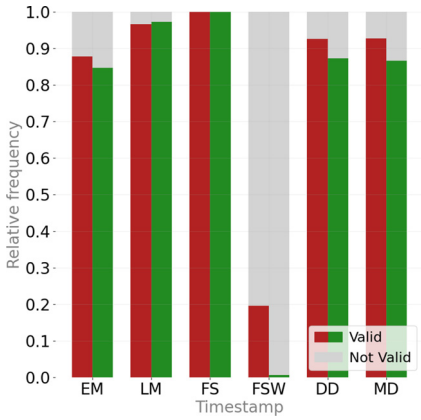


Fig. 2. *Validity* of timestamps.

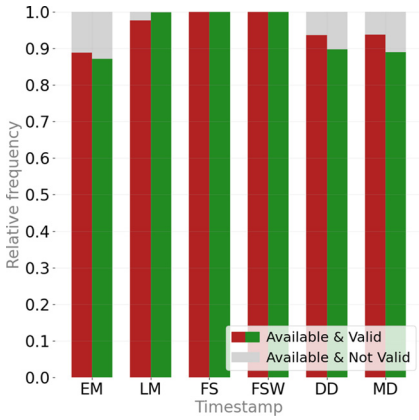


Fig. 3. *Suitability* of timestamps.

green for benign apps), while the grey areas indicate the proportion of data samples that did not have the specific timestamp available. Two main causes of *nonavailability* were found. In the case of *FSW,* the data might not be found in the detection report, so there was no timestamp set for the sample. For the other timestamps, a *nonavailable* timestamp was found as null-valued timestamp metadata, which was incorrectly parsed and retrieved by the software as *1980-00-00*.

As can be observed in Fig. 1, most of the timestamps are available for all data samples, as most of the bars reach beyond 97% prevalence. Furthermore, the *first seen* was defined for all data samples. This was expected as the timestamp is automatically set upon submission of the sample for the first time. The other timestamps, collected using different means and from the internal files, are mostly available, especially for malware instances. More precisely, a larger proportion of null-valued timestamps was found on legitimate apps. A fact that may seem counter-intuitive, but that was also pointed out by du Luxembourg (2021). An exception to the high availability of the timestamps is the *first seen in the wild* timestamp. As most of the reports retrieved did not provide data for this feature, it is missing for most of the applications, especially for benign apps (i.e., found only for 0.6% of them). This is logical as the scanning service aims to detect malware threats effectively (i.e., positive detection), so the *first seen in the wild* location for benign instances is actually irrelevant.

5.2.2. Validity of timestamps

The timestamp for any Android app sample should be located within the Android historical timeline, which encompasses from the 22nd of October 2008 (i.e., Android Google Market public release) to the present day. Any timestamp located within this time frame is deemed *valid*. Timestamps located in the future (e.g., 2107) or in the past (e.g., 1997), which were found in the data, are *impossible* configurations, suggesting tampering and consequently labeled as *not valid*. Fig. 2 reports the *validity* property for every timestamp and the whole data set. Similar to Fig. 1, the horizontal axis provides the timestamps, referenced in abbreviated form. The vertical bars report the proportion of *valid* timestamps for each class using green and red colors and the *not valid* as shaded areas.

Fig. 2 reports similar values to the ones in Fig. 1 for the *FS* and *FSW* timestamps. However, for the *EM, LM, DD,* and *MD* timestamps, the bars reach lower figures, especially for the *EM* timestamp. This indicates that this timestamp is the one that contains more non-valid values, followed by *DD* and *MD* timestamps. In all cases, except for *EM*, malware samples reach higher values than benign samples, which again might seem counter-intuitive. However, this fact may only reflect a general disregard regarding timestamps by benign apps' developers but does not provide any hint about the accuracy of the timestamp.

5.2.3. Suitability of timestamps

For the purpose of this research, the concept of *suitability* provides a notion about the most *usable* timestamps, that is, they are *available* and *valid*. Fig. 3 reports the *suitability* proportion per timestamp and class. In this case, the colored areas refer to class-wise timestamps that are both available and valid. The grey areas report the proportion of samples that have available but invalid timestamps.

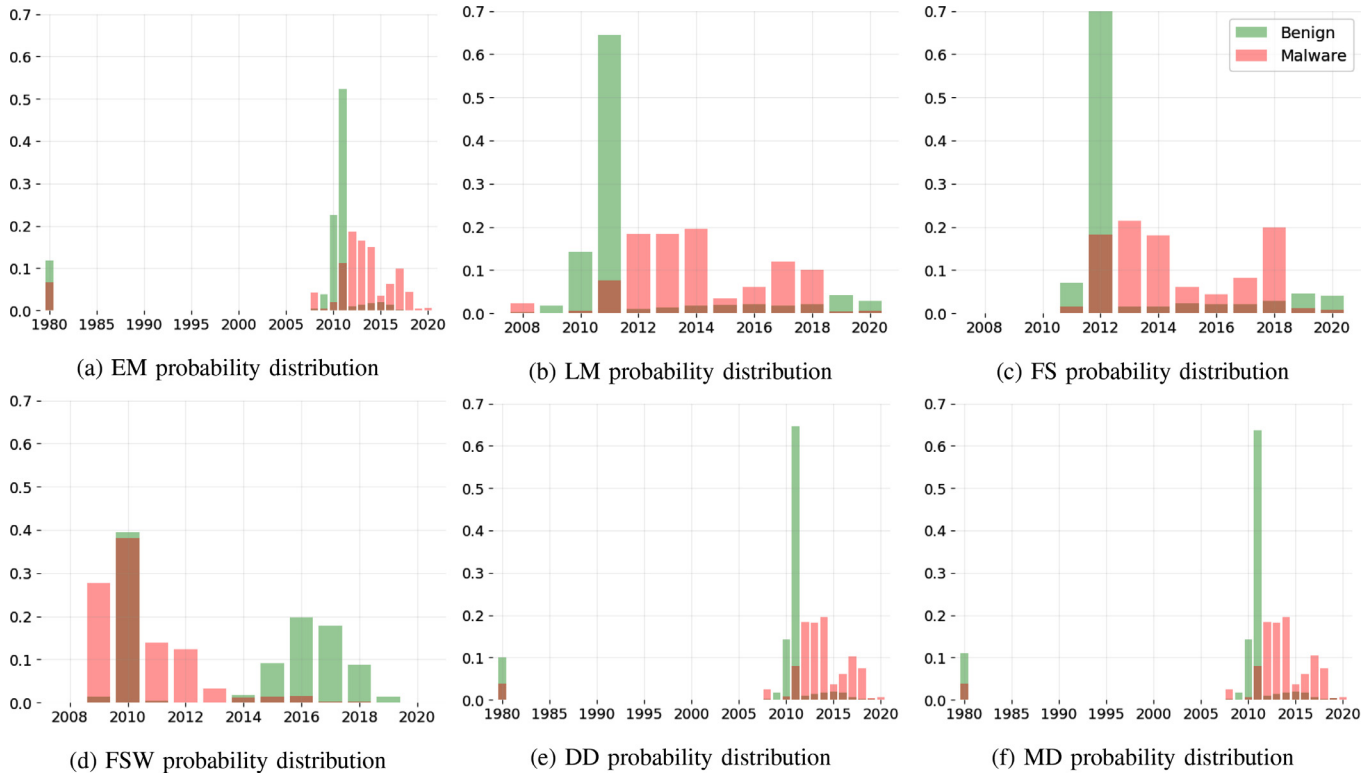
Fig. 3 conveys that *FS, FSW,* and *LM* are the most *suitable* timestamps, with a large proportion (i.e., 100% for *FS* and *FSW*) of available data that lie within the valid time frame. However, despite the high *suitability* of *FSW*, its low prevalence makes it a worse option than *FS* and *LM* if data quantity is a requirement. The *EM, DD,* and *MD* timestamps show values ranging from 87% to 93%, thus deemed as the *least suitable* options.

Figs. 1–3 enable us to rank the timestamps based on their combined properties. As a result, *FS* and *LM* significantly outperformed the other timestamps based on the analyzed criteria.

5.2.4. Variability: assessing the similarity of timestamps

The probability distribution of each timestamp (i.e., relative frequency) is provided in Fig. 4. The probability distribution is shown in a discrete representation to emphasize the differences among years. The relative frequencies are preferred to absolute values (i.e., counts in histograms) as they enable the visual comparison of the distributions. For each graph, the color of the bars refers to the class distribution (i.e., green for benign software and red for malware). The X-axis provides the year of the bar data, while the Y-axis provides the relative frequency for each year. It is worth noticing that the horizontal range is not the same for all graphs, whereas the vertical range is the same. This keeps the proportions on the Y-axis comparable while showing the whole range of values the distribution encompasses on the X-axis. The only exception is the *LM* graph, where only the range between 2008–2020 is provided due to the negligible proportion of *outliers* or non-valid values, which were not observable when plotting the whole 1980–2020 range. Furthermore, an enhanced visual comparison between *LM* and *FS* (i.e., the most *suitable* timestamps) is enabled when only the *valid* range is plotted.

As can be observed in the graphs in Fig. 4, the *internal* timestamps (i.e., *EM, LM, DD, MD)* seem to provide similar data distributions. The *LM*, however, does not show the large proportion of data points (i.e., around 10%) located in 1980 that the other *external* distributions have, but the distributions in the valid range



(a) EM probability distribution (b) LM probability distribution (c) FS probability distribution

(d) FSW probability distribution (e) DD probability distribution (f) MD probability distribution

Fig. 4. Probability distribution of each timestamp.

are relatively similar, especially when compared with *DD* and *MD*. It is worth noticing that due to different X-axis ranges the visual comparison is hindered. The *FSW* data distribution is radically different from the other distributions, showing malware data concentrated in the 2008–2016 range and legitimate data in the 2014–2019 range and also in 2010. An interesting observation occurs when *LM* and *FS* are compared, which motivates the plotting of *LM* only for the *valid* range. As can be observed, the two distributions seem relatively similar for legitimate data, peaking in one year and showing relatively low figures for the other years. However, the peak occurs in 2011 for *LM* and 2012 for *FS*. In the case of malware, the three consecutive bars around 0.2 probability value occur in both distributions in the range 2012–2014. However, the dispersion of data surrounding this range is distinct, with many samples in the years before this range for *LM* and in the years after this range, especially in 2018, for *FS*. These observations may suggest that the relatively similar but shifted shapes might have been caused by a *delay* in the *FS* timestamp regarding the *LM* timestamp. Further exploration of this hypothesis is addressed in the following paragraphs, using statistical means, and in Section 5.2.6.

The statistical analysis of timestamps distributions enables the assessment of their similarity, which provides a notion of the degree of variability among them. For this purpose, *Jensen–Shannon distance* (i.e., JSD) and *Kolmogorov–Smirnov 2-sample test* (i.e., KS) were used. The former uses the notion of distance between distributions to provide a similarity score, bounded in the [0, 1] interval, where higher values report greater dissimilarity, while the latter compares the experimental cumulative density distributions to statistically infer if the distributions are likely to belong to the same population, thus being similarly distributed. In this case, the *p-value* is used to assess the statistical significance of the results by accepting or rejecting the null hypothesis (i.e., the distributions are equal) at a specific confidence level α. Thus assessing the similarity between the distributions. In general, small *p-values* indicate a high probability that the distributions come from the same population, thus reporting a greater similarity between the timestamps to locate the data within the Android historical timeline.

As a result, both metrics provide complementary measurements to assess the similarity of the distributions. If distributions are similar, we would expect a *JSD* value close to 0 and a *KS* value close to 1. If they are significantly different, the *JSD* value is expected to be close to 1 and the *KS* value close to 0. Both measures are symmetric, meaning that the order used to compare the distributions does not matter (i.e., $d(P, Q) = d(Q, P)$).

The matrix in Fig. 5 provides the comparison among all pairs of timestamp distributions for the benign data. Given the symmetry of the calculated measures, they enable us to provide the computed values for both similarity measures in the form of a matrix where the main diagonal is left blank to separate the values. The values above the diagonal of the matrix provide the values for *KS* computations, while the values below the diagonal provide the obtained *JSD* values.

As can be observed in Fig. 5, all the timestamps seem to provide different distributions for the data. The only exception is for the pair *DD-MD,* which has an *almost* 0 distance (i.e., *almost* perfect similarity) and a KS of 1. These values strongly suggest that these two distributions are roughly the same. This fact was also spotted on the graphical visualization in Fig. 4. Furthermore, *DD* and *MD* have a high degree of similarity (i.e., they show a small distance and large KS value) with *LM* and, to a lesser extent, with *EM*. This fact shows the relatively close distance between the distributions based on *internal* timestamps, which confirms the spotted similarities in the plots. However, *EM* appears to have a significantly different cumulative function as reflected by the small *p-values*, probably caused by the large number of *invalid* values included in this distribution and the higher peaks in the early years (i.e., before 2015) observed in the graphical depiction of the distribution. In the case of *DD* and *MD*, their distributions can be interpreted as *almost* equivalent. Nevertheless, *LM* is preferred as it provides

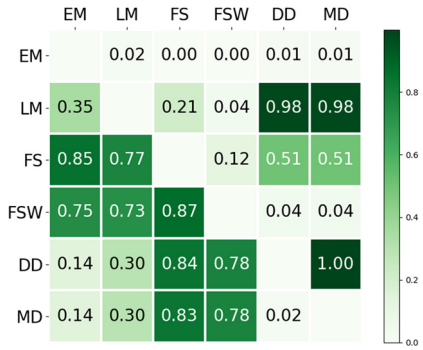


Fig. 5. JSD-KS matrix for benign data.

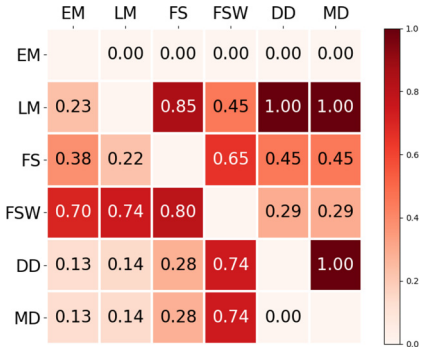


Fig. 6. JSD-KS matrix for malware data.

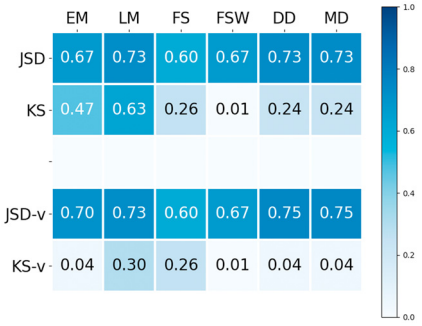


Fig. 7. JSD-KS inter-class.

more *suitable* timestamps than *DD* and *MD*. Regarding the *external* timestamps, *FS* and *FSW* show unique distributions, as evidenced by the large distance values and small p-values with almost all the other distributions. Despite having the same origin (i.e., VirusTotal), they are significantly different from each other and all other distributions, especially in the case of *FSW* (i.e., all *p-values* are near 0). When the most *suitable* timestamps are compared, *FS* and *LM* seem to have a great dissimilarity (i.e., a distance of 0.77, so they locate samples differently) but show a relatively low p-value (i.e., 0.21), which confirms the interesting observation from the graphs. Their cumulative functions are close enough to be found relatively similar using the KS-test but remarkably different when the distance-based similarity is used.

The matrix in Fig. 6 provides the comparison among all pairs of timestamp distributions for the malware data. The area above the diagonal provides the KS values, while the area below the diagonal provides the JSD values.

As can be seen in Fig. 6, the overall situation is similar to the benign case. The *external* timestamps show the greatest similarity, with *DD* and *MD* being almost identical to each other and also to *LM*. Again, *EM* shows to be relatively different from all other distributions, emphasized by the large proportion of data points in the *invalid* range. The *external* timestamps differ significantly, but less than for the benign data. An important exception to the similar trends with the benign data emerges when comparing *FS* and *LM*. For the malware case, these two distributions have a smaller distance and a significantly higher *p-value*. This fact shows that the range and overall shape of the distribution are similar and also that the cumulative function of the distributions is relatively similar. This fact supports the hypothesis of the *delay* between them and that even though they may provide similar distribution of malware data along the historical timeline, there is more concentration of samples in the early years for *LM* and in the latter years of the valid range for *FS*.

The previous statistical analysis compared the distribution of data according to timestamps for the same class (i.e., benign and malware). An interesting comparison is also the analysis of the similarity of class distributions within a given timestamp. The results are reported in Fig. 7, where the columns provide information about the specific timestamp and the horizontal rows about the statistical value computed when comparing the benign and malware distribution for that specific timestamp (i.e., *JSD* or *KS*). The upper rows in the figure provide the comparative results of *JSD* and *KS* for the whole time frame (i.e., whole distributions), whereas the lower rows provide the same information but just for the *valid* time frame, indicated as *JSD-v* and *KS-v*.

The overall interpretation of Fig. 7 is that the *valid* time frame emphasizes the differences between class distributions. In general, the values of distance increase and *p-values* diminish in the lower rows (i.e., *valid* range) compared with the upper rows (i.e., including the *invalid* range). The only exceptions to this are the *FS* and *FSW* timestamps which have the same values on both pairs of rows as they are always *valid*. Therefore, the class-wise distributions are significantly different across all timestamps.

5.2.5. Accuracy: a measure of historical context reliability

The evaluation of timestamp accuracy (i.e., how precise the location of a sample is within the Android historical timeline) is a significant challenge due to the absence of an exact ground-truth timestamp (i.e., it is not possible to ascertain when the malware sample was first released). In this regard, only approximations to the ground-truth timestamp can be achieved. This approximation might be based on external information such as open-source intelligence (OSINT) feeds, discovery reports of specific malware families by antivirus vendors, or media news. Therefore, the assessment of timestamps' accuracy and reliability is hindered and can only be approximated using these OSINT sources, which might be relatively *delayed* and not fully precise. In this research, this approach was used to obtain an approximation of the reliability of the timestamps evaluated. In this regard, the first *discovery* report released for specific malware families was used to contextualize each malware family within the historical timeline. The first two columns in Table 4 show 10 malware families (i.e., one per row) and *reference* dates as a context of the discovery time frame based on the reports (i.e., month/year). The data sources are provided in square brackets. They were taken from reliable sources when possible and contrasted with other media feeds. The following 6 columns are split into three sub-columns which are referred to as ± 6, > 6, and NV. For the sake of interpretation of the table, these columns have been colored in green, yellow, and grey, respectively. These columns show the proportion of data samples (i.e., percentages) of the data set dated with each specific timestamp that *lies within the reference value ±6 months* (i.e., *±6* column), *beyond the reference value + 6 months* (i.e., *>6* column) and that has a *non-valid* location (i.e., *NV* column). Note that the values *before the reference time - 6 months* are not provided but can be computed by summing the provided proportions and subtracting to 100. The rationale behind these computed proportions is the following. A precise timestamp should locate most of the samples of a specific family in the ± 6 and > 6 range, which is defined as the *valid* range for the specific family. The proportion of NV values and samples located *before* the valid range should be minimal (i.e., ideally zero). A larger proportion of values in the > 6 range may imply a *delay* in the timestamp or denote family evolution. The ± 6 range gives a notion of the amount of samples within this range, but due to family evolution it can only be interpreted in comparison with the other values, as the data set may contain fewer *original* samples than evolved samples. Furthermore, malware family denominations are not consistent among AV vendors or even an-

Table 4
Accuracy analysis of timestamps to locate specific malware families.

Family	Reference	EM			LM			FS			FSW			DD			MD		
		6	> 6	NV	6	> 6	NV	6	> 6	NV	6	> 6	NV	6	> 6	NV	6	> 6	NV
Geinimi	11/10 [90]	31.3	43.8	6.3	56.3	43.7	0	12.5	87.5	0	18.8	0	56.3	56.3	43.7	0	56.3	43.7	0
DroidDream	03/11 [91]	65.9	19.8	0	67	28.6	0	14.3	85.7	0	16.5	4.4	61.5	67	28.6	0	67	28.6	0
DroidKungFu	06/11 [92]	66.2	13.1	11.9	68	31.6	0.3	7.6	92.4	0	2.5	16.5	54	68.3	31.4	0.3	68.3	31.4	0.3
Plankton	06/11 [93]	23.8	71.4	1.6	22.2	77.8	0	6.3	93.7	0	0	1.6	92.1	22.2	77.8	0	22.2	77.8	0
GinMaster	08/11 [94]	21.2	70.4	6.6	17.6	81.5	0.2	0.7	98.9	0	5.2	3.9	69.8	18.4	80.7	0.2	18.4	80.7	0.2
AnserverBot	09/11 [92]	96.3	0	1.0	99.7	0	0	42.1	57.9	0	0.3	47.2	39.1	99.7	0	0	99.7	0	0
Slocker	05/14 [95]	23.2	50.3	25.8	23.1	57.9	18.4	1.1	98.5	0	0.1	1.1	93.9	23.2	55.1	21.1	23.3	54.4	21.7
MobiDash	01/15 [96]	3.3	60.1	36.7	3.3	90.2	6.5	0.7	99.3	0	0.7	1.3	96.7	3.3	60.8	35.9	3.3	60.1	36.6
BankBot	01/16 [97]	60.4	17.3	12.3	60.5	20.4	9.2	71.3	27	0	4	0.2	76	60.5	20.2	9.4	60.5	19.6	10
Triada	03/16 [98]	41.7	0	58.3	41.7	16.7	41.6	41.7	58.3	0	41.7	8.3	50	41.7	16.7	41.6	41.7	16.7	41.6
Total	-	43.3	34.6	16.1	45.9	44.9	7.6	19.8	80	0	9	8.5	69	46	41.5	10.9	46	41.3	11

References [90–98] citation mentioned in Table 4 (F-secure, 2021b; F-secure, 2021a; Jiang and Zhou, 2013; Shipman, 2011; Yu, 2013; Jiang and Zhou, 2013; Lipovsky et al., 2017; Kiss et al., 2016; Dr.Web, 2018; Buchka and Kuzin, 2016).

alysts, thus being a handicap for any malware family analysis. In our case, it is assumed that most of the labels are *certain*, which provides a relative degree of flexibility in interpreting the results. The last row of the table computes the *totals* or average value of each column for each specific timestamp.

As can be observed in Table 4, the individual proportions for specific malware families greatly vary among timestamps. For instance, *AnserverBot* seems to be well captured by all external timestamps in the reference ±6 months time frame, which may imply that the *reference* might be precise for the family discovery date and corresponding outbreak, and that these timestamps precisely locate this malware family. A different situation happens with the *MobiDash* family, where almost all timestamps convey the idea that the outbreak of this family happened in a later time frame, but also that the initial reference might be precise as the sum of the three proportions is 100% (i.e., no samples *dated* before the reference time frame). However, in general, individual values greatly vary across families and timestamps. A better and broader picture is provided by the *total* values. Even though the average value might be significantly biased due to *outlier* values, it is a good indicator of the overall trend. Based on the interpretation of the *total* values, it can be stated that the *LM* timestamp provides the most desirable properties of accuracy. It has a very low ratio of *non-valid* values and a high proportion of timestamps within the *valid* time frame (i.e., sum of values in the time frame encompassed by *reference* ±6 and > 6). The average values also confirm that the internal timestamps show similar distributions, with all of them showing similar proportions but with significantly lower *non-valid* values for the *LM* timestamp. The external timestamps show completely different pictures. The *FS* is characterized by always providing *valid* values, whereas the *FSW* shows a large proportion of *non-valid*, which correspond to *missing* data in this case and not actual non-valid values. Finally, when the *FS* and *LM* timestamps are compared, the average values show that *FS* captures most of the data beyond the *reference + 6 months* (i.e., > 6), whereas *LM* does it in similar proportions on both *valid* time frames. This supports the delayed nature of *FS* to capture malware outbreaks and the goodness of *LM* to locate most data samples with improved precision.

The results shown in Table 4 and the analysis performed suggest that *FS* and *LM* are significantly better timestamps than the other analyzed approaches in terms of suitability, statistical properties, and accuracy. To further investigate their relationship, the next section analyzes statistically the differences between them and their relation over time.

5.2.6. Last modification vs. first seen: a comparative analysis

Figs. 8 and 9 provide the differences between both timestamps, computed for each data sample, separately for benign and malware data. The base *difference* unit is *days* and the base timestamp used is the *last modification*, so that the differences can be expressed in positive terms (e.g., +8 days). The assumption was that the *last modification* timestamp would place the sample more accurately in the Android historical timeline, usually earlier than *first seen*. Therefore, it was chosen as the reference time. These graphs report relevant descriptive statistics regarding the temporal differences (i.e., *Y*-axis) for the samples in the data set located in a specific period (i.e., six months chunks) using the *last modification* timestamp (i.e., *X*-axis) concerning the *first seen* timestamp. The data is split into chunks of 6 months data (i.e., period) for better interpretation of the results and deeper exploration of the differences. Only the *valid* time frame is plotted (i.e., from 2009 to

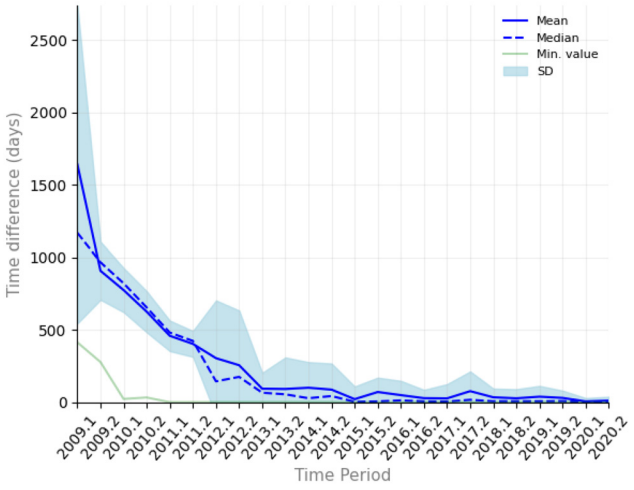


Fig. 8. Differences *LM-FS* for benign data.

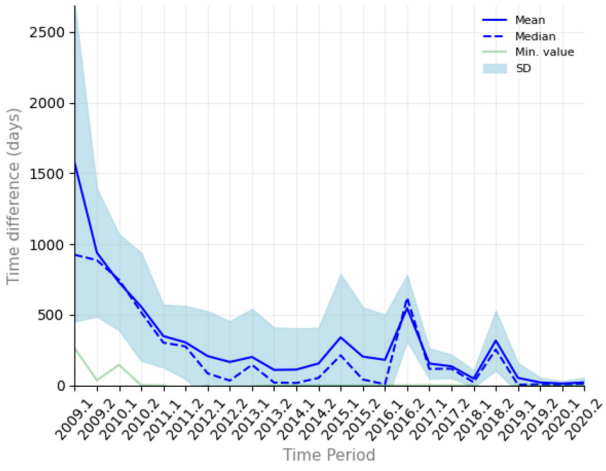


Fig. 9. Differences *LM-FS* for malware data.

2020). The semesters are referenced as appended suffixes to the year figure (e.g., *2009.1* reflects the first six months of 2009). The temporal differences are calculated individually per data sample and grouped into data periods. Descriptive statistics are calculated per period: *mean*, *median*, *minimum difference,* and *standard deviation*. In this regard, the solid blue line reports the *average* value for each period, while the dashed blue line provides the *median* value. These two central tendency measures report the average value of displacement of a sample per period. The green line provides the *minimum difference* value found in each period. The standard deviation, plotted as a blue area, conveys the average spread of differences around the mean for the data samples located in each chunk.

As expected, in both cases, a positive difference between both timestamps is observed. This evidences that the *first seen* timestamp locates the samples later in the timeline, thus *delayed* regarding the *last modification* timestamp. This fact is especially pronounced in the early years of Android history, with average differences of around 1500 days (i.e., four years) for both malware and benign applications. Thus it implies that an instance located in 2009.1 (i.e., the first semester of 2009) by the last modification timestamp might be located by the first seen timestamp in 2013.1 (i.e., the first semester of 2013). This significant difference in the temporal location of samples might impact the performance and adaptation capabilities of a detection system to deal with concept drift, as *first seen* may generate *artificial drift* by misplacing the data, which might be more complex to model effectively than real concept drift, generally smoother.

However, as can be observed in Figs. 8 and 9, the differences between these timestamps have been reducing over time, as evidenced by the monotonically decreasing mean value for benign instances and the significant decrease that has occurred in the case of malware instances, especially in recent years. This fact has made the timestamps more synchronized and closer over time, and even equivalent for the 2019–2020 time frame. For instance, 2020.1 and 2020.2 periods have mean values of 4.88 and 12.37 days and median values of 2 and 3 days, respectively, for benign samples, and average values of 15.9 and 16.45 days and medians of 5 and 11 days, respectively, in the case of malware samples. This is a dramatic change when compared to 2010.1, which shows a mean value of 728.4 days and a median of 747 days for malware, and an average of 774.3 days and a median of 821 days for benign data. As a result, the gap between both timestamping approaches to date samples has significantly decreased over time, making them converge and, consequently, increasing the reliability and accuracy of the *first seen* timestamp in more recent years (i.e., 2019–2020).

The convergence of both timestamps supports the hypothesis that the *last modification* timestamp is accurate and that it is *rarely* tampered with by attackers. Consequently, if the system has to learn from past data and predict about past samples, it might be safer to use the *last modification* timestamp, whereas, if the system uses mainly recent data, the convergence of the timestamps implies that both approaches could be appropriate and perform similarly against data drift. Furthermore, if data tampering is a major concern, the usage of *first seen* ensures that the data have not been tampered with, even though a delay should always be assumed.

5.3. Concept drift handling

For the purpose of this research, the existence of concept drift is assumed and not proven. The concept drift phenomenon has already been proved and explored in previous research for the feature spaces used in this study: system calls, permissions, and API calls (Guerra-Manzanares et al., 2022a; Onwuzurike et al., 2019; Xu et al., 2019).

Table 5
Data set size per timestamp in the period 2011.2-2018.1.

Timestamp	Malware	Benign	Total
EM	33,346	7602	40,948
LM	38,496	13,456	51,952
FS	40,376	32,870	73,246
FSW	2,137	116	2,253
DD	36,805	11,555	48,360
MD	36,810	11,500	48,310

The purpose of this experimental scenario was to evaluate the impact and adaptive response generated by the appearance of concept drift, which is distinct for different timestamps and feature spaces, and analyze what approach was better to model *natural* concept drift, which is assumed to be mostly a relatively *smooth* transition with eventual *sudden* drifts. The sudden changes may occur, for instance, when a new malware outbreak occurs and the ML model has to learn about the new threat, which is not similar to previous data, and, consequently, adapt to it by updating its knowledge.

The *adaptive* classifier used in this study requires the selection of hyper-parameters such as the *chunk size* (i.e., number of samples per chunk), *pool size* (i.e., number of classifiers in the pool), and *time constraint* (i.e., max. time frame of data included in the chunks). Based on the data distribution and experimental tests, a good set of hyper-parameters was 4000 samples per chunk, 12 classifiers in the pool, and 3 months of data per chunk. All the classifiers in the pool were Random Forest models, which have proved successful for the task in similar studies (Guerra-Manzanares et al., 2019b). As the data set is imbalanced towards the malware label, the random oversampling technique was applied to the training chunks to balance the classes and avoid a biased classifier.

Due to the characteristics of *KronoDroid* data and the demands of machine learning models, it was not possible to use the whole Android historical timeline to build effective ML-based classifiers per quarter (i.e., there is not enough data in the early years or more recent years), so the experimental setup was restricted to the period encompassing from the second semester of 2011 until the first semester of 2018. This time frame spans 7 years of Android history, including the most active years of Android malware development (AV-Test, 2021; Johnson, 2021).

The available data per timestamp for the selected period (i.e., from 2011.2 to 2018.1) is provided in Table 5.

As can be noticed in Table 5, *FS* provides most of its data within this range, whereas the external timestamps provide lower proportions of samples (i.e., especially in the case of *EM*). As expected, the data is imbalanced towards the malware class, thus justifying the usage of a data set balancing technique to avoid any class bias from the classifier. Finally, as the data provided by the *FSW* timestamp is not enough to build a single classifier, this timestamp was discarded and not used in the following experimental setups.

Three experimental setups are described in the following sections. The distinct feature spaces were explored individually and used as input features to the *adaptive* classifier algorithm. The data was split into chunks and processed sequentially. The timestamps were used to place the data samples into consecutive chunks, and the system performance was retrieved. The *F1* score performance metric is provided for each induced *adaptive* classification model (i.e., one per each combination of a timestamp and feature space). It is worth noticing that no hyper-parameter optimization was performed as the main aim was to test the same scenario for all timestamps and compare the outcome, so no model was optimized to ensure the same conditions. The performance graphs provide each model performance with different color and/or line style. *EM,*

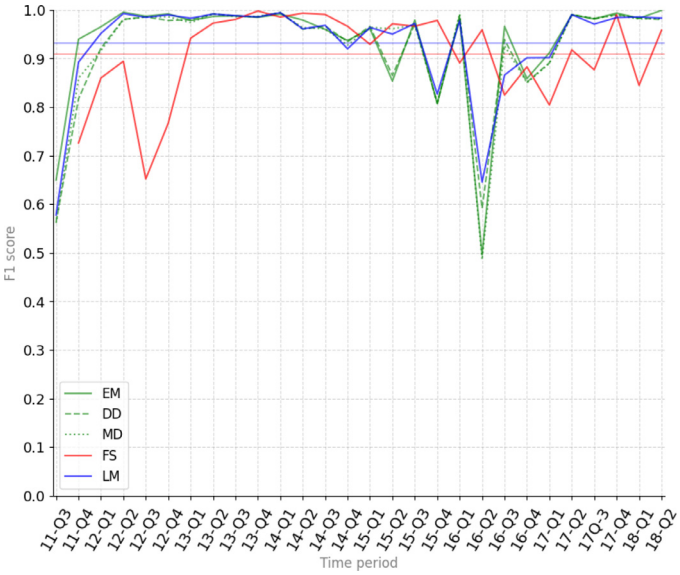


Fig. 10. F1 performance of permissions-based models using different timestamps.

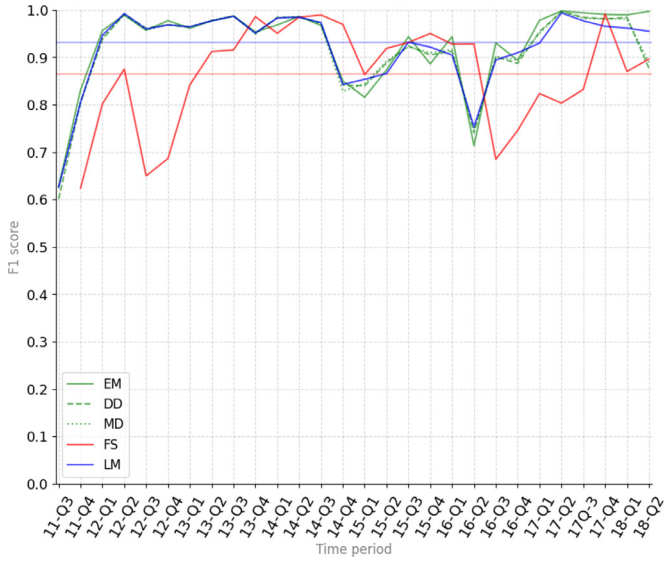


Fig. 11. F1 performance of system calls-based models using different timestamps.

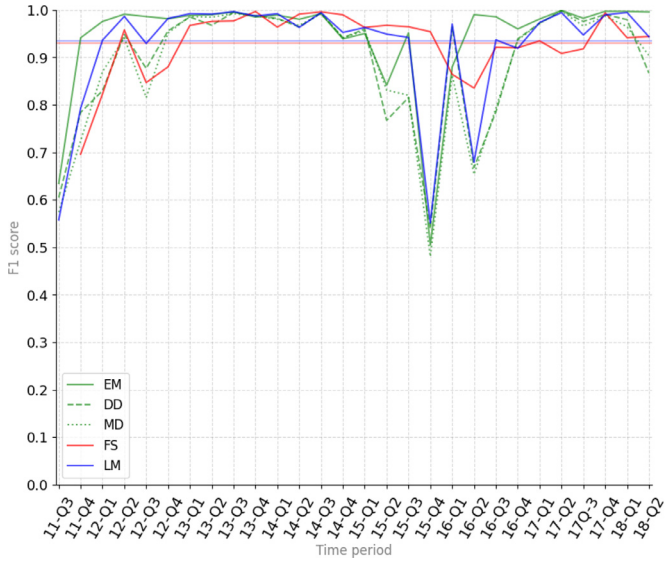


Fig. 12. F1 performance of API calls-based models using different timestamps.

DD and *MD* are provided in green color but with solid, dashed, and dotted line styles, respectively. *LM* is reported with a solid blue line while *FS* with a solid red line. As the most *suitable* options were *LM* and *FS*, their average performance for the whole period is reported with a horizontal overlay line, *LM* with a solid blue line and *FS* with a solid red line.

5.3.1. Permissions

The performance results of the models built for the permissions feature space and using the different timestamps are provided in Fig. 10. The permissions feature space is defined by categorical data and is the smallest of the analyzed ones (i.e., 166 dimensions). As can be observed, despite two sudden drops (and consequent recovery), the smoother line is provided by the *LM* timestamp. The other external timestamps provide similar performance but describe a more fluctuating surface. The *FS* timestamp performance is not smooth, characterized by several sudden peaks and bottoms in neighboring quarters, especially at the beginning and the end of the analyzed time frame (i.e., bumpy red line). Furthermore, it has the lowest average in the analyzed period. Despite that, the overall performance of all models is over 0.90 F1, which reflects the goodness of the system to deal with concept drift, and especially with *natural* data drift, which is better captured by the *internal* timestamps.

5.3.2. System calls

The performance of the models built using the system calls feature space is provided in Fig. 11. The system calls feature space is numeric and larger than the permissions space (i.e., 288 dimensions). Similar to Fig. 10, the external timestamps provide smoother performance lines, and, again, the *LM* timestamp seems to yield the best performance. *LM* enables the model to capture better the natural data drift, showing quick recovery after sudden data drifts. However, in this case, *EM* achieves similar performance over the whole range but shows a more irregular performance line. As in the permissions space, *FS* provides the worst performance and is characterized by sudden dips and peaks, likely caused by *artificial* data drift. An interesting fact in this plot is that from 13-Q3 until 16-Q3, *LM* and *FS* seem to perform synchronously. The *FS* performance line is relatively similar but delayed one quarter regarding *LM* and reaches more extreme values. This goes in line with the *median* differences observed in this time frame in Figs. 8 and 9 (i.e., below 90 days for both classes). Lastly, in this case, the difference between *FS* and *LM* average performance is significantly larger, with the average performance of *LM* of around 0.93 and at the 0.87 level for *FS*.

5.3.3. API calls

The performance of the models built using the API calls feature space is provided in Fig. 12. The API calls feature space is the largest feature space, with over 53,523 features. Similar to the other feature spaces, the performance of the internal timestamps is rather similar, and over the performance of *FS*. However, in this case, two deep dips are observed for the internal timestamps that are not observed for *FS*. It is worth noticing that, in the case of high-dimensionality spaces, the quantity of data is critical to building effective models (i.e., data density is needed to generate precise classification boundaries), a phenomenon called the *curse of dimensionality*. As reflected in Table 5, the data samples available for the internal timestamps are fewer and significantly reduced for these specific periods. Therefore, reduced performance might be observed due to insufficient data to cover the feature space effi-

ciently. However, despite the deep dips in those specific periods, the average performance of *LM* still outperforms *FS*.

6. Discussion

Timestamp selection has not been explored in concept drift-related research, where some common approaches are usually used without considering alternatives. However, depending on the timestamp selected, the same app might be located in different temporal contexts within the Android historical timeline. The position of the apps along the timeline determines the observed drift, whether natural or artificial. Therefore, timestamp selection is critical to handle concept drift effectively.

This research has demonstrated that the selection of the timestamp is an important decision in building long-lasting machine learning-based Android malware detection systems that adapt and learn over time about drifting data. The most common approaches for timestamps used in research studies, such as *FS* and *DD*, have been shown sub-optimal in tackling emerging concept drift. Furthermore, the test scenarios and statistical analysis suggest that *FS* should not be employed when *old* data is used, as it can misplace data samples along the historical timeline, generating artificial data drift that cannot be modeled efficiently by machine learning systems due to the randomness and noise added to the detection task. In the seek for alternatives, *LM* has emerged as a reliable and suitable timestamp that outperforms other options to handle concept drift in the Android malware detection task. Despite showing an inherent tampering risk, as it is extracted from the inner files of the *apk* and, consequently, in the scope of the attacker, it has been demonstrated to be robust and accurate. Other alternatives such as *DD, MD,* and *EM* can provide similar performance, but as their suitability is lower, fewer data samples can be used to address the drift, thus providing less effective results. Furthermore, the usage of specific file-related timestamps might be riskier as there is no guarantee that the timestamps are set for those specific files (du Luxembourg, 2021). A potential benefit of the usage of *LM* or even *EM* is that they do not rely on a single file to date the application but on any of the inner files, which may overcome the single-file dependency of the *DD* and *MD* timestamps.

The implications of our findings are relevant for any Android malware detection solution seeking long-term reliable performance. Most research studies focus on the optimization of their systems in a restricted testing set, disregarding temporal context and data evolution. However, the features used for Android malware detection are likely to suffer concept drift at any point, as the dynamism of the malware generation phenomenon (e.g., new malware trends or new malware capabilities) and the evolution of the Android framework (e.g., changes in API calls or permissions) are directly reflected in the features. Thus optimizing for static historical data sets does not guarantee effective future performance. The system might not be able to predict well future and *unknown* data, which is the main objective of machine learning-based detection systems. Furthermore, the traditional random split of data into testing/training sets generates impossible data configurations and yields over-inflated performance metrics caused by subtle but harmful *data snooping*.

This research has shown that, in general, inner timestamps might be more appropriate to build models in which past data is involved, whereas, for recent data, *FS* might be a good approach as the temporal gap between samples has been reducing over time, thus increasing the accuracy of *FS*. In any case, timestamp selection is critical to achieving long-lasting, high-performance Android malware detection systems.

The main limitation of our study is that the results strictly rely on the reliability of the data used. Even though the size of the data set is large enough to enable statistical inference, the *soft* labels of the data set were used, which are not as *certain* as the *hard* labels. We chose the *soft* label to perform our analysis as more data was available, so the error due to the class label might be minimized by increased data quantity. Furthermore, the malware family analysis relied on the malware family label provided by the data set (Guerra-Manzanares et al., 2021). However, malware family naming conventions vary significantly among antivirus vendors. Therefore, the accuracy of the malware family label is not fully guaranteed. To minimize this issue, for the malware family analysis, old and well-known malware families were used as it might increase the certainty around the actual malware family label for the sample.

7. Conclusions

The vast body of Android malware detection research has neglected the impact of concept drift in Android malware research. Timestamps, critical elements for concept drift handling, have not received the deserved attention in the related literature. Our study performed an extensive benchmarking about timestamp options and their capabilities to deal with concept drift effectively in different feature spaces. Our results show that timestamp selection is a critical decision and that the *last modification* and *first seen* timestamps are the best options to build effective, long-lasting ML models for Android malware detection under data evolution challenges.

Declaration of Competing Interest

We wish to confirm that there is no known conflict of interest associated with this publication and there has been no significant support for this work that could have influenced its outcome.

CRediT authorship contribution statement

Alejandro Guerra-Manzanares: Conceptualization, Methodology, Formal analysis, Software, Validation, Investigation, Visualization, Writing – original draft, Writing – review & editing. **Hayretdin Bahsi:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing.

References

Abderrahmane, A., Adnane, G., Yacine, C., Khireddine, G., 2019. Android malware detection based on system calls analysis and CNN classification. In: 2019 IEEE Wireless Communications and Networking Conference Workshop (WCNCW). IEEE, pp. 1–6.

Afonso, V.M., de Amorim, M.F., Grégio, A.R.A., Junquera, G.B., de Geus, P.L., 2015. Identifying android malware using dynamically obtained features. J. Comput. Virol. Hack. Tech. 11 (1), 9–17.

Aggarwal, C.C., 2015. Data Mining: The Textbook. Springer.

Ahsan-Ul-Haque, A., Hossain, M.S., Atiquzzaman, M., 2018. Sequencing system calls for effective malware detection in android. In: 2018 IEEE Global Communications Conference (GLOBECOM). IEEE, pp. 1–7.

Allix, K., Bissyandé, T.F., Klein, J., Le Traon, Y., 2015. Are your training datasets yet relevant? In: International Symposium on Engineering Secure Software and Systems. Springer, pp. 51–67.

Alzaylaee, M.K., Yerima, S.Y., Sezer, S., 2017. Emulator vs. real phone: android malware detection using machine learning. In: Proceedings of the 3rd ACM on International Workshop on Security and Privacy Analytics, pp. 65–72.

Alzaylaee, M.K., Yerima, S.Y., Sezer, S., 2020. Dl-droid: deep learning based android malware detection using real devices. Comput. Secur. 89, 101663.

Amin, M.R., Zaman, M., Hossain, M.S., Atiquzzaman, M., 2016. Behavioral malware detection approaches for android. In: 2016 IEEE International Conference on Communications (ICC), pp. 1–6. doi:10.1109/ICC.2016.7511573.

Android, 2021a. App manifest overview. https://developer.android.com/guide/topics/manifest/manifest-intro.

Android, 2021b. Package index. https://developer.android.com/reference/packages.

Arp, D., Quiring, E., Pendlebury, F., Warnecke, A., Pierazzi, F., Wressnegger, C., Cavallaro, L., Rieck, K., 2020. Dos and don'ts of machine learning in computer security. arXiv preprint arXiv:2010.09470

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C., 2014. Drebin: effective and explainable detection of android malware in your pocket. In: Ndss, vol. 14, pp. 23–26.

AV-Test, 2021. Malware. https://www.av-test.org/en/statistics/malware/.

Barbero, F., Pendlebury, F., Pierazzi, F., Cavallaro, L., 2020. Transcending transcend: revisiting malware classification with conformal evaluation. arXiv preprint arXiv:2010.03856

Bhatia, T., Kaushal, R., 2017. Malware detection in android based on dynamic analysis. In: 2017 International Conference on Cyber Security And Protection Of Digital Services (Cyber Security), pp. 1–6. doi:10.1109/CyberSecPODS.2017.8074847.

Buchka, N., Kuzin, M., 2016. Attack on zygote: a new twist in the evolution of mobile threats. https://securelist.com/attack-on-zygote-a-new-twist-in-the-evolution-of-mobile-threats/74032/.

Buczak, A.L., Guven, E., 2015. A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Commun. Surv. Tutor. 18 (2), 1153–1176.

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

Cai, H., 2020. Assessing and improving malware detection sustainability through app evolution studies. ACM Trans. Softw. Eng. Methodol. (TOSEM) 29 (2), 1–28.

Cai, H., Fu, X., Hamou-Lhadj, A., 2020. A study of run-time behavioral evolution of benign versus malicious apps in android. Inf. Softw. Technol. 122, 106291. doi:10.1016/j.infsof.2020.106291. https://www.sciencedirect.com/science/article/pii/S0950584920300410

Cai, H., Meng, N., Ryder, B., Yao, D., 2019. Droidcat: effective android malware detection and categorization via app-level profiling. IEEE Trans. Inf. Forensics Secur. 14 (6), 1455–1470. doi:10.1109/TIFS.2018.2879302.

Cai, L., Li, Y., Xiong, Z., 2021. Jowmdroid: android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters. Comput. Secur. 100, 102086.

Cai, L., Li, Y., Xiong, Z., 2021. Jowmdroid: android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters. Comput. Secur. 100, 102086.

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

Casolare, R., De Dominicis, C., Iadarola, G., Martinelli, F., Mercaldo, F., Santone, A., 2021. Dynamic mobile malware detection through system call-based image representation. J. Wirel. Mob. Netw. Ubiquitous Comput. Dependable Appl. 12 (1), 44–63.

Da, C., Hongmei, Z., Xiangli, Z., 2016. Detection of android malware security on system calls. In: 2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC), pp. 974–978. doi:10.1109/IMCEC.2016.7867355.

Desnos, A., Gueguen, G., Bachmann, S., 2018. Androguard. https://androguard.readthedocs.io/en/latest/index.html.

Dimjašević, M., Atzeni, S., Ugrina, I., Rakamaric, Z., 2016. Evaluation of android malware detection based on system calls. In: Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics, pp. 1–8.

Dr.Web, 2018. Doctor web: banking trojan android.bankbot.149.origin has become a rampant tool of cybercriminals. https://news.drweb.com/show/?i=11772.

F-secure, 2021a. Trojan:android/droiddream.a. https://www.f-secure.com/v-descs/trojan_android_droiddream_a.shtml.

F-secure, 2021b. Trojan:android/geinimi. https://www.f-secure.com/v-descs/trojan_android_geinimi.shtml.

Feizollah, A., Anuar, N.B., Salleh, R., Wahab, A.W.A., 2015. A review on feature selection in mobile malware detection. Digit. Invest. 13, 22–37.

Feng, P., Ma, J., Sun, C., Xu, X., Ma, Y., 2018. A novel dynamic android malware detection system with ensemble learning. IEEE Access 6, 30996–31011. doi:10.1109/ACCESS.2018.2844349.

Ferrante, A., Medvet, E., Mercaldo, F., Milosevic, J., Visaggio, C.A., 2016. Spotting the malicious moment: characterizing malware behavior using dynamic features. In: 2016 11th International Conference on Availability, Reliability and Security (ARES). IEEE, pp. 372–381.

Frenklach, T., Cohen, D., Shabtai, A., Puzis, R., 2021. Android malware detection via an app similarity graph. Comput. Secur. 109, 102386.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A., 2014. A survey on concept drift adaptation. ACM Comput. Surv. (CSUR) 46 (4), 1–37.

Gao, H., Cheng, S., Zhang, W., 2021. Gdroid: android malware detection and classification with graph convolutional network. Comput. Secur. 106, 102264.

Google, 2008. Android market: now available for users. https://android-developers.googleblog.com/2008/10/android-market-now-available-for-users.html.

Google, 2021. Google play protect. https://developers.google.com/android/play-protect.

Guerra-Manzanares, A., Bahsi, H., Nõmm, S., 2021. Kronodroid: time-based hybrid-featured dataset for effective android malware detection and characterization. Comput. Secur. 102399.

Guerra-Manzanares, A., Bahsi, H., Nõmm, S., 2019a. 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. doi:10.1109/IOTSMS48152.2019.8939268.

Guerra-Manzanares, A., Luckner, M., Bahsi, H., 2022a. Android malware concept drift using system calls: detection, characterization and challenges. Expert Syst. Appl. 117200. doi:10.1016/j.eswa.2022.117200. https://www.sciencedirect.com/science/article/pii/S0957417422005863

Guerra-Manzanares, A., Luckner, M., Bahsi, H., 2022b. Concept drift and cross-platform behavior: challenges and implications for effective android malware detection. Comput. Secur. 120, 102757.

Guerra-Manzanares, A., Nomm, S., Bahsi, H., 2019b. In-depth feature selection and ranking for automated detection of mobile malware. In: ICISSP, pp. 274–283.

Guerra-Manzanares, A., Nõmm, S., Bahsi, H., 2019c. 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). IEEE, pp. 1–8.

Hei, Y., Yang, R., Peng, H., Wang, L., Xu, X., Liu, J., Liu, H., Xu, J., Sun, L., 2021. Hawk: rapid android malware detection through heterogeneous graph attention networks. IEEE Trans. Neural Netw. Learn. Syst. 1–15. doi:10.1109/TNNLS.2021.3105617.

Hou, S., Saas, A., Chen, L., Ye, Y., 2016. Deep4maldroid: a deep learning framework for android malware detection based on Linux kernel system call graphs. In: 2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW), pp. 104–111. doi:10.1109/WIW.2016.040.

Irolla, P., Dey, A., 2018. The duplication issue within the drebin dataset. J. Comput. Virol. Hack. Tech. 14 (3), 245–249.

Islam, Z., 2021. Android malware on the rise, google's os is more "interesting" to cybercriminals than apple IoS. https://www.techspot.com/news/91519-android-more-interesting-average-cybercriminal-than-ios-malware.html.

Isohara, T., Takemori, K., Kubota, A., 2011. Kernel-based behavior analysis for android malware detection. In: 2011 Seventh International Conference on Computational Intelligence and Security. IEEE, pp. 1011–1015.

Jaiswal, M., Malik, Y., Jaafar, F., 2018. Android gaming malware detection using system call analysis. In: 2018 6th International Symposium on Digital Forensic and Security (ISDFS), pp. 1–5. doi:10.1109/ISDFS.2018.8355360.

Jang, J.-w., Yun, J., Woo, J., Kim, H.K., 2014. Andro-profiler: anti-malware system based on behavior profiling of mobile malware. In: Proceedings of the 23rd International Conference on World Wide Web, pp. 737–738.

Jiang, X., Zhou, Y., 2013. Android Malware. Springer.

Johnson, J., 2021. Development of new android malware worldwide from june 2016 to march 2020. https://www.statista.com/statistics/680705/global-android-malware-volume/.

Jordaney, R., Sharad, K., Dash, S.K., Wang, Z., Papini, D., Nouretdinov, I., Cavallaro, L., 2017. Transcend: detecting concept drift in malware classification models. In: 26th {USENIX} Security Symposium ({USENIX} Security 17), pp. 625–642.

Kapratwar, A., Di Troia, F., Stamp, M., 2017. Static and dynamic analysis of android malware. In: ICISSP, pp. 653–662.

Kiss, N., Lalande, J.-F., Leslous, M., Tong, V.V.T., 2016. Kharon dataset: android malware under a microscope. In: The {LASER} Workshop: Learning from Authoritative Security Experiment Results ({LASER} 2016), pp. 1–12.

Lei, T., Qin, Z., Wang, Z., Li, Q., Ye, D., 2019. Evedroid: event-aware android malware detection against model degrading for IoT devices. IEEE Internet Things J. 6 (4), 6668–6680. doi:10.1109/JIOT.2019.2909745.

Lin, Y.-D., Lai, Y.-C., Chen, C.-H., Tsai, H.-C., 2013. Identifying android malicious repackaged applications by thread-grained system call sequences. Comput. Secur. 39, 340–350.

Lindorfer, M., Neugschwandtner, M., Platzer, C., 2015. Marvin: efficient and comprehensive mobile app classification through static and dynamic analysis. In: 2015 IEEE 39th Annual Computer Software and Applications Conference, vol. 2. IEEE, pp. 422–433.

Lipovsky, R., Stefanko, L., Branisa, G., 2017. Trends in android ransomware. https://www.welivesecurity.com/wp-content/uploads/2017/02/ESET_Trends_2017_in_Android_Ransomware.pdf.

Liu, Z., Wang, R., Japkowicz, N., Tang, D., Zhang, W., Zhao, J., 2021. Research on unsupervised feature learning for android malware detection based on restricted boltzmann machines. Future Gener. Comput. Syst. 120, 91–108.

Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., Zhang, G., 2018. Learning under concept drift: a review. IEEE Trans Knowl Data Eng 31 (12), 2346–2363.

du Luxembourg, U., 2021. Androzoo - lists of apks. https://androzoo.uni.lu/lists.

Mahindru, A., Sangal, A., 2021. Mldroid-framework for android malware detection using machine learning techniques. Neural Comput. Appl. 33 (10), 5183–5240.

Malik, S., Khatter, K., 2016. System call analysis of android malware families. Indian J. Sci. Technol. 9 (21), 1–13.

Margara, A., Rabl, T., 2018. Definition of Data Streams. Springer International Publishing, Cham, pp. 1–4.

McLaughlin, N., Martinez del Rincon, J., Kang, B., Yerima, S., Miller, P., Sezer, S., Safaei, Y., Trickel, E., Zhao, Z., Doupé, A., et al., 2017. Deep android malware detection. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, pp. 301–308.

Narayanan, A., Yang, L., Chen, L., Jinliang, L., 2016. Adaptive and scalable android malware detection through online learning. In: 2016 International Joint Conference on Neural Networks (IJCNN), pp. 2484–2491. doi:10.1109/IJCNN.2016.7727508.

Naval, S., Laxmi, V., Rajarajan, M., Gaur, M.S., Conti, M., 2015. Employing program semantics for malware detection. IEEE Trans. Inf. Forensics Secur. 10 (12), 2591–2604. doi:10.1109/TIFS.2015.2469253.

Onwuzurike, L., Mariconti, E., Andriotis, P., Cristofaro, E.D., Ross, G., Stringhini, G., 2019. Mamadroid: detecting android malware by building Markov chains of behavioral models (extended version). ACM Trans. Privacy Secur. (TOPS) 22 (2), 1–34.

Pendlebury, F., Pierazzi, F., Jordaney, R., Kinder, J., Cavallaro, L., 2019. {TESSERACT}: eliminating experimental bias in malware classification across space and time. In: 28th {USENIX} Security Symposium ({USENIX} Security 19), pp. 729–746.

Rathore, H., Sahay, S.K., Nikam, P., Sewak, M., 2021. Robust android malware detection system against adversarial attacks using q-learning. Inf. Syst. Front. 23 (4), 867–882.
Saif, D., El-Gokhy, S., Sallam, E., 2018. Deep belief networks-based framework for malware detection in android systems. Alex. Eng. J. 57 (4), 4049–4057.
Samsung, 2021. About knox. https://www.samsungknox.com/en/about-knox.
Saracino, A., Sgandurra, D., Dini, G., Martinelli, F., 2018. Madam: effective and efficient behavior-based android malware detection and prevention. IEEE Trans. Dependable Secure Comput. 15 (1), 83–97. doi:10.1109/TDSC.2016.2536605.
Sasidharan, S.K., Thomas, C., 2021. Prodroidan android malware detection framework based on profile hidden Markov model. Pervasive Mob. Comput. 72, 101336.
Sharma, T., Rattan, D., 2021. Malicious application detection in android—A systematic literature review. Comput. Sci. Rev. 40, 100373.
Shipman, M., 2011. More bad news: two new pieces of android malware—plankton and yzhcsms. https://news.ncsu.edu/2011/06/wms-android-plankton/.
Sihag, V., Vardhan, M., Singh, P., Choudhary, G., Son, S., 2021. De-lady: deep learning based android malware detection using dynamic features. J. Internet Serv. Inf. Secur. (JISIS) 11 (2), 34–45.
Singh, L., Hofmann, M., 2017. Dynamic behavior analysis of android applications for malware detection. In: 2017 International Conference on Intelligent Communication and Computational Techniques (ICCT), pp. 1–7. doi:10.1109/INTELCCT.2017.8324010.
Statista, 2021. Mobile operating system market share worldwide, July 2020–July 2021. https://gs.statcounter.com/os-market-share/mobile/worldwide.
Surendran, R., Thomas, T., Emmanuel, S., 2020. A tan based hybrid model for android malware detection. J. Inf. Secur. Appl. 54, 102483.
Tchakounté, F., Dayang, P., 2013. System calls analysis of malwares on android. Int. J. Sci. Technol. 2 (9), 669–674.
Tong, F., Yan, Z., 2017. A hybrid approach of mobile malware detection in android. J. Parallel Distrib. Comput. 103, 22–31.
Vidal, J.M., Orozco, A.L.S., Villalba, L.G., 2017. Malware detection in mobile devices by analyzing sequences of system calls. World Acad. Sci., Eng. Technol., Int. J. Comput., Electr., Autom., Control Inf. Eng. 11 (5), 594–598.
Vinod, P., Zemmari, A., Conti, M., 2019. A machine learning based approach to detect malicious android apps using discriminant system calls. Future Gener. Comput. Syst. 94, 333–350.
Wahanggara, V., Prayudi, Y., 2015. Malware detection through call system on android smartphone using vector machine method. In: 2015 Fourth International Conference on Cyber Security, Cyber Warfare, and Digital Forensic (CyberSec). IEEE, pp. 62–67.
Wang, X., Li, C., 2021. Android malware detection through machine learning on kernel task structures. Neurocomputing 435, 126–150.
Wei, F., Li, Y., Roy, S., Ou, X., Zhou, W., 2017. Deep ground truth analysis of current android malware. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, pp. 252–276.
Xiao, X., Fu, P., Xiao, X., Jiang, Y., Li, Q., Lu, R., 2015. Two effective methods to detect mobile malware. In: 2015 4th International Conference on Computer Science and Network Technology (ICCSNT), vol. 1. IEEE, pp. 1041–1045.
Xiao, X., Xiao, X., Jiang, Y., Liu, X., Ye, R., 2016. Identifying android malware with system call co-occurrence matrices. Trans. Emerg. Telecommun. Technol. 27 (5), 675–684.
Xiao, X., Zhang, S., Mercaldo, F., Hu, G., Sangaiah, A.K., 2019. Android malware detection based on system call sequences and LSTM. Multimed. Tools Appl. 78 (4), 3979–3999.
Xu, K., Li, Y., Deng, R., Chen, K., Xu, J., 2019. Droidevolver: self-evolving android malware detection system. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, pp. 47–62.
Yu, R., 2013. Ginmaster : a case study in android malware. https://www.virusbulletin.com/conference/vb2013/abstracts/ginmaster-case-study-android-malware.
Yu, W., Zhang, H., Ge, L., Hardy, R., 2013. On behavior-based detection of malware on android platform. In: 2013 IEEE Global Communications Conference (GLOBECOM), pp. 814–819. doi:10.1109/GLOCOM.2013.6831173.
Yuan, Z., Lu, Y., Wang, Z., Xue, Y., 2014. Droid-sec: deep learning in android malware detection. In: Proceedings of the 2014 ACM conference on SIGCOMM, pp. 371–372.
Zhang, X., Zhang, Y., Zhong, M., Ding, D., Cao, Y., Zhang, Y., Zhang, M., Yang, M., 2020. Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, pp. 757–770.
Zhou, Y., Jiang, X., 2012. Dissecting android malware: characterization and evolution. In: 2012 IEEE Symposium on Security and Privacy, pp. 95–109. doi:10.1109/SP.2012.16.
Zyblewski, P., Sabourin, R., Woźniak, M., 2021. Preprocessed dynamic classifier ensemble selection for highly imbalanced drifted data streams. Inf. Fusion 66 (June 2020), 138–154. doi:10.1016/j.inffus.2020.09.004.

Alejandro Guerra Manzanares is a Ph.D. candidate at the Center for Digital Forensics and Cyber Security, Department of Software Science, Tallinn University of Technology (Estonia). He received a BA degree in criminology from the Autonomous University of Barcelona (Spain) in 2013, and a BS degree in ICT engineering from the Polytechnic University of Catalonia (Spain) in 2017. In 2018, he received a M.Sc. in cyber security from Tallinn University of Technology (Estonia). His research interests are in the application of machine learning techniques to digital forensics and cybersecurity-related issues, such as mobile malware detection and IoT botnet detection.

Hayretdin Bahsi is a research professor at the Center for Digital Forensics and Cyber Security at Tallinn University of Technology, Estonia. He has two decades of professional and academic experience in cybersecurity. He received his Ph.D. from Sabanc University (Turkey) in 2010. He was involved in many R&D and consultancy projects about cybersecurity as a researcher, consultant, trainer, project manager, and program coordinator at the National Cyber Security Research Institute of Turkey between 2000 and 2014. His research interests include machine learning and its application to cyber security and digital forensic problems.