Computers & Security 138 (2024) 103654

Contents lists available at ScienceDirect

Computers & Security

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

Machine Learning for Android Malware Detection: Mission Accomplished? A Comprehensive Review of Open Challenges and Future Perspectives

Alejandro Guerra-Manzanares

Centre for Digital Forensics and Cyber Security, Department of Software Science, Tallinn University of Technology, Tallinn, Estonia

A B S T R A C T

The extensive research in machine learning based Android malware detection showcases high-performance metrics through a wide range of proposed solutions. Consequently, this fosters the (mis)conception of being a *solved problem*, diminishing its appeal for further research. However, after surveying and scrutinizing the related literature, this deceptive deduction is debunked. In this paper, we identify five significant unresolved challenges neglected by the specialized research that prevent the qualification of Android malware detection as a *solved problem*. From methodological flaws to invalid postulates and data set limitations, these challenges, which are thoroughly described throughout the paper, hamper effective, long-term machine learning based Android malware detection. This comprehensive review of the state of the art highlights and motivates future research directions in the Android malware detection domain that may bring the problem closer to being *solved*.

1. Introduction

The Android operating system (OS) is the leading platform for mobile devices since 2012. At the present time, over 70% of mobile handsets use this open-source and customizable OS (Laricchia, 2022). Despite the significant security enhancements introduced by Google and original equipment manufacturers (OEMs), Android devices are continuously targeted and successfully infected by malware campaigns (Dassanayake, 2021), accounting for over 98% of cyberattacks targeting mobile devices (Kaspersky, 2020). These attacks are performed using a variety of attack vectors exploiting the dynamic attack surface exposed by mobile devices (Townsend, 2020). Even though Android malware figures are significantly smaller compared to *Windows* malware (AV-Test, 2022; R. C., 2022), the continued evolution of the threat landscape (i.e., increasing sophistication (Gurubaran, 2022)) and consistency over time puts Android end users at a permanent high risk of malware infection (Spadafora, 2022). Given that traditional antivirus measures (e.g., *fingerprinting* and *blacklisting*) have proved to be ineffective and limited in protecting end users in the mobile domain (Timothy, 2022), particularly against encrypted and zero-day malware, Android malware researchers have turned their attention to machine learning algorithms in the search for more effective malware detection solutions. In this regard, a vast body of research has been produced on this subject matter during the last decade (e.g., 7,980 results were retrieved on *Google Scholar* searching for *Android Malware Detection at the time of writing* through whole documents, while 2,557 articles were retrieved using the same terms on *title-abstract-keywords* document sections in *Scopus*), and is still an active field of research (e.g., 336 of the *Scopus* articles were published in 2022).

The vast majority of ML-based Android malware detection studies report high-performance metrics (i.e., over 90% accuracy and F_1 score values) on testing data sets using a myriad of increasingly complex algorithms (Muzaffar et al., 2022), which enables the logical deduction that the proposed ML-based solutions are actually effective to detect *future* and *unseen* malware samples. Therefore, based on the abundance of great detection solutions, the general conception is that the Android malware detection problem can be deemed as a *problem solved*, thus attracting marginal attention by the leading cyber security and digital forensics conferences and reputable journals and, consequently, the interest and effort of most cyber security researchers shifts to other less explored and emerging problems (e.g., Internet of Things security).

However, the analyses performed in this paper evidence that this deductive reasoning is a *fallacy*; the Android malware detection problem is far from being able to be tagged as *solved*, and there are critical factors and methodological nuances in building effective production solutions that have either not been addressed or only superficially considered in the related literature. This study does not intend to provide a systematic literature review about Android malware detection (relevant references are provided in Section 2 on that matter) but to analyze the existing literature from a relational and more qualitative perspective, scrutinizing it to identify unsolved challenges and promote future research directions. Therefore, we aim to identify research gaps and encourage

E-mail address: alejandro.guerra@taltech.ee.

https://doi.org/10.1016/j.cose.2023.103654
Received 20 February 2023; Received in revised form 15 November 2023; Accepted 11 December 2023
Available online 14 December 2023

research addressing existing critical challenges for Android malware detection, which may ultimately lead to the generation of actual long-term effective Android malware detection solutions.

The primary research question behind this study was: *Can Android malware detection be regarded as a problem solved?* (RQ1). For our purpose, we would categorize a problem as solved if there are no significant challenges left unaddressed by the specialized research literature in the problem domain (i.e., state of the art). After thorough inspection and evaluation of the literature, in case of a negative answer to *RQ1*, we also considered: *What are the unresolved challenges hampering effective, long-term Android malware detection?* (RQ2). To this end, our objective was to provide a comprehensive review focusing on research gaps and open challenges in the specialized literature concerning Android malware detection.

We scrutinized the machine learning-based Android malware detection literature, filtering and analyzing relevant and representative studies in the problem domain, including historical and state-of-the-art solutions. It is worth mentioning that due to the large number of studies in the field and for the sake of brevity, to illustrate some aspects only the most representative studies are referenced in this article, which still includes over 230 papers. Our extensive review and analysis of the related research literature identify and summarize research gaps and current challenges in ML-based Android malware detection, which is the main contribution of this paper. Briefly, the main challenges faced by current research work are: (i) scarce, outdated and low quality data sets, (ii) wrongful assumption of data consistency across platforms, (iii) concept drift neglection, (iv) shallow exploration of model security aspects, and (v) excessive focus on model performance disregarding model understanding and explainability. We hypothesize that addressing these existing challenges may significantly enhance Android malware detection production systems, making them capable of adapting to an ever-evolving threat landscape, yielding effective long-term detection performance, and bringing the problem closer to being actually *solved de facto*.

The remainder of this paper is structured as follows. Section 2 references related works in the literature. Sections 3–7 identify and describe currently unsolved challenges hampering effective, long-term Android malware detection. Section 8 provides recommendations for future work, while, finally, Section 9 summarizes the paper.

2. Related work: profiling Android malware detection research

Literature reviews, whether systematic or not, survey the existing literature in a problem domain, providing a good synthesis of the state of the art at a particular point in time. Recent literature review papers in Android malware detection provide a good overview of past works and relevant aspects characterizing the research effort (i.e., refer to Dave and Rathod (2022); Kouliaridis and Kambourakis (2021); Liu et al. (2020, 2021); Meijin et al. (2022); Molina-Coronado et al. (2023); Muzaffar et al. (2022); Razgallah et al. (2021); Sharma and Rattan (2021) for relevant recent works on the subject). However, they tend to lack on thorough critical analysis of the relational and more qualitative aspects of existing research work as well as the identification of future research directions and challenges that require the attention of the research community. This is the main contribution of this work, which may be seen as complementary to the relevant literature reviews performed by the Android malware research community. While most reviews focus on aggregating, summarizing and describing main trends on ML-based Android malware detection, only Molina-Coronado et al. (2023) develops a critical discourse supported by thorough experimentation. More specifically, Molina-Coronado et al. (2023) disclose five factors behind the over-optimistic results reported by static-feature based Android malware detection studies. While some of their findings align with this work (dataset quality and concept drift), they focus exclusively on static malware classifiers and datasets, and, therefore, they do not cover issues related to dynamic and hybrid malware detection approaches (e.g., dynamic data consistency) and disregard relevant machine learning aspects like model security (i.e., adversarial attacks) and model interpretability, which are deeply covered in this work. An expanded scope, covering static, dynamic and hybrid classification models and datasets, the inclusion of dynamic data collection challenges, and analysis of additional machine learning modeling aspects such as model security and interpretability are all novel points provided by this work.

Scrutinizing the specialized literature on the problem domain, it can be observed that, despite the algorithmic and other methodological differences, most Android studies share similar characteristics and assumptions, that is, a common *profile* and viewpoint of the problem, a fact that is leveraged in this study to identify and describe current challenges hampering long-term effective Android malware detection. These challenges are primarily founded on shared perspectives and approaches to the task, leaving relevant aspects unexplored. They are enumerated and thoroughly described in Sections 3–7, aiming to bring awareness to the research community of these missing aspects and provide relevant research directions for future research efforts on the problem domain.

3. Challenge I: about data sets, labels, and features

The performance of ML-based detection systems is influenced by numerous variables, such as data set, features, labeling accuracy, algorithm selection, hyper-parameters, etc. Despite the large number of significant variables, they can be abstracted into two major categories: data-related and model-related aspects. Even though model selection and modeling assumptions are important aspects (e.g., linear classifiers fail to model nonlinear data sets), data quantity and quality are critical factors for accurate and generalizable data modeling using machine learning algorithms. In fact, most of the current challenges faced by Android malware detection systems are data-related aspects.

At first sight, given the high-performance metrics reported by the ML-based solutions proposed in the Android malware detection domain, the data problem may seem relatively irrelevant and more related to modeling aspects. However, a close inspection of the data sets used in the studies provides relevant insights into significant data issues.

The most utilized data sets for Android malware research are summarized in Table 1. Note that this list is not exhaustive of all available/used data sets; it focuses on the most used publicly available data sets for research purposes. A thorough analysis of Table 1 data and related literature provides relevant insights about biases and significant data issues in Android malware detection studies, which are described as follows.

3.1. Utilization of imbalanced, small and old data sets

The number of citations of the paper introducing the data set (i.e., *# cit.* column) provides a general notion of the usage of the data set for research. In this regard, as can be observed in Table 1, *MalGenome* and *Drebin* are, by far, the most widely used data sets (at the time of writing). More specifically, 82.8% of the total number of citations (i.e., 4975/6005) correspond to either of the data sets. An 8.2% decrease from the 91% reported in Guerra-Manzanares et al. (2021) on 2021 data. Despite this notable reduction, it manifests that the vast majority of research studies have utilized data collected between 2010 and 2012, representing the threat landscape as it was over ten years ago. Considering the dynamic evolution of the threat landscape in Android data, this makes these data sets too old and obsolete, and thus not representative of the current (or even recent) threat landscape as they were collected when malware capabilities and behavior were significantly distinct from the present ones. For instance, *FakeDefender*, the first Android ransomware, which was discovered in 2013 (Savage et al., 2015), is a malware type that is not included in these widely used data sets. Moreover, most data sets are relatively small, especially if only malware is considered. In this regard, seven of the ten data sets reported provide less than 20,000 samples. Most significantly, *MalGenome*

Table 1
Most used Android malware data sets.

Name (data set repository)	Size		Time frame	Malware families	Feature set	Timestamps	Year	Publication	# cit.
	Malware	*Benign*							
MalGenome[1] (Zhou and Jiang, 2015)	1,260	0	2010–2011	49	n/a[2]	✗	2013	Zhou and Jiang (2012)	2,757
Drebin (T. U. Braunschweig, 2020)	5,560	123,453	2010–2012	179	static	✗	2014	Arp et al. (2014)	2,218
AndroidBot[3] (U. of New Brunswick, 2020a)	1,929	0	2010–2014	14	n/a[2]	✗	2015	Kadir et al. (2015)	94
Kharon (Kiss et al., 2021)	19	0	2011–2015	19	static	✗	2016	Kiss et al. (2016)	46
AMD[1] (ArgusLab, 2020)	24,553	0	2010–2016	71	n/a[2]	✗	2017	Wei et al. (2017)	412
AAGM2017[3] (U. of New Brunswick, 2020b)	400	1500	2015–2016	10	dynamic	✗	2017	Lashkari et al. (2017)	97
AndMal2017[3] (U. of New Brunswick, 2020c)	426	5,065	2015–2017	42	hybrid	✗	2018	Lashkari et al. (2020)	166
InvesAndMal2019[3] (U. of New Brunswick, 2020d)	426	5,065	2015–2017	42	hybrid	✗	2019	Taheri et al. (2019)	108
AndMal2020[3] (U. of New Brunswick, 2020e)	200,000	200,000	-	191	static	✗	2020	Rahali et al. (2020)	22
MalDroid2020[3] (U. of New Brunswick, 2020f)	11,598	0	2017–2018	33	hybrid	✗	2020	Mahdavifar et al. (2020)	62
KronoDroid (Guerra-Manzanares, 2022)	41,382	36,755	2008–2020	240	hybrid	✓	2021	Guerra-Manzanares et al. (2021)	23

[1] Discontinued projects. Data may be unavailable.
[2] The data sets only provide APKs and not processed data features.
[3] Usually referenced with the prefix *CIC*, acronym of the *Canadian Institute for Cybersecurity*.

and *Drebin*, the most used data sets (Alswaina and Elleithy, 2020) are small, only providing 1,260 and 5,560 malware samples, respectively. Furthermore, the larger size of *Drebin* is challenged by the presence of duplicates (Irolla and Dey, 2018), which limits the available data and that, if not addressed, may introduce *data snooping* bias, a remarkable issue for model validation (i.e., *data leakage*). Despite these significant issues, *Drebin* has recently been used in research as the main or single source of malware (Reddy et al., 2021; Syrris and Geneiatakis, 2021; Zhao et al., 2021b).

Half of the data sets in Table 1 do not provide legitimate samples. Therefore, to build supervised detection models, additional data samples must be collected, which will likely belong to different time frames than the malware data. For instance, a typical study may use *Drebin* and collect legitimate samples from *Google Play* or *AndroZoo* (Allix et al., 2016), which can introduce significant temporal bias between data classes and yield over-inflated and not representative results (Arp et al., 2022) (more in Section 5). In the data sets providing both classes (i.e., legitimate and malware samples), the data are usually imbalanced in remarkable proportions (e.g., *Drebin* has a class ratio of *1:22*, while *AndMal2017* a class ratio of *1:12*). Many ML algorithms are sensitive to imbalanced data, generating biased models that can produce misleading results if inappropriate metrics are reported (Arp et al., 2022) (e.g., *accuracy* is the preferred comprehensive metric reported by most studies, but it is not reliable for imbalanced data scenarios). To avoid the generation of biased models due to class imbalance issues, additional data must be included (i.e., some studies combine data sets or add additional samples from malware repositories such as VirusShare (VirusShare, 2022), or AndroZoo (Allix et al., 2016)), or data balancing techniques utilized in the model generation pipeline, which adds an additional layer of complexity to the modeling process (Arp et al., 2022). Lastly, except for *KronoDroid* and *AMD*, the data sets do not span more than four years of Android historical data, thus representing a static snapshot of the threat landscape at a specific point in time. *KronoDroid* provides data for the whole Android historical time frame until 2020, whereas *AMD* includes data between 2010 and 2016.

Therefore, most of the proposed methods in the literature – which are mainly based on a single malware data source – are optimized for malware detection at specific snapshots of the Android history. The threat landscape is then modeled at a specific period and aims to generalize to posterior data frames, which may include significant changes on threat types and data evolution. In this regard, most of the proposed detection solutions neither consider this scenario nor provide model updating mechanisms. This means that static models, built using data collected at a specific point in time and never updated, are assumed to generalize well on future data, thus keeping their detection capabilities over time. This fact neglects the non-stationary character of malware data (i.e., threat landscape evolution) and makes the model prone to *concept drift* and its degenerative impact on the model's performance (more in Section 5).

3.2. The power of hybrid feature sets is neglected

Android malware detection systems use *static* (e.g., permissions, intent filters, API calls) or *dynamic* features (e.g., system calls, network traffic) extracted from Android applications (Liu et al., 2020).

Static features of Android applications are collected directly from the source code, without executing the app (i.e., static malware analysis), typically from two data sources: the disassembled source code (i.e., *classes.dex*) and the *Android manifest* (i.e., *AndroidManifest.xml*). While some works combine features engineered from both data sources (Arp et al., 2014; Felt et al., 2011; Kabakus, 2022; Li et al., 2018, 2019a; Peiravian and Zhu, 2013; Taheri et al., 2020a; Wang et al., 2019; Yerima et al., 2015), most use single-source features. In this regard, the program flow or *API calls* are the most commonly used from the disassembled source code (Cai et al., 2021; Frenklach et al., 2021; Grace et al., 2012; Hou et al., 2017; Ou and Xu, 2022; Yang et al., 2021; Zhu et al., 2017), whereas *security permissions* (Enck et al., 2009; Guerra-Manzanares et al., 2022a; Liang and Du, 2014; Mcdonald et al., 2021; Peng et al., 2012; Şahin et al., 2021; Talha et al., 2015) and *intent filters* (Feizollah et al., 2017; Idrees and Rajarajan, 2014) are the preferred features from the *AndroidManifest.xml*.

API calls, which can also be collected at run-time, can be used to recreate the program flow and collect the functionalities requested by the app from specific *Application Programming Interfaces* (APIs) of code libraries (e.g., Android Platform APIs (Android, 2022a)). Android *security permissions* define the privileges the app has on the system, that is, the actions it can perform and the data it can access (e.g., sensitive data) (Android, 2022b). Android *intents* and *intent filters* are messaging objects that enable apps to request and receive actions from another app component, thus denoting the actions that the app is intended to perform on the platform (Android, 2022c).

Static features are easy to acquire, provide extensive code coverage, and can be used for on-device detection systems. However, the detection systems built using only these features are prone to be deceived if code obfuscation techniques are implemented. In this regard, encryption, update attacks, code obfuscation, and polymorphic techniques are leveraged to hide malicious code and avoid detection (Alzaylaee et al., 2020).

Dynamic features are collected during the execution of the app in a *live* environment, at run-time, tracing the interaction between apps and the operating system or network. *System calls* (Burguera et al., 2011; Guerra-Manzanares et al., 2019a,b; Hou et al., 2016; Tam et al., 2015) and network data flow (Arora et al., 2014; Lashkari et al., 2017) are the typical feature sets utilized to build dynamic features-based detection systems (Liu et al., 2020; Muzaffar et al., 2022). *System calls* are the mechanism utilized by the Android framework for app-OS communication, enabling the tracing of the app behavior at run-time, whereas the network data flow, which is obtained from the app-network interaction, provides the network profile. Other run-time attributes such as CPU and RAM utilization, running processes, battery statistics and run-time API calls have also been used alone (Amos et al., 2013; Enck et al., 2014; Schmidt, 2011) or jointly with system calls or network packets (Dini et al., 2012; Shabtai et al., 2012).

The collection of dynamic features (i.e., dynamic analysis) is time-consuming and technically challenging, requiring the app to be installed and executed in a *sandbox* device (i.e., isolated and controlled environment). Although dynamic feature-based detection systems can be bypassed (Petsas et al., 2014), and the security constraints of the Android framework impede on-device detection, these systems are typically robust against code obfuscation and encryption techniques. Besides, an additional challenge of dynamic data collection is user interaction. While some work opted for not including user interaction in their scenario (Guerra-Manzanares et al., 2019a,c; Vidal et al., 2017), others used real (Burguera et al., 2011) or emulated user interaction (Dimjašević et al., 2016; Hou et al., 2016; Tam et al., 2015) using developer tools like *Monkey* (Android, 2023). While *Monkey* enables the simulation of user behavior via pseudo-random stream event injection, increasing code coverage, it will likely not showcase a realistic user behavior or traverse all instruction paths. For this latter purpose, several tools have been proposed in related work (Hou et al., 2016; Tam et al., 2015). Only Guerra-Manzanares and Välbe (2022) experimented with both scenarios, showing that user interaction may improve detection performance. However, further research is needed to evaluate the impact of user interaction in the collected features and its impact on detection quality.

Although most detection systems are built using only feature sets of one type (e.g., system calls or permissions), the joint utilization of static and dynamic features, the so-called hybrid feature sets, has been explored by a small proportion of the specialized research (Alzaylaee et al., 2020; Bläsing et al., 2010; Grace et al., 2012; Guerra-Manzanares and Bahsi, 2022a; Guerra-Manzanares et al., 2019c; Kabakus and Dogru, 2018; Ullah et al., 2022; Yuan et al., 2014). The hybrid feature sets produce enriched and more complete information about malware samples, complementing run-time behavior with relevant static data. Even though they are more complex and time-consuming to collect and process, they tend to enable richer modeling of the problem, yielding better detection results than single-type approaches (Alzaylaee et al., 2020; Guerra-Manzanares and Bahsi, 2022a; Guerra-Manzanares et al., 2019c).

As can be observed, due to methodological or pragmatic reasons, most studies focus on single approaches, using either static or dynamic feature sets, thus neglecting the potential of the combination of features. This may have also been promoted by the fact that early data sets only provided static data features, as observed in Table 1. However, more recent data sets also include dynamic features, enabling research on the potential benefits of using hybrid feature sets, which can enhance detection performance by using complementary data attributes from different perspectives (i.e., similar to the current trend of enhancing classification using *multimodal* deep learning (Lin et al., 2022)).

3.3. The labeling issue: high cost and uncertainty

The vast majority of proposed solutions for Android malware detection are based on *supervised learning* (i.e., binary or multi-class classification). A supervised learning algorithm utilizes a collection of labeled training examples $\{(x_1, y_1), ..., (x_N, y_N)\}$, where each training example is defined by the pair (x_i, y_i), such that x_i is the feature vector of the *i-th* example and y_i is its label (i.e., class), to seek a function $f : X \rightarrow Y$, where f maps from the input space X to the output space Y. The training data set is used in combination with specific functions f from some space of possible functions F (i.e., machine learning algorithms) to fit the input/output data, finding specific patterns in the data that enable the induction of a *classification model*. All machine learning algorithms make implicit or explicit assumptions about the patterns in the data, which means that each algorithm can learn a specific family of models. Each *model* is called a *hypothesis*, while the set of all the hypotheses an algorithm can learn is regarded as the *hypothesis space*.

Regardless of the algorithm selected to induce the detection model for the classification task, each data sample within the data set must be labeled appropriately. In the case of binary classification models (i.e., the vast majority of Android detection systems), each sample has to be labeled either as *malware* or *benign* sample. In multi-class classification settings, samples are labeled as belonging to a single class from a possible set of k classes, where $k > 2$ (e.g., malware family classification (Alswaina and Elleithy, 2020)).

While data labeling in some machine learning application domains is relatively straightforward and can be performed quickly without particular skills or high technical knowledge (e.g., image tagging for object recognition tasks), in the cyber security domain, the labeling of data samples is technically challenging and time-consuming, requiring high technical skills and particular domain expert knowledge (e.g., reverse engineering, programming and OS system knowledge for malware analysis). This fact limits the labeling speed and the size of the available data sets notably. It also significantly increases the labeling cost, which is directly related to the time invested and expertise required for labeling. For example, *ImageNet*, a common benchmark data set for computer vision algorithms, is composed of 14,197,122 images organized into 21,841 classes, whereas, except for *AndMal2020*, *Drebin* and *KronoDroid*, the data sets in Table 1 have less than 25,000 samples. This data set size limitation can restrict the learning of ML algorithms that require a large amount of data, like deep learning techniques, making them prone to *overfitting*. Overfitting affects the generalization of the trained model (Brownlee, 2018), resulting in reduced performance on unseen data. To address this issue, studies combine several data sets or add additional samples from malware repositories such as *AndroZoo* (Allix et al., 2016; U. du Luxembourg, 2022), a growing repository that contains more than 21 million samples (most of them unlabeled), *VirusTotal Academic* (VirusTotal, 2022), *VirusShare* (VirusShare, 2022), *Contagio Mobile* (Parkour, 2019), and *Koodous* (Koodous, 2022).

In addition to the inherent challenges of Android data labeling, the absence of common terminology to identify specific malware – as there is no naming convention for malware family denominations – adds more complexity and uncertainty to the already arduous task. Malware family attribution is an important task as it enables the identification of malware samples into well-known categories, enhancing malware identification, characterization, and detection (Guerra-Manzanares et al., 2021). Despite its importance, antivirus vendors and malware analysts have different interpretations and names for malware families (Hurier et al., 2016), which undermines the trust in malware family labels, generating uncertainty. A recently discovered piece of malware can be categorized as belonging to an existing malware family, a *variant* of it, a *descendant* (Cohen and Walkowski, 2019) or a *new* malware family. The lack of naming conventions is evidenced in *VirusTotal* analysis reports. For example, the sample with hash *883e6dc7cfcf47c646bb580d7684db4c8dffff9ead153023e8556cd66e5bd7d7* (SHA-256) is flagged as malicious by 26 out of 64 security vendors. Of the positive detections, the particular instance is categorized as belonging to the *GoldDream* family by some AVs, to *Youmi.A* or *Youmi.B* by others, while it belongs to *KungFu* or *KyView* for some others. It is a generic trojan (i.e., Trojan/Android.Generic), a PUA, a riskware, or a generic *adware* for others, while it is also designated as a piece of *generic* malware with different malicious scores (i.e., 85 and 97), and Android malware categorized using vendor-specific cryptic denominations (e.g., *Artemis!8AA114A1F2E7* or *ApplicUnwnt@#2smy9aq1px39u*). Consequently, a single piece of malware is likely to receive as many different names as the number of scanners detecting the sample as malicious, even for well-known families such as *GoldDream*. As a consequence (or cause) of the lack of naming conventions, AV vendors use their own denominations (Kaspersky, 2021; Microsoft, 2021), which may be very difficult to interpret, posing a significant challenge to actual malware family identification and categorization even for seasoned malware analysts (Hahn, 2019a). In this regard, *Euphony* is a label unifier solution that can be used to parse malware labels from *VirusTotal* scanning reports and attribute a single family per sample (Fmind, 2019), whereas Hurier et al. (2016) proposed a set of metrics that enable the assessment of AV label consensus, which can be utilized to improve *ground-truth* labels.

Correct malware family identification is critical for proper malware elimination, cleaning, and restoration after an infection (Hahn, 2019b). In research, malware family attribution enables the characterization and understanding of different malware families, their development, and evolution, enabling the generation of not only better countermeasures but also more effective binary and multi-class detection systems. The absence of naming conventions and utilization of confusing malware family denominations are major obstacles for them.

Another major obstacle to building effective detection systems is the constant change and evolution of the threat landscape, which requires constant labeling efforts and updated knowledge to detect new and evolved malware variants that can remain undetected for long periods of time. This issue is a very particular and challenging characteristic of the cyber security domain as the adversary generating the piece of malware is a human that intelligently designs novel attacks and enhances his/her creations to bypass defensive responses. This increases the complexity of designing and maintaining effective detection systems over time as the threats are constantly changing based on intelligent agents. Thus, this requires a constant labeling effort from humans to address and update the knowledge of classifiers if the *new* threat is undetectable on the basis of historical knowledge, which means a change in the distribution of the input data that is not reflected in the model's training set (see more in Section 5). *Active learning*, a form of semi-supervised learning, the application of which has been overlooked in the cyber security domain, can assist in the generation of more efficient data labeling pipelines and model updates, significantly reducing the labeling cost by concentrating the available human resources to label the most relevant instances to update the model instead of randomly selecting samples and without needing to label all the incoming data (Guerra-Manzanares and Bahsi, 2022a).

Finally, even though active learning can help reduce the labeling cost and optimize labeling efforts, labeling instability or certainty is a major challenge in the cyber security domain in general and for mobile malware detection in particular. Not only the different names for the same pieces of malware may create confusion for multi-class classifiers, but more importantly, labeling consistency is a major contributing factor to model degradation.

Almost the totality of Android malware detection studies use *VirusTotal* scanning service to check and assign the class label of a data instance (i.e., *benign* or *malware*). The app is submitted to the service and a detection report is returned. The class of a sample is typically decided using thresholds (i.e., a minimum number of AV scanners have to detect the sample as malware to categorize it as malware) or selecting high-reputation engines whose detection results are considered more trustworthy than others (Hurier et al., 2016; Zhu et al., 2020a). Regardless, the label of the sample is usually *fixed* and assumed to be certain. However, Zhu et al. (2020a,b) studied the labeling dynamics of AV vendors and proved that malware labeling changes over time, especially for *new* malware instances. This implies that label certainty cannot be assumed, at least initially, and that label flips are frequent (i.e., from benign to malware class and vice-versa). Rescanning apps frequently can help but is not a guarantee of label certainty as *VirusTotal* may change the scanners that process the samples over time (Salem et al., 2021). This introduces a new challenge for Android malware detection systems, which can result in involuntary *poisoning* of the classifier (see Section 6 for more about *training set poisoning*), generating less precise or wrong decision boundaries and degrading performance. To overcome the *VirusTotal* label dynamics issue, Salem et al. (2021) proposed a method relying on requesting the label from *correct* scanners at different periods and using Random Forest algorithm to accurately and consistently label data instances.

Despite the weaknesses of *VirusTotal* label dynamics, which can provide faulty labels, we argue that relying on a single analyst or expert to label the samples could provide even worse results. In this regard, in security operations centers or antivirus companies, factors such as the level of expertise of the analyst, experience, or time constraints can affect the labeling outcome significantly, and relying on a single expert judgment can exacerbate the labeling error, which may lead to wrong decision boundaries resulting in easy-to-bypass faulty detection models. Regardless, most Android studies use single-check labeling (i.e., no recheck is performed after the initial label attribution) (Zhu et al., 2020a) or rely on the labels provided by the data set authors at the time of data set creation. This can yield misleading results about the effectiveness of the proposed method and does not reflect the real challenges faced in production setups. The usage of old data sets may provide more certainty about the label but generates detection models based on old and non-representative data concerning the current threat landscape, which may provide misleading and over-inflated results on similar testing data but ineffective protection against current threats (Arp et al., 2022).

Data generation, processing, and curation are critical components to building effective Android malware detection systems that have been overlooked by the specialized research. Domain application weaknesses such as the utilization of old, non-representative data sets, the predominance of single-source features (i.e., static features), neglect of data imbalance issues, and labeling dynamics, make the effective implementation of most of the proposed solutions in the related literature in production setups impracticable. The aforementioned challenges must be considered and addressed by any detection solution aiming for real-world implementation.

4. Challenge II: the postulate of dynamic data consistency

Static features such as metadata, security permissions, and disassembled source code API calls cannot be modified without altering the hash value of the sample. Due to their origin, static features do not depend on the acquisition platform. Despite their significant differences, a similar assumption of data consistency across platforms has typically been applied to behavioral data (i.e., dynamic features), which require the execution of the samples in a *live* environment to be acquired. This fundamental assumption has been leveraged in the research context to utilize a myriad of devices and Android OS versions for data collection. As can be observed in Table 2, both real devices and emulators have been widely used as data collection platforms but seldomly together.

The data provided in Table 2 provides relevant and recent papers concerning Android malware detection using dynamic features. In par-

Table 2
Relevant and recent studies using dynamic feature sets.

Device type	Features	Publication
Real device	System calls	Xiao et al. (2019), Amin et al. (2016), Wahanggara and Prayudi (2015), Xiao et al. (2015), Ahsan-Ul-Haque et al. (2018), Canfora et al. (2015), Da et al. (2016), Isohara et al. (2011), Xiao et al. (2016), **Vidal et al. (2017)**[1], **Wei et al. (2022)**[1], **Burguera et al. (2011)**[1], **Yu et al. (2013)**[1]
	Other	Saracino et al. (2018), **Alzaylaee et al. (2020)**[1], **Wang and Li (2021)**[1], **Shabtai et al. (2012)**[1]
Emulator	System calls	Dimjašević et al. (2016), Guerra-Manzanares et al. (2019c), Hou et al. (2016), Lin et al. (2013), Malik and Khatter (2016), Abderrahmane et al. (2019), Bhatia and Kaushal (2017), Kapratwar et al. (2017), Casolare et al. (2021), Jaiswal et al. (2018), Leeds et al. (2017), Singh and Hofmann (2017), Surendran et al. (2020a), Zhang et al. (2021b), Tong and Yan (2017), Ananya et al. (2020), Vinod et al. (2019), Surendran et al. (2020b), Jerbi et al. (2020), **Naval et al. (2015)**[2], **Sihag et al. (2021)**[2], **Tchakounté and Dayang (2013)**[2]
	Other	Afonso et al. (2015), Ferrante et al. (2016), **Feng et al. (2018)**[2], **Jang et al. (2014)**[2], **Lindorfer et al. (2015)**[2], **Saif et al. (2018)**[2], **Yuan et al. (2014)**[2], **Han et al. (2020)**[2]
Real & emulator	System calls	Guerra-Manzanares et al. (2019b), Guerra-Manzanares et al. (2019a), **Guerra-Manzanares and Välbe (2022)**[1]
	Other	Alzaylaee et al. (2017)

[1] **Bold** indicates the utilization of more than one real device.
[2] **Bold** indicates the utilization of a specialized sandbox.

ticular, it specifies the device type utilized in the referred publications (i.e., real device, emulator, or both) and the dynamic feature set employed. In this latter regard, *system calls* refers to publications that used only this feature set as the dynamic data source (i.e., regardless of using static attributes), whereas *other* refers to papers that used more than one dynamic feature set (i.e., either including or not system calls as features). As can be observed in Table 2, *standard* Android emulators (e.g., *Android emulator* or *GenyMotion*) are the most frequently used type of devices in research as data collection *sandbox*. These devices are mostly used to acquire system call traces, which are the most commonly used dynamic data in Android malware detection studies (Liu et al., 2020; Muzaffar et al., 2022).

Android emulators are *virtual* Android devices running on a *host* machine that allows mimicking almost all the capabilities of a wide range of real devices and Android versions without owning any physical device (Android, 2021). They are easy to deploy, manage, and integrate into automated analysis and detection systems (Dimjašević et al., 2016). Despite their versatility, malware with sandbox detection capabilities can deceive them by not triggering the malicious behavior and avoid positive detection and forensics analysis (Lindorfer et al., 2015). Even though some specialized sandboxes have been created to address this issue (Naval et al., 2015; Vinod et al., 2019), they still have limitations, such as lacking the sufficient interactive capabilities needed by some pieces of malware to trigger malicious activities (e.g., SMS message or SIM card detection (Feng et al., 2018)). Moreover, despite that they usually provide *root* accounts, their forensic capabilities are limited to x86 or x86-64 architecture-compatible applications, which requires the application to include additional specific libraries given that real devices are based on ARM architectures.

Real devices are physical handsets. They are compatible with most applications, showing much fewer compatibility issues, provide full interaction, and are naturally immune to anti-sandbox techniques. However, they are more difficult to manage and integrate into automated solutions. *Rooting* the device is required to perform many forensic activities, which wipes the device data and can *brick* the device, making it unusable. Furthermore, ensuring the same conditions for all experiments can be challenging, and cleaning the devices after each data collection is time-consuming (Lin et al., 2013).

Regardless of the rationale behind the selection of the data collection platform in each study, the main assumption across all studies, which has even been explicitly stated (Burguera et al., 2011; Lin et al., 2013; Vidal et al., 2017), is that the behavior of applications is consistent across Android devices and OS versions. This means that the results obtained using a specific combination of device and OS version are generalized to all other kinds of devices and OS versions. This fact explains the myriad of data collection platforms and OS versions used for research purposes and the lack of homogenization and device selection criteria. The behavior is assumed to be *fully consistent* regardless of the execution platform. While this is true regarding static data, the same cannot be implied for dynamic data as evidenced by the results of the small proportion of works that utilized both kinds of devices in their experimental setup (see last rows in Table 2). More precisely, all the studies that used more than a single type of device (i.e., emulators and real devices) found notable inconsistencies in the logged behavior of the same set of Android apps across devices, both for system calls (Guerra-Manzanares and Välbe, 2022; Guerra-Manzanares et al., 2019a,b) and API calls (Alzaylaee et al., 2017). These inconsistencies led to diminished cross-device detection performance (Guerra-Manzanares and Välbe, 2022). Therefore, the results of these works challenge the validity of the consistent behavior across devices postulate, which can have significant implications for production detection systems.

Despite that the reduced number of studies might not be found as conclusive evidence as to falsify the shared assumption across dynamic features-based studies, it provides enough grounds to foster further research on the topic needed to evaluate the significant implications that this fact can have in the design and implementation of effective production detection systems. Inconsistent cross-device behavior hampers the transferability of knowledge across devices, impeding the generation of hybrid solutions that combine emulator and real device data. Furthermore, if the behavior is also inconsistent across the same type of devices (e.g., across real devices or emulators), as suggested in Guerra-Manzanares and Välbe (2022), it prevents the implementation of on-device data collection for collective privacy-preserving knowledge-sharing architectures, such as federated learning-based solutions. The design and implementation of production systems using dynamic data attributes must consider this important factor in the data selection and aggregation pipelines to provide effective detection performance for the end users. The successful application of the detection solutions proposed in research studies into production systems directly depends on assessing these critical factors through further research on the topic.

5. Challenge III: concept drift

Most machine learning-based models are *static*, thus assuming stationary input data distributions, which are consistent over time. More specifically, the training data used to build the model and the testing data used to evaluate the model are assumed to be very *similar* (i.e., coming from the same data distribution). While this might apply to some application domains, most ML problems face non-stationary data distributions, where the statistical properties or defining features

Table 3
Research studies considering concept drift in Android malware detection.

Reference	Time frame	Data set(s)	Feature set	Timestamp	Performance (%)
Narayanan et al. (2016)	2014	Google Play/Anzhi/AppChina/SlideMe/HiApk/Fdroid/Angeeks	ICFGs[1]	Compilation date	Acc: 84
Onwuzurike et al. (2019)	2010–2016	Drebin/Virushare	API calls[2]	*First seen*	F_1: 99–87
Cai et al. (2019)	2009–2017	MalGenome/Drebin/AndroZoo/VirusShare/Google Play	API calls[3]	*First seen*	F_1: 97
Jordaney et al. (2017)	2010–2014	Drebin/MARVIN	Static	-	F_1: 82
Xu et al. (2019)	2011–2016	AndroZoo	API calls[2]	Compilation date	F_1: 95–85
Lei et al. (2019)	2012–2018	PlayDrone/Google Play/VirusShare	API calls[2]	*First seen*	F_1: 99–84
Pendlebury et al. (2019)	2014–2016	AndroZoo	Static	Compilation date	F_1: 91–82
Barbero et al. (2020)	2014–2018	AndroZoo	Static	Compilation date	F_1: 90–70
Zhang et al. (2020b)	2012–2018	VirusShare/VirusTotal/AMD/Google Play/AndroZoo	API calls[2]	*First seen*	F_1: 92–68
Cai (2020)	2010–2017	VirusShare/AndroZoo/Google Play	API calls[3]	Compilation date	F_1: 92–72
Ceschin et al. (2023)	2010–2012 2016–2018	Drebin AndroZoo	Static	*First seen*	F_1: 89–75
Guerra-Manzanares et al. (2022b)	2011–2018	KronoDroid	System calls	*First seen* *Last modification*	F_1: 95
Guerra-Manzanares et al. (2022a)	2011–2018	KronoDroid	Static	*Last modification*	F_1: 93

[1] Inter-procedural control-flow graphs (dynamic features).
[2] Source code API calls (static features).
[3] Run-time API calls (dynamic features).

of the target variable change over time in an unpredictable fashion (Lu et al., 2018), a phenomenon named *concept drift*. Formally, given a set of examples $S_{t_0,t_1} = \{s_{t_0}, ..., s_{t_1}\}$ defined in a bounded period of time $[t_0, t_1]$, where $s_i = (x_i, y_i)$ is a single example, $x_i = (x_i^1, x_i^2, \ldots, x_i^n) \in \mathbf{X}$ is the feature vector, $y_i \in \mathbf{Y}$ refers to the target label, and S_{t_0,t_1} follows a particular data distribution $F_{t_0,t_1}(\mathbf{X}, \mathbf{Y})$ (Guerra-Manzanares and Bahsi, 2022b), the phenomenon of *concept drift* happens at t_2 if $F_{t_0,t_1}(\mathbf{X},\mathbf{Y}) \neq F_{t_2,\infty}(\mathbf{X},\mathbf{Y})$, and is denoted as $\exists t : P_t(\mathbf{X},\mathbf{Y}) \neq P_{t+1}(\mathbf{X},\mathbf{Y})$ (Lu et al., 2018). Based on this definition, *concept drift* at period t_i is related to a change in the joint probability of $\mathbf{X}$ and $\mathbf{Y}$ at time t_i (i.e., $P_{t_i}(\mathbf{X},\mathbf{Y})$). Given that $P_{t_i}(\mathbf{X},\mathbf{Y}) = P_{t_i}(\mathbf{X}) \times P_{t_i}(\mathbf{Y} \mid \mathbf{X})$, *concept drift* can be originated from three sources (Lu et al., 2018): (1) $P_t(\mathbf{X}) \neq P_{t+1}(\mathbf{X})$ and $P_t(\mathbf{Y} \mid \mathbf{X}) = P_{t+1}(\mathbf{Y} \mid \mathbf{X})$ (i.e., named *virtual drift*, where the change in the data distribution does not affect the decision boundary of the model and does not require the adoption of adaptive measures); (2) $P_t(\mathbf{X}) = P_{t+1}(\mathbf{X})$ and $P_t(\mathbf{Y} \mid \mathbf{X}) \neq P_{t+1}(\mathbf{Y} \mid \mathbf{X})$ (i.e., referred to as *real concept drift*, which requires adaptive measures as the change in the posterior probability affects the decision boundary of the model and produces a decrease in the model's performance), and (3) $P_t(\mathbf{X}) \neq P_{t+1}(\mathbf{X})$ and $P_t(\mathbf{Y} \mid \mathbf{X}) \neq P_{t+1}(\mathbf{Y} \mid \mathbf{X})$ (i.e., another case of *real concept drift*, which requires adaptive measures due to the change in the feature data distributions and the decision boundary).

Based on the previous definitions, only real concept drift affects the model's decision boundary, resulting in a decrease in the generalization of the model that leads to model obsolescence over time. Therefore, from the viewpoint of predictive modeling, only the shifts affecting the model's decision boundary, which directly relates to the model's predictions, require the adoption of adaptive measures (Gama et al., 2014) (i.e., model update).

The cyber security domain in general, and Android malware detection in particular, are characterized by the constant evolution of the threat landscape (e.g., malware evolution or the emergence of new families). Therefore, *static* models for Android malware detection are prone to concept drift issues that lead to performance decay and model obsolescence over time if adaptive measures are not taken (i.e., in this context, *static model* refers to a model that is not updated over time). Despite that, the vast majority of Android malware detection solutions proposed in the specialized literature are *static*, neglecting concept drift and do not propose or consider any adaptive measures (e.g., model updating mechanisms or retraining schedules). Consequently, these models are simply ML-based optimizations for specific snapshots of Android historical data, which will fail to deal effectively with *new* and *evolved* data, not represented in the training data set, in the short-term (i.e., worst-case scenario) or long-term (i.e., best-case scenario). Moreover, the usual practice of splitting randomly the data set – typical in machine learning workflows and Android malware detection studies – neglects the existence of concept drift and the temporal order among the data samples. As a result, the historical coherence between the training and testing sets is undermined (Allix et al., 2015; Pendlebury et al., 2019), yielding biased, over-inflated and historically incoherent results caused by *data snooping* (Arp et al., 2022). This is a major validation flaw present in most Android malware detection studies.

Table 3 summarizes the small proportion of studies in the Android malware detection literature that considered concept drift in their design and validation. As can be seen in Table 3, only a few research works dealing with Android malware detection have considered concept drift in their design and validation. These works proposed ML-based detection systems that can *adapt* to data changes over time (i.e., data evolution) and, consequently, reduce or avoid the performance decay caused by concept drift over time. Data drift detection techniques have been proposed (Barbero et al., 2020; Jordaney et al., 2017; Pendlebury et al., 2019) in the related literature, which can be used as indicators of emerging concept drift. As reported in Table 3, most of the proposed detection systems focused on API calls as input features (Cai, 2020; Cai et al., 2019; Lei et al., 2019; Narayanan et al., 2016; Onwuzurike et al., 2019; Xu et al., 2019; Zhang et al., 2020b), a feature set that can be collected both statically and dynamically. Similar to Section 3.2, the superior discriminatory power and robustness against obfuscation and encryption techniques provided by the hybrid feature sets have not been leveraged by these solutions. Most approaches proposed static features-based solutions, which are prone to be deceived and misguided by *adversarial* samples and attacks (see Section 6). Despite that the reported performance of most works is over 90% F_1 score, the studied time frame varies significantly among studies, with the studies using short time frames assuming concept drift but not evidencing its existence. In this regard, only Guerra-Manzanares et al. (2022b) proved the existence of concept drift in the time frame encompassed by the data set to justify the adoption of the proposed methodology.

As can be observed in Table 3, most studies combine data sets and malware repositories to cover the specified time frame. This emphasizes the limitations of the existing data sets for concept drift analysis. The only exception is *KronoDroid*, a data set that was conceived to investigate concept drift and cross-device detection issues (Guerra-Manzanares et al., 2021). *AndroZoo*, a huge repository of *apks* is used by a large number of works to complement existing data sets with malware and legitimate Android apps.

The central concept underlying concept drift handling and analysis are *timestamps*. Timestamps enable the ordering of applications along the historical timeline, which is essential for *historical coherence*. Due to the characteristics of malware generation, it is often not possible to locate instances with certainty along the temporal axis. For that purpose, temporal approximations (i.e., timestamping approaches) are used. In this regard, different timestamping approaches provide different ordering of data samples along the historical timeline (Guerra-Manzanares and Bahsi, 2022b). As can be seen in Table 3, the most frequent timestamps used in research are *first seen* and *compilation date*. These timestamps are retrieved from *VirusTotal* reports and used to order the whole data set chronologically.

Compilation date is an *internal* timestamp that reports the creation or compilation day of the application (i.e., *apk* archive). Despite being referred to as a reliable timestamp by past works (Pendlebury et al., 2019) and used in related research (Barbero et al., 2020; Cai, 2020; Pendlebury et al., 2019; Xu et al., 2019), it is an unusable approach nowadays as recent apps provide an invalid value (i.e., 1980) (Guerra-Manzanares and Bahsi, 2022b; U. du Luxembourg, 2021). On the other hand, the *first seen* timestamp is an *external* timestamp, also referred to as *appearance* or *submission* time in the related research, which reports the day on which the application was first submitted to *VirusTotal*. *External* timestamps can be deemed more robust as they are outside of the scope of the attacker (i.e., they are provided by a reliable third party). However, they are prone to significant delays as evidenced in Guerra-Manzanares and Bahsi (2022b), where *first seen* and the *last modification* timestamp are thoroughly compared. More specifically, in Guerra-Manzanares and Bahsi (2022b), the authors perform a thorough analysis and benchmarking of timestamps for concept drift handling in Android malware detection, showing that *first seen* is always delayed concerning the *last modification* timestamp and that the latter provides better handling of concept drift for old data. However, for recent samples, the delay appears to be insignificant and both timestamps are almost equivalent. This means that both approaches could be used effectively to handle concept drift in production systems.

Despite the importance of timestamps for concept drift handling, none of the available data sets except for *KronoDroid* includes them, as can be seen in Table 1. This is aligned with the neglect of concept drift shown by the specialized research. *KronoDroid* includes six timestamps per sample (i.e., two internal and four external timestamps) that can be used as benchmarks for concept drift-handling solutions, as in Guerra-Manzanares et al. (2022a,b). Despite its limitations, the data set intends to be the seed for further research on the topic and the inspiration for improved data sets (Guerra-Manzanares et al., 2021).

A concept drift-related issue, which has not been explored in the literature, is the detection of new malware families in an automated fashion for multi-class classification purposes (i.e., malware family detection model). While a drift signaling technique can be used to decide when to retrain a binary classification model, a more interesting investigation is the automatic detection of new malware families (i.e., malware threats that deviate significantly from historical, familial data) and their integration into a multi-class classification model. This aspect has not been explored in the related literature, which is also notably scarce concerning multi-class detection solutions (Alswaina and Elleithy, 2020).

Most research work concerning Android malware detection neglect concept drift and its impact on the model's performance over time. Due to the constant evolution of the threat landscape, any solution aiming for effective long-term detection should consider concept drift adaptation in its design and implementation. The small number of studies that considered concept drift in the specialized literature focused on particular timestamps, which is a critical aspect behind concept drift handling. These works provide the seed to foster research on the topic, which can help to enhance production systems significantly. More specifically, more research is needed on effective methodologies to handle concept drift in Android malware research, in data sets facilitating this exploration, and on the proposition of feasible timestamping alternatives and update schedules for production detection systems dealing with concept drift.

6. Challenge IV: model security – adversarial machine learning

As shown in the previous sections of this paper, ML-based Android malware detection solutions may provide high detection performance in the short-term or the long-term if concept drift is addressed. This assumes that the detection system is trained and deployed in a *benign* setting, in which, regardless of the emergence of natural concept drift, the data samples are genuine. However, this cannot always be ensured. There are *adversarial* scenarios in which motivated attackers may intentionally synthesize input data to provoke mistakes in the predictions of well-trained classification models. These motivated attacks on ML classifiers have promoted a substantial research effort on the security and robustness of machine learning (Biggio and Roli, 2018; Huang et al., 2011).

The security of ML-based Android malware detection models has also been explored by the specialized research community, which has produced a significant number of papers on the topic. Even though some general surveys include adversarial Android malware studies (Li et al., 2021a), we perform a comprehensive categorization of the adversarial Android malware detection studies based on the simplified threat model taxonomy of attacks against machine learning described in Biggio and Roli (2018). The categorization of the studies is presented in Table 4 and is explained as follows.

The simplified threat model described in Biggio and Roli (2018) can be depicted using a 2x3 matrix, where the first dimension describes *attacker's capability* to access specific data (i.e., test and training data), whereas the second dimension describes the *attacker's goal* to compromise: integrity, availability or privacy/confidentiality. Attacks on the integrity of machine learning models do not compromise the normal operation of the system, while availability attacks do. Privacy/confidentiality attacks are based on querying strategies that aim to reveal confidential information of the model or its users (Biggio and Roli, 2018). Integrity attacks on test data are named *evasion* attacks or *adversarial examples*, as the main objective of the attacker is to craft samples to avoid detection (i.e., in the case of malware). Integrity attacks on training data are called *poisoning integrity* attacks, which aim to introduce *backdoors* or *trojan* samples for subsequent intrusions. Availability attacks to the training data, also referred to as *poisoning* or *poisoning availability* attacks, aim to maximize the generalization error of the model (i.e., maximize the classification error on the test data to make the detection model ineffective). Finally, the privacy/confidentiality attacks are perpetrated using crafted test data samples to obtain confidential information and can be categorized as *model extraction/stealing* or *model inversion*, depending on whether the target is to steal the model or extract sensitive information about its users, respectively (Biggio and Roli, 2018). Due to the particularities of the Android malware detection problem, the studies focus on the investigation of two attacks on the security of classifiers: *integrity attacks* at the testing phase (i.e., *evasion* attacks aiming to bypass positive detection of perturbed or crafted malware instances), and *availability attacks*, which aim to *poison* the training data and cause a denial of service (Chen et al., 2018). Evasion attacks can be further categorized as *feature-space* attacks, also referred to as theoretical, or *problem-space* attacks, also referred to as physical, depending on the nature of the transformation performed on the input data to deceive the classifier (Pierazzi et al., 2020). While in feature-space attacks the adversary perturbs the feature vector extracted from the sample to deceive the classifier, problem-space attacks craft a new, real and fully functional evasive sample for such a purpose (Zhao et al., 2021a). Therefore, Table 4 classifies the studies in the literature according to the attack they investigate: *evasion*, *poisoning*, or both. For evasion attacks, we also differentiate between feature-space attacks (bold font), problem-space attacks (underlined font), and both (bold and underlined font).

As can be observed in Table 4, the vast majority of adversarial studies in the Android malware detection domain investigates *evasion* techniques and *adversarial sample* generation. In this regard, some studies

Table 4
Classification of studies according to the type of adversarial attack(s) investigated.

Attack target \ Feature set	Static	Dynamic	Hybrid
Integrity (evasion)	**Chen et al. (2017a)**[1], **Liu et al. (2019)**[1], **Chen et al. (2018)**[1], **Rathore et al. (2021b)**[1], **Li et al. (2019a)**[1], **Chen et al. (2017b)**[1], **Li et al. (2021c)**[1], **Li et al. (2019b)**[1], **Yumlembam et al. (2022)**[1], **Li et al. (2021d)**[1], **Rathore et al. (2021c)**[1], **Darwaish et al. (2021)**[1], **Zhang et al. (2020a)**[1], **Shahpasand et al. (2019)**[1], **Ahmed et al. (2021)**[1], **Rathore et al. (2021a)**[1], **Rathore et al. (2022)**[2], **Renjith et al. (2022b)**[2], **Abaid et al. (2017)**[2], **Li et al. (2021b)**[2], **Li and Li (2020)**[2], Chen et al. (2019b)[3], Xu et al. (2018)[3], Yang et al. (2017)[3], Renjith et al. (2022a)[3], Zhao et al. (2021a)[3], Zhang et al. (2021a)[3], Cara et al. (2020)[3]	**Ananya et al. (2020)**[1]	**Ahmed et al. (2022)**[1]
Availability (poisoning)	Taheri et al. (2020b), Taheri et al. (2020c), Chen et al. (2016), Chen et al. (2019a), Taheri et al. (2020d),	Vinod et al. (2019)	-
Integrity & Availability	**Bala et al. (2021)**[1]	-	**Hou et al. (2019)**[1], **Anupama et al. (2022)**[1]

[1] **Bold** font indicates that the study focused on feature-space evasion attacks.
[2] **Bold and underlined** font indicates that the study dealt with feature-space and problem-space evasion attacks.
[3] Underlined font indicates that the study focused on problem-space evasion attacks.

propose malware detection methods and show their robustness against adversarial samples (Chen et al., 2017a; Li et al., 2021c,d; Xu et al., 2018; Yumlembam et al., 2022), while others propose techniques to generate *adversarial samples* leveraging ML methodologies such as Generative Adversarial Networks (GANs) and Reinforcement Learning (RL) (Chen et al., 2019b; Darwaish et al., 2021; Li and Li, 2020; Li et al., 2019a; Liu et al., 2019; Renjith et al., 2022b; Shahpasand et al., 2019; Yang et al., 2017; Zhang et al., 2021a; Zhao et al., 2021a). Feature-space attacks, which remain at theoretical level and are easier to implement, are the most common evasion attacks explored in the related literature, while problem-space attacks, which require greater technical skills to craft the new sample (i.e., disassembly, modification and repackaging of a new functional app) have been significantly less explored.

In addition, almost all studies in Table 4 use static feature sets and the *Drebin* data set in some way. More specifically, while some use the complete *Drebin* data set (Bala et al., 2021; Chen et al., 2019b; Li and Li, 2020; Li et al., 2021b; Rathore et al., 2022; Yang et al., 2017; Zhang et al., 2020a), others use subsets of features/data of this popular data set (Chen et al., 2017a, 2018; Li et al., 2019a; Rathore et al., 2021a,b; Renjith et al., 2022b; Shahpasand et al., 2019; Taheri et al., 2020b), or use the samples but not features (Ananya et al., 2020), and others use the *Drebin* classifier, induced using the data set (Abaid et al., 2017; Chen et al., 2016). This confirms the popularity of the data set for recent works, despite the sample redundancy issue found by previous works (Irolla and Dey, 2018) and the *outdated* malware data it contains. Moreover, most studies use benign samples collected at the time of research, which creates a significant gap between malware and benign samples that can produce unrealistic results. Further research is needed to validate the results with more recent malware data, including more complete data sets and recent attack vectors. The observation made in Section 3.2 is also confirmed by the data in Table 4, which shows the high concentration of research in static feature-based models and the disregard for the usage of dynamic and hybrid feature sets that have been proven to be also more robust and resilient in adversarial settings than classifiers built using static features (Anupama et al., 2022).

Several defensive strategies against adversarial attack have been proposed (Chen et al., 2017b; Li et al., 2021b; Taheri et al., 2020b), including adversarial training (Bai et al., 2021; Wang et al., 2021). Due to its importance, further research is needed on the security of ML-based Android detection models, especially using dynamic and hybrid feature sets, which can act as a defensive strategy by themselves but are not immune to attacks (Anupama et al., 2022).

Lastly, the focus on *evasion* has neglected the importance of training data *poisoning*, which is relevant and feasible in the federated learning context, as evidenced by Taheri et al. (2020d), where an adversary can easily tamper training data by posing as an ordinary node and contaminate the global model. Similarly, the non-stationary nature of the threat landscape has been overlooked by the adversarial studies, which have focused on the assumption of stationary data. Consequently, *adversarial concept drift* has not been explored by the related literature. In this context, the adversary injects fake concept drifts to mislead the detection system towards unnecessary adaptive measures that have the ultimate goal of downgrading the performance of the classification system. *Adversarial concept drift* adds an extra layer of complexity to the detection models as it requires the discrimination between natural and adversarial concept drifts without hindering the adaptation process to natural data evolution (Korycki and Krawczyk, 2022). Further investigation is needed to explore these critical aspects for enhancing the model's robustness against real adversaries in production detection systems.

Adversarial studies in the Android malware detection domain have focused on a specific subset of the whole problem space (i.e., *evasion* attacks on *static* models induced utilizing *static* feature sets). Consequently, most of the problem space remains unexplored or has been superficially investigated. In this regard, relevant future research directions to increase the robustness of production detection systems against real adversaries include the utilization and analysis of *adversarial* attacks against models induced using *dynamic* and *hybrid* feature sets, *poisoning* attacks in federated contexts and *adversarial concept drift*.

7. Challenge V: explainable AI – understanding the model and its predictions

Explainable Artificial Intelligence (XAI) aims to reveal the decision process behind the predictions made by machine learning models, which are mostly regarded as *black box* models (i.e., especially deep learning-based systems). Besides their implementation to meet legal requirements of model's decision transparency in some jurisdictions (Goodman and Flaxman, 2017), the application of interpretation methods to understand the detection model and its predictions can provide relevant information not only about model behavior but also about data set behavior itself (i.e., the threat landscape in Android malware detection). For instance, Guerra-Manzanares et al. (2022b) utilized *feature importance* evolution to understand concept drift issues on system calls data, which was expanded in Guerra-Manzanares et al. (2022c) for device-based analysis, and Guerra-Manzanares et al. (2022a) provided a similar analysis for security permissions and specific malware families.

The research effort for Android malware detection solutions has focused on the optimization of performance metrics, which aligns with the primary objective of the application of AI to cyber security issues, generating increasingly complex architectures adopted from other AI application domains to deal with the Android malware detection problem (e.g., computer vision models (Yadav et al., 2022) or graph transformers (Fan et al., 2021)). The widespread utilization of deep learning architectures in research works, often regarded as the paradigm of a black box model for AI applications, has shifted focus away from more interpretable approaches like Decision Trees and Linear Regression (Molnar, 2022). This shift has heightened interest in model interpretability, promoting the adoption of model-agnostic interpretation methods (such as feature importance (Breiman, 2001) and Shapley values (Shapley, 1953)) and the development of techniques specific to neural networks (e.g., pixel attribution for image classifiers (Simonyan et al., 2013)). More interestingly, *global* and *local* interpretation methods have been developed to explain the average behavior of a model and individual predictions, respectively Molnar (2022). However, despite the recent investigative effort on the field and the variety of solutions available, the application of interpretation methods to the Android malware detection domain is scarce, mostly restricted to *feature importance* analysis (e.g., Guerra-Manzanares et al., 2019a,b,c).

Although some studies in the domain have highlighted the importance of explainability of the predictions to review model outputs to improve detection mechanisms (Kinkead et al., 2021; Scalas et al., 2019), the scope and application of XAI techniques in cybersecurity and for Android malware detection in particular is still limited (Wu et al., 2021). In this regard, Scalas et al. (2019) used explainability methods to find the top discriminatory API calls used by Android apps, while Kinkead et al. (2021) used *LIME* to find the most important *global* features for overall classification and specific malware families. Karn et al. (2021) compared different XAI techniques for cloud-based malware detection, while Iadarola et al. (2021) proposed an explainable deep learning model for Android malware detection and family identification. *Permutation feature importance* was used in Guerra-Manzanares et al. (2022c) and Guerra-Manzanares et al. (2022b) to analyze important system calls over time for Android malware discrimination, while Guerra-Manzanares et al. (2022a) utilized a similar approach for permissions and malware family evolution. Wu et al. (2021) proposed an interpretable deep learning classifier that discriminates malware and benign samples, providing the relevant features and rationale behind the model decision. Melis et al. (2018) introduced a methodology to generalize explainable decisions of locally-explainable Android malware detectors such as *Drebin* to any nonlinear machine-learning algorithm (i.e., *black box* model) and explain the *global* features influencing the model outputs. Melis et al. (2022) explored the usefulness of gradient-based explanations to assess the robustness of an Android malware detection system. Wu et al. (2022) leveraged heatmaps to analyze and understand the most discriminatory sensitive API calls for malware family classification, while Morcos et al. (2022) used *Shapley Additive Explanations* (SHAP) (i.e., based on *Shapley values*) for global model understanding and local explanations.

All these studies evidence that ML models can be used to detect malware accurately and obtain relevant insights about Android malware data that can be used to expand and enrich domain knowledge and enhance current detection systems, as evidenced in Scalas et al. (2019), or explain historical evolution (Guerra-Manzanares et al., 2022a). Furthermore, model explanations can help to build trust in the predictive model by its users (e.g., expert analysts in SOC environments) and help them to decide and assess different models (Iadarola et al., 2021).

Although interpretation methods are not free from limitations and assumptions (Arrieta et al., 2020; Molnar, 2022), so their output must be contrasted and analyzed carefully (Keane et al., 2021), when applied to high-performance models, they can serve not only to meet legal mandates and predict accurately but also to understand model predictions and extract relevant data insights (Wu et al., 2021), thus building intuitions and trust for model users. While the application of XAI to model and prediction understanding has increased in recent applications, in alignment with the increase of deep learning models, most solutions still focus solely on performance, which limits the knowledge that can be retrieved from the research work. Besides, in the global framework of AI adoption and regulation (Goodman and Flaxman, 2017), model and decisions transparency may constitute a decisive criterion behind the adoption and implementation of certain detection models in production systems. Therefore, adding XAI techniques into the model pipeline can be of great benefit to expanding domain knowledge while predicting accurately and facilitating the adoption of future ML-based solutions.

8. Recommendations for future work

This section provides a brief summary of less studied and unexplored topics within the domain of machine learning based Android malware detection as well as general recommendations for future work. Note that the following list is not exhaustive and we encourage revisiting each challenge section for better context and specific details (Sections 3–7).

- Challenge I: scarcity of updated and representative data sets. In this regard, the focus should be placed on not only data quantity but also, more importantly, on data quality. Studies and data sets should address data-related issues such as inclusion of old and recent threats/samples, representativeness, inclusion of hybrid features, data imbalance and malware/family labeling.
- Challenge II: despite its relevancy, cross-device behavior consistency has been minimally considered by the related work. This issue should be considered in studies using dynamic features and aiming for generalization, focusing on how it can be addressed in production systems.
- Challenge III: concept drift has been not investigated extensively. However, data and threat evolution are actual challenges faced by real-life implementations. Malware detection solutions aiming for real applicability beyond lab testing should evaluate their approaches under data evolution constraints and incorporate updating mechanisms to handle concept drift by design.
- Challenge IV: machine learning security work in the domain of Android malware detection have focused on feature-space evasion attacks, which are mostly demonstrated in a theoretical plane (i.e., input vector manipulation). Future work should aim to expand the domain knowledge to more realistic and practical approaches such as problem-space attacks, data set poisoning and adversarial concept drift.
- Challenge V: explainability is considered in a small number of related work. However, XAI is not only key to regulatory compliance but also to understand threat evolution, being capable of providing valuable insights to improve detection. In this regard, studies are encouraged to not treat their models merely as detection *black boxes* but also aim for model and threat understanding as effective means to expand domain knowledge and enhance threat detection.

9. Conclusions

The large number of research works dealing with Android malware detection, which usually report high-performance metrics using a wide variety of ML algorithms, can be used to tag the problem as *solved* and demotivate further research. This study aimed to assess the validity of this claim and elucidate if Android malware detection can actually be considered a *solved problem* (RQ1). Our conclusion is clear: Android malware detection cannot be considered a problem solved yet. After scrutinizing the literature, we identify five unsolved challenges that support our answer. Ranging from methodological flaws to invalid postulates and data set limitations, these unaddressed issues hamper the road to effective long-term Android malware detection, paving the foundation for further research on the topic. Pending challenges are enumerated and described in detail throughout the paper, motivating future research directions in the problem domain. This comprehensive review of the state of the art aims to elucidate unexplored research directions based on a careful revision of the related literature, motivating research works in the domain that may bring the problem closer to being factually considered as a *problem solved*.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

Abaid, Z., Kaafar, M.A., Jha, S., 2017. Quantifying the impact of adversarial evasion attacks on machine learning based Android malware classifiers. In: 2017 IEEE 16th International Symposium on Network Computing and Applications (NCA). IEEE, pp. 1–10.

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. Hacking Tech. 11 (1), 9–17.

Ahmed, U., Lin, J.C.-W., Srivastava, G., 2021. Generative ensemble learning for mitigating adversarial malware detection in iot. In: 2021 IEEE 29th International Conference on Network Protocols (ICNP). IEEE, pp. 1–5.

Ahmed, U., Lin, J.C.-W., Srivastava, G., 2022. Mitigating adversarial evasion attacks of ransomware using ensemble learning. Comput. Electr. Eng. 100, 107903.

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.

Allix, K., Bissyandé, T.F., Klein, J., Le Traon, Y., 2016. Androzoo: collecting millions of Android apps for the research community. In: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR). IEEE, pp. 468–471.

Alswaina, F., Elleithy, K., 2020. Android malware family classification and analysis: current status and future directions. Electronics 9 (6), 942.

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.

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.

Amos, B., Turner, H., White, J., 2013. Applying machine learning classifiers to dynamic Android malware detection at scale. In: 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC), pp. 1666–1671.

Ananya, A., Aswathy, A., Amal, T., Swathy, P., Vinod, P., Mohammad, S., 2020. Sysdroid: a dynamic ml-based Android malware analyzer using system call traces. Clust. Comput. 23 (4), 2789–2808.

Android, 2021. Run apps on the Android emulator. https://developer.android.com/studio/run/emulator.

Android, 2022a. Package index. https://developer.android.com/reference/packages.

Android, 2022b. Permissions on Android. https://developer.android.com/guide/topics/permissions/overview.

Android, 2022c. Intents and intent filters. https://developer.android.com/guide/components/intents-filters.

Android, 2023. Ui/application exerciser monkey. https://developer.android.com/studio/test/other-testing-tools/monkey.

Anupama, M., Vinod, P., Visaggio, C.A., Arya, M., Philomina, J., Raphael, R., Pinhero, A., Ajith, K., Mathiyalagan, P., 2022. Detection and robustness evaluation of Android malware classifiers. J. Comput. Virol. Hacking Tech. 18 (3), 147–170.

ArgusLab, 2020. Amd dataset - argus cyber security lab. http://amd.arguslab.org/.

Arora, A., Garg, S., Peddoju, S.K., 2014. Malware detection using network traffic analysis in Android based mobile devices. In: 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies, pp. 66–71.

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.

Arp, D., Quiring, E., Pendlebury, F., Warnecke, A., Pierazzi, F., Wressnegger, C., Cavallaro, L., Rieck, K., 2022. Dos and don'ts of machine learning in computer security. In: Proc. of the USENIX Security Symposium.

Arrieta, A.B., Díaz-Rodríguez, N., Del Ser, J., Bennetot, A., Tabik, S., Barbado, A., García, S., Gil-López, S., Molina, D., Benjamins, R., et al., 2020. Explainable artificial intelligence (xai): concepts, taxonomies, opportunities and challenges toward responsible ai. Inf. Fusion 58, 82–115.

AV-Test, 2022. Malware. https://www.av-test.org/en/statistics/malware/.

Bai, T., Luo, J., Zhao, J., Wen, B., Wang, Q., 2021. Recent advances in adversarial training for adversarial robustness. arXiv preprint, arXiv:2102.01356.

Bala, N., Ahmar, A., Li, W., Tovar, F., Battu, A., Bambarkar, P., 2021. Droidenemy: battling adversarial example attacks for Android malware detection. Digit. Commun. Netw.

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.

Biggio, B., Roli, F., 2018. Wild patterns: ten years after the rise of adversarial machine learning. Pattern Recognit. 84, 317–331.

Bläsing, T., Batyuk, L., Schmidt, A.-D., Camtepe, S.A., Albayrak, S., 2010. An Android application sandbox system for suspicious software detection. In: 2010 5th International Conference on Malicious and Unwanted Software. IEEE, pp. 55–62.

Breiman, L., 2001. Random forests. Mach. Learn. 45 (1), 5–32.

Brownlee, J., 2018. Better Deep Learning: Train Faster, Reduce Overfitting, and Make Better Predictions. Machine Learning Mastery.

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. 29 (2), 1–28.

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.

Cai, M., Jiang, Y., Gao, C., Li, H., Yuan, W., 2021. Learning features from enhanced function call graphs for Android malware detection. Neurocomputing 423, 301–307 [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231220316295.

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.

Cara, F., Scalas, M., Giacinto, G., Maiorca, D., 2020. On the feasibility of adversarial sample creation using the Android system api. Information 11 (9), 433.

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.

Ceschin, F., Botacin, M., Gomes, H.M., Pinagé, F., Oliveira, L.S., Grégio, A., 2023. Fast & furious: on the modelling of malware detection as an evolving data stream. Expert Syst. Appl. 212, 118590.

Chen, L., Hou, S., Ye, Y., 2017a. Securedroid: enhancing security of machine learning-based detection against adversarial Android malware attacks. In: Proceedings of the 33rd Annual Computer Security Applications Conference, pp. 362–372.

Chen, L., Hou, S., Ye, Y., Chen, L., 2017b. An adversarial machine learning model against Android malware evasion attacks. In: Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data. Springer, pp. 43–55.

Chen, L., Hou, S., Ye, Y., Xu, S., 2018. Droideye: fortifying security of learning-based classifier against adversarial Android malware attacks. In: 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM). IEEE, pp. 782–789.

Chen, S., Xue, M., Xu, L., 2016. Towards adversarial detection of mobile malware: poster. In: Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking, pp. 415–416.

Chen, S., Xue, M., Fan, L., Ma, L., Liu, Y., Xu, L., 2019a. How can we craft large-scale Android malware? An automated poisoning attack. In: 2019 IEEE 1st International Workshop on Artificial Intelligence for Mobile (AI4Mobile). IEEE, pp. 21–24.

Chen, X., Li, C., Wang, D., Wen, S., Zhang, J., Nepal, S., Xiang, Y., Ren, K., 2019b. Android hiv: a study of repackaging malware for evading machine-learning detection. IEEE Trans. Inf. Forensics Secur. 15, 987–1001.

Cohen, R., Walkowski, D., 2019. Banking trojans: a reference guide to the malware family tree. https://www.f5.com/labs/articles/education/banking-trojans-a-reference-guide-to-the-malware-family-tree.

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.

Darwaish, A., Naït-Abdesselam, F., Titouna, C., Sattar, S., 2021. Robustness of image-based Android malware detection under adversarial attacks. In: ICC 2021-IEEE International Conference on Communications. IEEE, pp. 1–6.

Dassanayake, D., 2021. Millions of Android phones infected by dangerous malware, these phones are at risk. https://www.express.co.uk/life-style/science-technology/1527645/Millions-Android-phones-infected-dangerous-malware-these-phones-at-risk.

Dave, D.D., Rathod, D., 2022. Systematic review on various techniques of Android malware detection. In: International Conference on Computing Science, Communication and Security. Springer, pp. 82–99.

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.
Dini, G., Martinelli, F., Saracino, A., Sgandurra, D., 2012. Madam: a multi-level anomaly detector for Android malware. In: International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security. Springer, pp. 240–253.
Enck, W., Ongtang, M., McDaniel, P., 2009. On lightweight mobile phone application certification. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, pp. 235–245.
Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.-G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N., 2014. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Trans. Comput. Syst. 32 (2), 1–29.
Fan, Y., Ju, M., Hou, S., Ye, Y., Wan, W., Wang, K., Mei, Y., Xiong, Q., 2021. Heterogeneous temporal graph transformer: an intelligent system for evolving Android malware detection. In: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, pp. 2831–2839.
Feizollah, A., Anuar, N.B., Salleh, R., Suarez-Tangil, G., Furnell, S., 2017. Androdialysis: analysis of Android intent effectiveness in malware detection. Comput. Secur. 65, 121–134.
Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D., 2011. Android permissions demystified. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, pp. 627–638.
Feng, P., Ma, J., Sun, C., Xu, X., Ma, Y., 2018. A novel dynamic Android malware detection system with ensemble learning. IEEE Access 6, 30 996–31 011.
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.
Fmind, 2019. Euphony - harmonious unification of cacophonous anti-virus vendor labels for Android malware. https://github.com/fmind/euphony.
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. 46 (4), 1–37.
Goodman, B., Flaxman, S., 2017. European Union regulations on algorithmic decision-making and a “right to explanation”. AI Mag. 38 (3), 50–57.
Grace, M., Zhou, Y., Zhang, Q., Zou, S., Jiang, X., 2012. Riskranker: scalable and accurate zero-day Android malware detection. In: Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, pp. 281–294.
Guerra-Manzanares, A., 2022. About the kronodroid dataset. https://github.com/aleguma/kronodroid.
Guerra-Manzanares, A., Bahsi, H., 2022a. On the application of active learning to handle data evolution in Android malware detection. In: International Conference on Digital Forensics and Cyber Crime. Springer, pp. 256–273.
Guerra-Manzanares, A., Bahsi, H., 2022b. On the relativity of time: implications and challenges of data drift on long-term effective Android malware detection. Comput. Secur. 122, 102835.
Guerra-Manzanares, A., Välbe, M., 2022. Cross-device behavioral consistency: benchmarking and implications for effective Android malware detection. Mach. Learn. Appl. 9, 100357.
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). IEEE, pp. 399–404.
Guerra-Manzanares, A., Nõmm, S., Bahsi, H., 2019b. 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.
Guerra-Manzanares, A., Nomm, S., Bahsi, H., 2019c. In-depth feature selection and ranking for automated detection of mobile malware. In: ICISSP, pp. 274–283.
Guerra-Manzanares, A., Bahsi, H., Nõmm, S., 2021. Kronodroid: time-based hybrid-featured dataset for effective Android malware detection and characterization. Comput. Secur. 110, 102399.
Guerra-Manzanares, A., Bahsi, H., Luckner, M., 2022a. Leveraging the first line of defense: a study on the evolution and usage of Android security permissions for enhanced Android malware detection. J. Comput. Virol. Hacking Tech., 1–32.
Guerra-Manzanares, A., Luckner, M., Bahsi, H., 2022b. Android malware concept drift using system calls: detection, characterization and challenges. Expert Syst. Appl. 206, 117200.
Guerra-Manzanares, A., Luckner, M., Bahsi, H., 2022c. Concept drift and cross-device behavior: challenges and implications for effective Android malware detection. Comput. Secur., 102757.
Gurubaran, 2022. Hackers use new sophisticated version of Android spyware to conduct mobile surveillance. https://cybersecuritynews.com/sophisticated-version-of-android-spyware/.
Hahn, K., 2019a. Malware naming hell part 1: taming the mess of av detection names. https://www.gdatasoftware.com/blog/2019/08/35146-taming-the-mess-of-av-detection-names.
Hahn, K., 2019b. Ransomware identification for the judicious analyst. https://www.gdatasoftware.com/blog/2019/06/31666-ransomware-identification-for-the-judicious-analyst.
Han, Q., Subrahmanian, V.S., Xiong, Y., 2020. Android malware detection via (somewhat) robust irreversible feature transformations. IEEE Trans. Inf. Forensics Secur. 15, 3511–3525.
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). IEEE, pp. 104–111.
Hou, S., Saas, A., Chen, L., Ye, Y., Bourlai, T., 2017. Deep neural networks for automatic Android malware detection. In: Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017, pp. 803–810.
Hou, S., Fan, Y., Zhang, Y., Ye, Y., Lei, J., Wan, W., Wang, J., Xiong, Q., Shao, F., 2019. αcyber: enhancing robustness of Android malware detection system against adversarial attacks on heterogeneous graph based model. In: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, pp. 609–618.
Huang, L., Joseph, A.D., Nelson, B., Rubinstein, B.I., Tygar, J.D., 2011. Adversarial machine learning. In: Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, pp. 43–58.
Hurier, M., Allix, K., Bissyandé, T.F., Klein, J., Le Traon, Y., 2016. On the lack of consensus in anti-virus decisions: metrics and insights on building ground truths of Android malware. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. Springer, pp. 142–162.
Iadarola, G., Martinelli, F., Mercaldo, F., Santone, A., 2021. Towards an interpretable deep learning model for mobile malware detection and family identification. Comput. Secur. 105, 102198 [Online]. Available: https://doi.org/10.1016/j.cose.2021.102198.
Idrees, F., Rajarajan, M., 2014. Investigating the Android intents and permissions for malware detection. In: 2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), pp. 354–358.
Irolla, P., Dey, A., 2018. The duplication issue within the drebin dataset. J. Comput. Virol. Hacking Tech. 14 (3), 245–249.
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.
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.
Jerbi, M., Dagdia, Z.C., Bechikh, S., Said, L.B., 2020. On the use of artificial malicious patterns for Android malware detection. Comput. Secur. 92, 101743.
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.
Kabakus, A.T., 2022. Droidmalwaredetector: a novel Android malware detection framework based on convolutional neural network. Expert Syst. Appl. 206, 117833.
Kabakus, A.T., Dogru, I.A., 2018. An in-depth analysis of Android malware using hybrid techniques. Digit. Investig. 24, 25–33 [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1742287617303183.
Kadir, A.F.A., Stakhanova, N., Ghorbani, A.A., 2015. Android botnets: what urls are telling us. In: International Conference on Network and System Security. Springer, pp. 78–91.
Kapratwar, A., Di Troia, F., Stamp, M., 2017. Static and dynamic analysis of Android malware. In: ICISSP, pp. 653–662.
Karn, R.R., Kudva, P., Huang, H., Suneja, S., Elfadel, I.M., 2021. Cryptomining detection in container clouds using system calls and explainable machine learning. IEEE Trans. Parallel Distrib. Syst. 32 (3), 674–691.
Kaspersky, 2020. Mobile security: Android vs ios - which one is safer? https://www.kaspersky.com/resource-center/threats/android-vs-iphone-mobile-security.
Kaspersky, 2021. Rules for naming. https://encyclopedia.kaspersky.com/knowledge/rules-for-naming.
Keane, M.T., Kenny, E.M., Delaney, E., Smyth, B., 2021. If only we had better counterfactual explanations: five key deficits to rectify in the evaluation of counterfactual xai techniques. arXiv preprint, arXiv:2103.01035.
Kinkead, M., Millar, S., McLaughlin, N., O'Kane, P., 2021. Towards explainable cnns for Android malware detection. Proc. Comput. Sci. 184 (2019), 959–965 [Online]. Available: https://doi.org/10.1016/j.procs.2021.03.118.
Kiss, N., Lalande, J.-F., Leslous, M., Viet Triem Tong, V., 2016. Kharon dataset: Android malware under a microscope. In: Learning from Authoritative Security Experiment Results. The USENIX Association, San Jose, United States [Online]. Available: https://hal-univ-orleans.archives-ouvertes.fr/hal-01300752.
Kiss, N., Lalande, J.-F., Leslous, M., Viet Triem Tong, V., 2021. Kharon malware dataset. http://kharon.gforge.inria.fr/dataset.
Koodous, 2022. Koodous - collaborative platform for Android malware analysts. https://koodous.com/.
Korycki, Ł., Krawczyk, B., 2022. Adversarial concept drift detection under poisoning attacks for robust data stream mining. Mach. Learn., 1–36.
Kouliaridis, V., Kambourakis, G., 2021. A comprehensive survey on machine learning techniques for Android malware detection. Information 12 (5).

Laricchia, F., 2022. Mobile operating systems' market share worldwide from 1st quarter 2009 to 4th quarter 2022. https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/.

Lashkari, A.H., Kadir, A.F.A., Gonzalez, H., Mbah, K.F., Ghorbani, A.A., 2017. Towards a network-based framework for Android malware detection and characterization. In: 2017 15th Annual Conference on Privacy, Security and Trust (PST), pp. 233–23 309.

Lashkari, A.H., Kadir, A.F.A., Taheri, L., Ghorbani, A.A., 2020. Toward developing a systematic approach to generate benchmark Android malware datasets and classification. In: 2018 International Carnahan Conference on Security Technology (ICCST). IEEE, pp. 1–7.

Leeds, M., Keffeler, M., Atkison, T., 2017. A comparison of features for Android malware detection. In: Proceedings of the SouthEast Conference, Ser. ACM SE '17. Association for Computing Machinery, New York, NY, USA, pp. 63–68.

Lei, T., Qin, Z., Wang, Z., Li, Q., Ye, D., 2019. Evedroid: event-aware Android malware detection against model degrading for iot devices. IEEE Int. Things J. 6 (4), 6668–6680.

Li, D., Li, Q., 2020. Adversarial deep ensemble: evasion attacks and defenses for malware detection. IEEE Trans. Inf. Forensics Secur. 15, 3886–3900.

Li, D., Wang, Z., Xue, Y., 2018. Fine-grained Android malware detection based on deep learning. In: 2018 IEEE Conference on Communications and Network Security (CNS), pp. 1–2.

Li, D., Li, Q., Ye, Y., Xu, S., 2021a. Arms race in adversarial malware detection: a survey. ACM Comput. Surv. 55 (1), 1–35.

Li, D., Li, Q., Ye, Y., Xu, S., 2021b. A framework for enhancing deep neural networks against adversarial malware. IEEE Trans. Netw. Sci. Eng. 8 (1), 736–750.

Li, H., Zhou, S., Yuan, W., Li, J., Leung, H., 2019a. Adversarial-example attacks toward Android malware detection system. IEEE Syst. J. 14 (1), 653–656.

Li, H., Zhou, S., Yuan, W., Luo, X., Gao, C., Chen, S., 2021c. Robust Android malware detection against adversarial example attacks. In: Proceedings of the Web Conference 2021, pp. 3603–3612.

Li, W., Bala, N., Ahmar, A., Tovar, F., Battu, A., Bambarkar, P., 2019b. A robust malware detection approach for Android system against adversarial example attacks. In: 2019 IEEE 5th International Conference on Collaboration and Internet Computing (CIC). IEEE, pp. 360–365.

Li, X., Kong, K., Xu, S., Qin, P., He, D., 2021d. Feature selection-based Android malware adversarial sample generation and detection method. IET Inf. Secur. 15 (6), 401–416.

Liang, S., Du, X., 2014. Permission-combination-based scheme for Android mobile malware detection. In: 2014 IEEE International Conference on Communications (ICC), pp. 2301–2306.

Lin, P., Ye, K., Hu, Y., Lin, Y., Xu, C.-Z., 2022. A novel multimodal deep learning framework for encrypted traffic classification. IEEE/ACM Trans. Netw.

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.

Liu, K., Xu, S., Xu, G., Zhang, M., Sun, D., Liu, H., 2020. A review of Android malware detection approaches based on machine learning. IEEE Access 8, 124 579–124 607.

Liu, X., Du, X., Zhang, X., Zhu, Q., Wang, H., Guizani, M., 2019. Adversarial samples on Android malware detection systems for iot systems. Sensors 19 (4), 974.

Liu, Y., Tantithamthavorn, C., Li, L., Liu, Y., 2021. Deep learning for Android malware defenses: a systematic literature review. arXiv preprint, arXiv:2103.05292.

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.

Mahdavifar, S., Kadir, A.F.A., Fatemi, R., Alhadidi, D., Ghorbani, A.A., 2020. Dynamic Android malware category classification using semi-supervised deep learning. In: 2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech). IEEE, pp. 515–522.

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

Mcdonald, J., Herron, N., Glisson, W., Benton, R., 2021. Machine learning-based Android malware detection using manifest permissions. In: Proceedings of the 54th Hawaii International Conference on System Sciences, p. 6976.

Meijin, L., Zhiyang, F., Junfeng, W., Luyu, C., Qi, Z., Tao, Y., Yinwei, W., Jiaxuan, G., 2022. A systematic overview of Android malware detection. Appl. Artif. Intell. 36 (1), 2007327.

Melis, M., Maiorca, D., Biggio, B., Giacinto, G., Roli, F., 2018. Explaining black-box Android malware detection. In: 2018 26th European Signal Processing Conference (EUSIPCO). IEEE, pp. 524–528.

Melis, M., Scalas, M., Demontis, A., Maiorca, D., Biggio, B., Giacinto, G., Roli, F., 2022. Do gradient-based explanations tell anything about adversarial robustness to Android malware? Int. J. Mach. Learn. Cybern. 13 (1), 217–232.

Microsoft, 2021. Malware names. https://docs.microsoft.com/en-us/windows/security/threat-protection/intelligence/malware-naming.

Molina-Coronado, B., Mori, U., Mendiburu, A., Miguel-Alonso, J., 2023. Towards a fair comparison and realistic evaluation framework of Android malware detectors based on static analysis and machine learning. Comput. Secur. 124, 102996.

Molnar, C., 2022. Interpretable Machine Learning, 2nd ed. [Online]. Available: https://christophm.github.io/interpretable-ml-book.

Morcos, M., Al Hamadi, H., Damiani, E., Nandyala, S., McGillion, B., 2022. A surrogate-based technique for Android malware detectors' explainability. In: 2022 18th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), pp. 112–117.

Muzaffar, A., Hassen, H.R., Lones, M.A., Zantout, H., 2022. An in-depth review of machine learning based Android malware detection. Comput. Secur., 102833.

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.

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.

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. Priv. Secur. 22 (2), 1–34.

Ou, F., Xu, J., 2022. S3feature: a static sensitive subgraph-based feature for Android malware detection. Comput. Secur. 112, 102513.

Parkour, M., 2019. Contagio minidump. http://contagiominidump.blogspot.com/.

Peiravian, N., Zhu, X., 2013. Machine learning for Android malware detection using permission and api calls. In: 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, pp. 300–305.

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.

Peng, H., Gates, C., Sarma, B., Li, N., Qi, Y., Potharaju, R., Nita-Rotaru, C., Molloy, I., 2012. Using probabilistic generative models for ranking risks of Android apps. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, pp. 241–252.

Petsas, T., Voyatzis, G., Athanasopoulos, E., Polychronakis, M., Ioannidis, S., 2014. Rage against the virtual machine: hindering dynamic analysis of Android malware. In: Proceedings of the Seventh European Workshop on System Security, pp. 1–6.

Pierazzi, F., Pendlebury, F., Cortellazzi, J., Cavallaro, L., 2020. Intriguing properties of adversarial ml attacks in the problem space. In: 2020 IEEE Symposium on Security and Privacy (SP). IEEE, pp. 1332–1349.

Ruth C., 2022. Over 30 million new malware samples found in 2022 as cyber threats evolve. https://atlasvpn.com/blog/over-30-million-new-malware-samples-found-in-2022-as-cyber-threats-evolve.

Rahali, A., Lashkari, A.H., Kaur, G., Taheri, L., Francois, G., Massicotte, F., 2020. Didroid: Android malware classification and characterization using deep image learning. In: 10th International Conference on Communication and Network Security.

Rathore, H., Nikam, P., Sahay, S.K., Sewak, M., 2021a. Identification of adversarial Android intents using reinforcement learning. In: 2021 International Joint Conference on Neural Networks (IJCNN). IEEE, pp. 1–8.

Rathore, H., Sahay, S.K., Nikam, P., Sewak, M., 2021b. Robust Android malware detection system against adversarial attacks using q-learning. Inf. Syst. Front. 23 (4), 867–882.

Rathore, H., Samavedhi, A., Sahay, S.K., Sewak, M., 2021c. Robust malware detection models: learning from adversarial attacks and defenses. Forensic Sci. Int., Digit. Investig. 37, 301183.

Rathore, H., Sasan, A., Sahay, S.K., Sewak, M., 2022. Defending malware detection models against evasion based adversarial attacks. Pattern Recognit. Lett. 164, 119–125.

Razgallah, A., Khoury, R., Hallé, S., Khanmohammadi, K., 2021. A survey of malware detection in Android apps: recommendations and perspectives for future research. Comput. Sci. Rev. 39, 100358.

Reddy, R., Swamy, M.K., Kumar, D.A., 2021. Feature and sample size selection for malware classification process. In: ICCCE 2020. Springer, pp. 217–223.

Renjith, G., Laudanna, S., Aji, S., Visaggio, C.A., Vinod, P., 2022a. Gang-mam: gan based engine for modifying Android malware. SoftwareX 18, 100977.

Renjith, G., Vinod, P., Aji, S., 2022b. Evading machine-learning-based Android malware detector for iot devices. IEEE Syst. J.

Şahin, D.Ö., Kural, O.E., Akleylek, S., Kılıç, E., 2021. A novel permission-based Android malware detection system using feature selection based on linear regression. Neural Comput. Appl., 1–16.

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.

Salem, A., Banescu, S., Pretschner, A., 2021. Maat: automatically analyzing virustotal for accurate labeling and effective malware detection. ACM Trans. Priv. Secur. 24 (4).

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.

Savage, K., Coogan, P., Lau, H., 2015. The evolution of ransomware. Symantec.

Scalas, M., Maiorca, D., Mercaldo, F., Visaggio, C.A., Martinelli, F., Giacinto, G., 2019. On the effectiveness of system API-related information for Android ransomware detection. Comput. Secur. 86, 168–182 [Online]. Available: https://doi.org/10.1016/j.cose.2019.06.004.

Schmidt, A.-D., 2011. Detection of Smartphone Malware.

Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., Weiss, Y., 2012. "andromaly": a behavioral malware detection framework for Android devices. J. Intell. Inf. Syst. 38 (1), 161–190.

Shahpasand, M., Hamey, L., Vatsalan, D., Xue, M., 2019. Adversarial attacks on mobile malware detection. In: 2019 IEEE 1st International Workshop on Artificial Intelligence for Mobile (AI4Mobile). IEEE, pp. 17–20.

Shapley, L., 1953. Quota Solutions of N-Person Games, p. 343. Edited by Emil Artin and Marston Morse.
Sharma, T., Rattan, D., 2021. Malicious application detection in Android—a systematic literature review. Comput. Sci. Rev. 40, 100373.
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. 11 (2), 34–45.
Simonyan, K., Vedaldi, A., Zisserman, A., 2013. Deep inside convolutional networks: visualising image classification models and saliency maps. arXiv preprint, arXiv:1312.6034.
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.
Spadafora, A., 2022. Millions of Android users at risk of attack after widespread security issue uncovered. https://www.techradar.com/news/millions-of-android-users-at-risk-of-attack-after-widespread-security-issue-uncovered.
Surendran, R., Thomas, T., Emmanuel, S., 2020a. A tan based hybrid model for Android malware detection. J. Inf. Secur. Appl. 54, 102483.
Surendran, R., Thomas, T., Emmanuel, S., 2020b. Gsdroid: graph signal based compact feature representation for Android malware detection. Expert Syst. Appl. 159, 113581.
Syrris, V., Geneiatakis, D., 2021. On machine learning effectiveness for malware detection in Android os using static analysis data. J. Inf. Secur. Appl. 59, 102794.
T. U. Braunschweig, 2020. The drebin dataset. https://www.sec.cs.tu-bs.de/~danarp/drebin/index.html.
Taheri, L., Kadir, A.F.A., Lashkari, A.H., 2019. Extensible Android malware detection and family classification using network-flows and api-calls. In: 2019 International Carnahan Conference on Security Technology (ICCST). IEEE, pp. 1–8.
Taheri, R., Ghahramani, M., Javidan, R., Shojafar, M., Pooranian, Z., Conti, M., 2020a. Similarity-based Android malware detection using Hamming distance of static binary features. Future Gener. Comput. Syst. 105, 230–247.
Taheri, R., Javidan, R., Shojafar, M., Pooranian, Z., Miri, A., Conti, M., 2020b. On defending against label flipping attacks on malware detection systems. Neural Comput. Appl. 32 (18), 781–14 800.
Taheri, R., Javidan, R., Shojafar, M., Vinod, P., Conti, M., 2020c. Can machine learning model with static features be fooled: an adversarial machine learning approach. Clust. Comput. 23 (4), 3233–3253.
Taheri, R., Shojafar, M., Alazab, M., Tafazolli, R., 2020d. Fed-iiot: a robust federated malware detection architecture in industrial iot. IEEE Trans. Ind. Inform. 17 (12), 8442–8452.
Talha, K.A., Alper, D.I., Aydin, C., 2015. Apk auditor: permission-based Android malware detection system. Digit. Investig. 13, 1–14.
Tam, K., Khan, S.J., Fattori, A., Cavallaro, L., 2015. Copperdroid: automatic reconstruction of Android malware behaviors. In: Ndss.
Tchakounté, F., Dayang, P., 2013. System calls analysis of malwares on Android. Int. J. Sci. Technol. 2 (9), 669–674.
Timothy, M., 2022. Why you should uninstall your Android antivirus software. https://www.makeuseof.com/uninstall-android-antivirus/.
Tong, F., Yan, Z., 2017. A hybrid approach of mobile malware detection in Android. J. Parallel Distrib. Comput. 103, 22–31.
Townsend, K., 2020. How smartphones have become one of the largest attack surfaces. https://blog.avast.com/smartphones-and-increasing-mobile-threats-avast.
U. du Luxembourg, 2021. Androzoo - lists of apks. https://androzoo.uni.lu/lists.
U. du Luxembourg, 2022. Androzoo. https://androzoo.uni.lu.
U. of New Brunswick, 2020a. Android botnet dataset. https://www.unb.ca/cic/datasets/android-botnet.html.
U. of New Brunswick, 2020b. Android adware and general malware dataset (cic-aagm2017). https://www.unb.ca/cic/datasets/android-adware.html.
U. of New Brunswick, 2020c. Android malware dataset (cic-andmal2017). https://www.unb.ca/cic/datasets/andmal2017.html.
U. of New Brunswick, 2020d. Investigation of the Android malware (cic-invesandmal2019). https://www.unb.ca/cic/datasets/invesandmal2019.html.
U. of New Brunswick, 2020e. Cccs-cic-andmal-2020. https://www.unb.ca/cic/datasets/andmal2020.html.
U. of New Brunswick, 2020f. Cicmaldroid 2020. https://www.unb.ca/cic/datasets/maldroid-2020.html.
Ullah, S., Ahmad, T., Buriro, A., Zara, N., Saha, S., 2022. Trojandetector: a multi-layer hybrid approach for trojan detection in Android applications. Appl. Sci. 12 (21).
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.
VirusShare, 2022. Virusshare. https://virusshare.com/.
VirusTotal, 2022. Virustotal academic malware samples. http://www.virustotal.com.
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, C., Zhang, L., Zhao, K., Ding, X., Wang, X., 2021. Advandmal: adversarial training for Android malware detection and family classification. Symmetry 13 (6), 1081.
Wang, W., Zhao, M., Wang, J., 2019. Effective Android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. J. Ambient Intell. Humaniz. Comput. 10 (8), 3035–3043.
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.
Wei, W., Wang, J., Yan, Z., Ding, W., 2022. Epmdroid: efficient and privacy-preserving malware detection based on sgx through data fusion. Inf. Fusion.
Wu, B., Chen, S., Gao, C., Fan, L., Liu, Y., Wen, W., Lyu, M.R., 2021. Why an Android app is classified as malware: toward malware classification interpretation. ACM Trans. Softw. Eng. Methodol. 30 (2), 1–29.
Wu, Y., Dou, S., Zou, D., Yang, W., Qiang, W., Jin, H., 2022. Contrastive learning for robust Android malware familial classification. IEEE Trans. Dependable Secure Comput., 1–14.
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.H., Chen, K., 2018. Deeprefiner: multi-layer Android malware detection system applying deep neural networks. In: 2018 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, pp. 473–487.
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.
Yadav, P., Menon, N., Ravi, V., Vishvanathan, S., Pham, T.D., 2022. Efficientnet convolutional neural networks-based Android malware detection. Comput. Secur. 115, 102622.
Yang, W., Kong, D., Xie, T., Gunter, C.A., 2017. Malware detection in adversarial settings: exploiting feature evolutions and confusions in Android apps. In: Proceedings of the 33rd Annual Computer Security Applications Conference, pp. 288–302.
Yang, Y., Du, X., Yang, Z., Liu, X., 2021. Android malware detection based on structural features of the function call graph. Electronics 10 (2).
Yerima, S.Y., Sezer, S., Muttik, I., 2015. High accuracy Android malware detection using ensemble learning. IET Inf. Secur. 9 (6), 313–320.
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.
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.
Yumlembam, R., Issac, B., Jacob, S.M., Yang, L., 2022. Iot-based Android malware detection using graph neural network with adversarial defense. IEEE Int. Things J.
Zhang, J., Zhang, C., Liu, X., Wang, Y., Diao, W., Guo, S., 2021a. Shadowdroid: practical black-box attack against ml-based Android malware detection. In: 2021 IEEE 27th International Conference on Parallel and Distributed Systems (ICPADS). IEEE, pp. 629–636.
Zhang, N., Xue, J., Ma, Y., Zhang, R., Liang, T., Tan, Y.-a., 2021b. Hybrid sequence-based Android malware detection using natural language processing. Int. J. Intell. Syst. 36 (10), 5770–5784.
Zhang, S., Xie, X., Xu, Y., 2020a. A brute-force black-box method to attack machine learning-based systems in cybersecurity. IEEE Access 8, 250–128 263.
Zhang, X., Zhang, Y., Zhong, M., Ding, D., Cao, Y., Zhang, Y., Zhang, M., Yang, M., 2020b. 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.
Zhao, K., Zhou, H., Zhu, Y., Zhan, X., Zhou, K., Li, J., Yu, L., Yuan, W., Luo, X., 2021a. Structural attack against graph based Android malware detection. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 3218–3235.
Zhao, L., Wang, J., Chen, Y., Wu, F., Liu, Y., et al., 2021b. Droidmfc: a novel Android malware family classification scheme based on static analysis. arXiv preprint, arXiv:2101.03965.
Zhou, Y., Jiang, X., 2012. Dissecting Android malware: characterization and evolution. In: 2012 IEEE Symposium on Security and Privacy, pp. 95–109.
Zhou, Y., Jiang, X., 2015. Malgenome project. http://www.malgenomeproject.org/.
Zhu, Dali, Jin, Hao, Yang, Ying, Wu, D., Chen, Weiyi, 2017. Deepflow: deep learning-based malware detection by mining Android application for abnormal usage of sensitive data. In: 2017 IEEE Symposium on Computers and Communications (ISCC), pp. 438–443.
Zhu, S., Shi, J., Yang, L., Qin, B., Zhang, Z., Song, L., Wang, G., 2020a. Measuring and modeling the label dynamics of online {anti-malware} engines. In: 29th USENIX Security Symposium (USENIX Security 20), pp. 2361–2378.
Zhu, S., Zhang, Z., Yang, L., Song, L., Wang, G., 2020b. Benchmarking label dynamics of virustotal engines. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, Ser. CCS '20. Association for Computing Machinery, New York, NY, USA, pp. 2081–2083.

Alejandro Guerra Manzanares is currently a researcher at the Center for Digital Forensics and Cyber Security, Department of Software Science, at TalTech (Estonia), where he obtained his Ph.D. degree in Information Technology (cybersecurity) in September, 2022. Previously, he received a BA degree in criminology (cum laude) from the Autonomous University of Barcelona (Spain) in 2013 and a BS degree in ICT engineering (cum laude) from the Polytechnic University of Catalonia (Spain) in 2017. In 2018, he received an M.Sc. in cybersecurity (cum laude) from TalTech. His research interests are on the intersection of machine learning and cybersecurity, where he has published over 15 scientific articles in the areas of Android malware detection and IoT botnet detection.