Trust Under Siege: Label Spoofing Attacks Against Machine Learning for Android Malware Detection

Tianwei Lan[1], Luca Demetrio[2], Farid Nait-Abdesselam[1], Yufei Han[3], and Simone Aonzo[4]

[1]Université Paris Cité, France
[2]University of Genova, Italy
[3]INRIA, France
[4]EURECOM, France

***Abstract*—Machine Learning (ML) malware detectors rely heavily on crowd-sourced AntiVirus (AV) labels, with platforms like VirusTotal serving as trusted sources of malware annotations. But what if attackers could manipulate these labels to classify benign software as malicious? We introduce label spoofing attacks, a new threat that contaminates crowd-sourced datasets by embedding minimal and undetectable malicious patterns into benign samples. These patterns coerce AV engines into misclassifying legitimate files as harmful, enabling poisoning attacks against ML-based malware classifiers trained on those data. We demonstrate this scenario by developing AndroVenom, a methodology for polluting realistic data sources and launching subsequent poisoning attacks against ML malware detectors. Experiments show that not only are state-of-the-art feature extractors unable to filter such injections, but various ML models experience Denial-of-Service (DoS) with as little as 1% poisoned samples. Additionally, attackers can flip decisions for specific unaltered benign samples by modifying only 0.015% of the training data, threatening their reputation and market share, while evading anomaly detectors operating on the training data. We conclude by raising concerns about the trustworthiness of ML training processes based on AV annotations and argue that further investigation is needed to develop more reliable labeling strategies.**

***Index Terms*—Poisoning attack, Android malware detection, AntiVirus engine.**

I. Introduction

In response to the proliferation of malware samples, both the security research community and industrial security service providers increasingly rely on Machine Learning (ML) techniques to discriminate malicious from benign applications [1]. It is achieved by capturing correlations between features extracted from samples and their corresponding labels within the training dataset. Hence, the quality of ML-based malware detectors directly depends on the availability of reliable ground truth for both benign and malicious applications in the training set. Over time, the community has refined best practices for constructing high-quality datasets for training purposes. To collect benign applications, researchers and practitioners often acquire samples from well-monitored repositories. In the case of Android applications (APKs), benign samples are typically fetched from the Google Play Store [2], where each application undergoes rigorous vetting processes that minimize the presence of malware on the platform. Conversely, for Windows programs, data is often extracted from fresh installations of community-maintained packages hosted on Chocolatey, which also enforces strict moderation and review procedures to prevent contamination [3]. To collect malicious APKs, security service providers and practitioners turn to online crowdsourcing platforms such as VirusTotal [4], [5], which aggregates scan results from over 70 commercial AntiVirus (AV) engines [6] while storing each submitted sample. Samples are typically included in datasets as malicious when the number of AV detections exceeds a predefined threshold [2], [3], [5], [7]–[9]. However, while the use of these platforms streamlines and automates malware annotation, it also introduces a subtle yet critical vulnerability. Commercial AVs often rely on signature-based techniques [10], detecting malware by identifying specific artifacts such as cryptographic hashes, unique strings, or characteristic code patterns. This reliance enables an attacker to strategically embed recognized *malicious byte patterns* into otherwise benign programs, forcing AV engines to flag them as malicious. Notably, injected signatures, such as the EICAR signature [11], do not introduce new malicious functionality, and the modified programs do not exhibit harmful behavior at runtime. Once uploaded to VirusTotal or similar antivirus detection services, these modified samples are labeled as malicious solely due to the presence of the injected signature. This situation creates a fundamental ambiguity: while AVs correctly identify a known malicious artifact, the program's functionality remains benign. We define and exploit this ambiguity as a *label spoofing attack*. When downstream users retrieve such mislabeled samples from crowdsourcing platforms and incorporate them into ML training datasets, attackers gain indirect influence over the resulting models. Specifically, they can stage *poisoning attacks* [12], [13], ranked among the most impactful threats to ML systems by OWASP [14]. These attacks can either render malware detectors unusable (*Denial of Service*) or trigger excessive false alarms against targeted benign applications, damaging their reputation and hindering their distribution. Such degradation can lead to *(i)* erosion of trust in crowdsourced malware annotations, *(ii)* financial losses for AV vendors due to increased false positives, and *(iii)* reputational harm to software developers whose products are repeatedly misclassified as malicious.

In extreme scenarios, this mechanism could be abused by malicious actors or state entities to censor legitimate software under the guise of national security by deliberately polluting training datasets. Furthermore, human analysts may waste significant time investigating spoofed samples that contain no real malicious behavior. To analyze the threat posed by label spoofing attacks against ML-based malware classifiers, we investigate the following research questions:

RQ1. How can an attacker inject a malicious signature with *minimal* manipulation that remains undetected by feature extractors?

RQ2. What conditions enable label spoofing attacks, and what level of effort is required for an attacker to tamper with the labeling systems of AVs?

RQ3. How many poisoned benign samples are required to launch Denial of Service (DoS) attacks that degrade the performance of ML-based malware detectors in production?

RQ4. Can label spoofing attacks induce targeted false alarms on selected benign samples while preserving overall malware classification accuracy?

We answer these questions by exposing the risks of over-reliance on AV labeling through the development of *AndroVenom*, a framework that exploits this weakness and paves the way for poisoning attacks against ML malware detectors (Sect. III). Focusing on Android, given its unrivaled market share [15], we address **RQ1** by introducing a novel manipulation of Android Applications (APKs). This manipulation is negligible in size relative to the original application yet causes legitimate samples to be flagged as malicious by an average of 20 AV engines, a threshold commonly considered indicative of true malware [5]. We further demonstrate that this modification remains *invisible* to state-of-the-art feature extraction pipelines, including Drebin, MaMaDroid, and MalScan [8], [16], [17], as their preprocessing outputs remain unchanged. We then address **RQ2** by demonstrating the practical feasibility of label spoofing attacks on Android APKs (Sect. IV-B). Repackaged applications are consistently flagged as malicious because AV engines identify the injected artifact via its cryptographic hash and associate it with attacker-selected malware families. Finally, we answer **RQ3** and **RQ4** through extensive experimental evaluation (Sect. IV-E and Sect. IV-F). Using a dataset of 40K samples to emulate a realistic ML malware classification pipeline using Drebin, MaMaDroid, and MalScan feature extractors, we demonstrate that poisoning just 1% of the training set can degrade detector performance by 15% in low-false-positive regimes. More alarmingly, only 0.015% poisoned samples are sufficient to induce targeted misclassifications of specific benign applications while maintaining overall performance. Lastly, we show that removing label-spoofed samples is non-trivial: ad-hoc defenses against clean-label poisoning can introduce performance degradation unacceptable in production environments. While poisoning attacks are already well studied, AndroVenom highlights the overlooked risks stemming from blind trust in crowdsourced AV annotations—an issue that demands joint attention from both ML practitioners and AV vendors.

II. Background

Before delving into the details of our work, we summarize the core concepts needed to understand our methodology.

Android Applications. Mobile apps running on Android are distributed in the form of App Package (APK) files, which are an extension of the Java JAR format. An APK contains several files and folders, and in particular: *(i)* `AndroidManifest.xml`, a file needed to identify and run the APK, containing metadata, such as its name, version, permissions, and high-level components; *(ii)* `classes.dex`, files that contain the compiled bytecode, usually written in Java or Kotlin; *(iii)* `lib`, a folder containing the native libraries of the APK written in C or C++, compiled for different architectures (ARM and x86), and loaded through the Java Native Interface (JNI); *(iv)* `assets`, a folder containing resources that do not require compilation, such as texts, XML, fonts, music, and videos; *(v)* `res`, a folder of resources that require compilation, like XML layouts, images, strings, and colors.

AntiVirus Software (AV). This family of software is tasked with detecting (binary classification) and identifying (family classification) malicious activity from an input byte stream (e.g., a single file or network traffic). It plays a vital role in defending modern IT systems [1], [5], [10], [18]–[20]. Being composed of different layers of defenses, one key component of AVs is the collection of *signatures*, used to spot already-known threats. Signatures are typically a collection of cryptographic hashes or byte streams extracted from previously collected malware samples [21]. It is usually the responsibility of human analysts to write these signatures when they encounter new malicious code, but they can also be generated automatically [22]. With efficient pattern-matching algorithms, signatures are tested on input samples, and those are flagged as malicious once matches are found. Due to their speed and efficiency, detection through pattern matching is the most typical and preferred method for AVs. Crucially, their outputs are frequently reused as labels for training ML-based malware detectors, creating a transitive trust dependency between AV decisions and ML training data.

APK Repackaging. It refers to the practice of extracting contents from an APK, modifying it, and recreating another valid APK. There are several contexts in which repackaging can be used, including reverse engineering tasks or illegally cloning APKs from the Play Store. While in theory this operation is always possible, Apktool [23], the primary tool used for this operation, is obviously not free of bugs.

Android Malware Classification with ML. There are multiple ways to detect Android malware with ML techniques. In this study, we focus on *static analysis*, which relies on features extracted from the structure and metadata of programs rather than their behavior at runtime. While we have reviewed many different feature extractors (see Sect. V), we will focus on the Drebin, MaMaDroid, and MalScan feature set [8], [16], [17] in this work. The Drebin feature set relies on permissions, API calls, and network addresses that can be extracted from input APKs, turning them into binary feature vectors. The MaMaDroid feature set extracts the sequence of API calls retrieved by browsing the code. It then constructs

a Markov chain to represent the transitions between API calls, grouped at varying levels of abstraction, such as API families or packages, to capture the APK's behavioral patterns. Features derived from the Markov chain, such as transition probabilities, are used to train machine learning classifiers to distinguish between benign and malicious APKs. The MalScan feature set extracts function call graphs of APKs based on static analysis and performs social-network-based centrality analysis to represent the semantic features of the graphs. The obtained feature vectors are fed to machine learning models for classification.

Poisoning Attacks. These strategies manipulate the behavior of the targeted model by deliberately introducing tampered training samples. Poisoning attacks can be formalized as *Denial of Service* (also known as *availability*) and *integrity* attacks [13], [24]. A *Denial of Service* (DoS) attack aims to subvert the overall accuracy at the test time of the model trained with poisoned training data. Consequently, the attack becomes noticeable only after the poisoned model has been deployed, not before. On the other hand, *integrity* attacks seek to deform the victim model's performance on specific test samples while preserving the overall test-time accuracy. Attackers can employ various training data poisoning strategies, including *clean-label*, *dirty-label*, and *label-flip* attacks. Specifically, *clean-label* attacks indicate that the attacker can only tamper with the features of training samples without changing their labels. In contrast, in *dirty-label* attacks, the attacker can manipulate both features and labels to poison training samples. *Label-flip* attacks only change the labels of training samples without changing their features.

III. AndroVenom: Label Spoofing Attacks against Android Malware Detection

We now describe AndroVenom (Fig. 1), our methodology for poisoning training datasets used in ML-based malware classification via label spoofing. We first detail the selection process for identifying easy-to-spot malware samples intended for injection into benign APKs (Sect. III-A). Subsequently, we outline the staging of the poisoning attacks (Sect. III-B).

Crucially, this attack does not target the internal learning mechanisms of AntiVirus (AV) engines themselves. Instead, it targets the labeling pipeline that feeds ML-based malware classifiers, which are often trained on crowd-sourced AV annotations. Our **threat model** does not assume that AV vendors directly retrain their primary detection engines on public datasets or VirusTotal submissions. Rather, the attack targets the downstream ML pipeline, where AV consensus (typically aggregated via platforms like VirusTotal) is treated as the ground truth for dataset curation. By coercing AV engines into producing incorrect labels through malicious signature injection, an attacker effectively contaminates the training data used for future ML-based detectors.

A. Label Spoofing: Coercing Mispredictions from AVs

We define *label spoofing attacks* as attacks that aim to force an AV to misclassify a legitimate input APK as malicious and to attribute it to a specific malware family. This strategy is in stark contrast with the usual behavior of attackers that try to bypass AVs. To the best of our knowledge, we are the first to identify and analyze this counterintuitive behavior. In particular, we envision attackers that can easily pollute crowd-sourced datasets because of the blind trust attributed to annotations collected from AVs. To do so, we first need to study how an attacker can manipulate benign APKs to achieve their goal. Among all possible manipulations, in this work, we seek to inject malicious samples that satisfy different constraints to be considered effective: *(i)* their inclusion must trigger as many AV detections as possible and be categorized as a specific malware family; *(ii)* it should take significant engineering effort to filter out the presence of these malicious samples; and *(iii)* the injection location must be chosen accordingly to avoid impacting the output of feature extraction algorithms (we report our findings related to those questions in Table I).

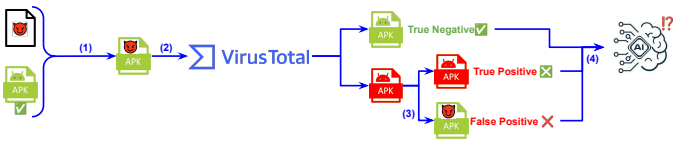

Figure 1: Workflow of AndroVenom: (1) the attacker injects a malicious file into benign APKs and repacks them; (2) the attacker submits the modified benign APKs to VirusTotal, where they are now mislabelled as malware; (3) these mislabelled APKs are included in training sets of ML Android malware detectors; (4) the detectors are compromised after training.

Detectable Malware Samples. To satisfy the first constraint, we consider samples that trigger more than 20 AV detections on VirusTotal (see the *Detection* column in Table I). This conservative threshold was chosen to eliminate false positives, as prior research suggests that false detection rates drop significantly after 5 detections [5]. Because we require these files to be consistently classified within a specific malware family, we must determine the consensus among potentially conflicting AV responses. We leverage the AVClass tool [25], [26] to resolve these labels, filtering out any samples for which a definitive malware family cannot be identified.

Disguised as Common Files. To minimize the risk of being flagged by manual analysis (the second constraint), we focus on malicious samples that share file types commonly found in benign APKs. For example, Unity-based games often contain Windows executables due to cross-platform deployment; consequently, a Windows library inside a mobile game archive may not appear suspicious. Our analysis utilizes the benign APK dataset proposed by Ruggia et al. [7], which comprises 20,769 benign APKs across 50 Google Play Store categories. Due to legal constraints, we used the hashes provided by the authors to download the APKs from VirusTotal. We then identified the *ten most common MIME (Multipurpose Internet Mail Extensions) types* within these APKs and their likelihood of appearance (column *Inclusion* in Table I). We refined our selection of malicious samples by retaining only those that match these common MIME types. While thousands of

MIME Type	Inclusion	Malicious File MD5	Detections	Family	Size [B]
text/xml	100%	c4358d29533117779296b7e555aca54f	29/62	groooboor	316
app/octet-stream	99.4%	1ce7054c91ea5aca1e9ca75e97c7c1d8	33/62	gnaeus	650
image/gif	98.7%	8575a8e68f927705a04e668fdc2246c1	32/62	chopper	55
text/html	97.9%	b66b412c448f20d420ec83bb326b9f34	46/62	scrinject	92
text/plain	97.2%	a509796ddb1b0aa5e07f95b52dba7696	23/62	smallasp	22
app/java-archive	92.8%	b3ee9807df3e040d764a3bc795e01d5b	22/64	webshell	985
app/json	91.6%	1c3f8cd54127e869d4879a9b7499da3a	21/62	coinminer	1100
app/x-sharedlib	89.8%	2506f9f2800c20da206776b0d43a5946	33/65	lotoor	4505
app/javascript	85.7%	bca61a7bf8679fb23824f1c6c2d1a43c	21/62	scrinject	590
app/gzip	85.1%	f0bea558da34ef8bd9561cb16caded27	31/62	dloadr	782

Table I: Top ten most common file types (MIME) and their likelihood of inclusion inside benign APKs, associated with a malicious sample of the same type.

samples met our criteria, we selected the smallest file for each MIME type to keep the payload footprint minimal.

Invisible Injection. To satisfy the third constraint, we identified locations within the APK that are typically overlooked by feature extractors. We analyzed state-of-the-art extractors summarized in Table IX, focusing on Drebin, MaMaDroid, and MalScan (see Sect. II). Our analysis reveals that these malware feature extractors consistently ignore the content of the *res* folder. Conversely, we exclude `AndroidManifest.xml` and DEX files, as they contain the permissions and bytecode analyzed by nearly all established feature extraction algorithms. By satisfying these constraints with the 10 malicious samples highlighted in Table I, we demonstrate that attackers can use Apktool to unpack a benign APK, inject the malicious files into the *res* folder, and repack it. This confirms the feasibility of **RQ1**.

B. Poisoning Attacks against ML Malware Detectors

We leverage label spoofing to execute poisoning attacks, which are formalized as an optimization problem:

$$\tilde{D} \in \max_{D} \mathcal{L}(D_{ts}, f^\star) \quad s.t. \quad f^\star \in \min_{f \in \mathbf{H}} L(D_{tr} \cup D) \qquad (1)$$

where $\mathbf{H}$ denotes the hypothesis space of classifiers f; $\mathcal{L}$ and L are loss functions quantifying errors at test and training time, respectively; D_{tr} and D_{ts} are the training and test sets; $\tilde{D}$ is the set of poisoned data; and $f^\star$ is the model trained on the union of regular and poisoned data. When D_{ts} represents the entire test set, Eq. 1 describes an availability or *Denial of Service* (DoS) attack. In our realistic threat model, the attacker uses label spoofing to flip labels from benign to malicious by repackaging the application content. This scenario differs from traditional clean-label or label-flip settings because the attacker does not assign the label directly; rather, they exploit the AV's internal logic to generate the desired label, leveraging the implicit trust practitioners place in these systems.

Maiming the Reputation of Legitimate APKs. The formulation in Eq. 2 can also represent targeted misclassifications:

$$\tilde{D} \in \max_{D} \mathcal{L}(\mathbf{x}_t, f^\star) \quad s.t. \quad f^\star \in \min_{f \in \mathbf{H}} L(D_{tr} \cup D) \qquad (2)$$

In this case, the attacker maximizes the error for a specific target sample $\mathbf{x}_t$ (a benign APK). The resulting model maintains general performance but misclassifies the chosen point. To trigger this specific misclassification, we use clones of the targeted APK, a strategy inspired by [27]. We augment label spoofing by injecting API calls into the APK bytecode, guarded by opaque predicates (logical statements that are always true or false). This ensures that the original functionality remains intact while static feature extractors are tricked into processing the injected calls. By distributing these clones and forcing AVs to label them as malicious, we populate the feature space with samples that surround the target benign APK, forcing a misprediction at test time *only* for that specific application. As shown in Sect. IV-F, in some configurations, only five clones are required to successfully flip a detection.

IV. Experiments

We now conduct an extensive experimental analysis to demonstrate the feasibility of label spoofing (Sect. IV-B) and its impact on ML models in both untargeted and targeted scenarios (Sect. IV-E and Sect. IV-F). Finally, we show that existing countermeasures may fall short, not only in detecting these attacks but also in maintaining reliable predictive performance when deployed (Sect. IV-G).

A. Experiment Setup

Dataset. We replicate the dataset proposed by Ruggia et al. [2], which consists of malicious and benign APKs collected between September 2022 and April 2023. Due to legal restrictions, the authors provided only the APK hashes; we subsequently downloaded the corresponding files from VirusTotal to ensure our data source remained consistent with the original study. Our collection includes a *Malware Dataset* of 20,329 samples across 196 families (ranging from 100 to 114 samples per family) and a *Goodware Dataset* of 20,769 benign APKs spanning 50 Google Play Store categories.We conduct our experiments across two settings: *(i)* a *large-scale* experiment using a dataset $\mathcal{D}_l$ of 40K samples, equally partitioned between the *Malware* and *Goodware* datasets; and *(ii)* a *small-scale* experiment using a dataset $\mathcal{D}_s$ of 1K samples. The latter comprises 500 benign and 500 malicious APKs, sampled from $\mathcal{D}_l$ to include 10 benign samples from each Google Play Store category and 2 to 3 malicious samples from each family. In both settings, we maintain a balanced class ratio to minimize the impact of class imbalance on classification performance, thereby highlighting the specific accuracy degradation caused by label-spoofed training samples.

Feature Extractors and Model Architectures. We evaluate three prominent feature extractors, Drebin, MaMaDroid, and MalScan [8], [16], [17], paired with various ML-based malware classification models. Following established literature [3], [8], [28], we pair the Drebin feature extractor with a Linear Support Vector Machine (LSVM) and Gradient Boosted Trees (GBT). We set the LSVM regularization parameter to $\lambda = 1$, while the GBT is configured with 300 trees for $\mathcal{D}_s$ and 3,000 trees for $\mathcal{D}_l$. Additionally, we implement a Neural Network (NN) featuring a 5-layer feed-forward architecture (with 24000, 240, 120, 60, and 1 neurons, respectively) and Sigmoid activation functions to generate a final probability of malice. To manage the high dimensionality of Drebin feature vectors, which reach 732,800 features in our dataset, we utilize

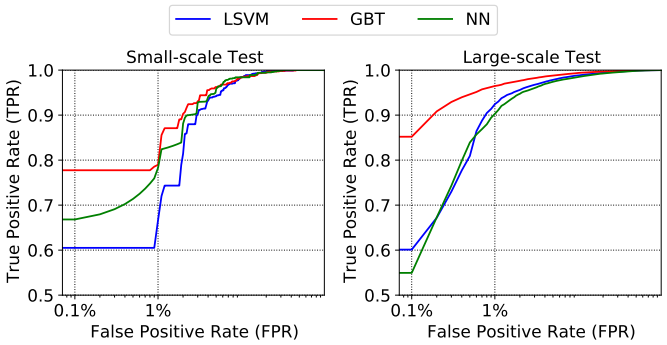

Figure 2: ROC curves of three classifiers (LSVM, GBT, NN) using Drebin features on the small-scale and large-scale test.

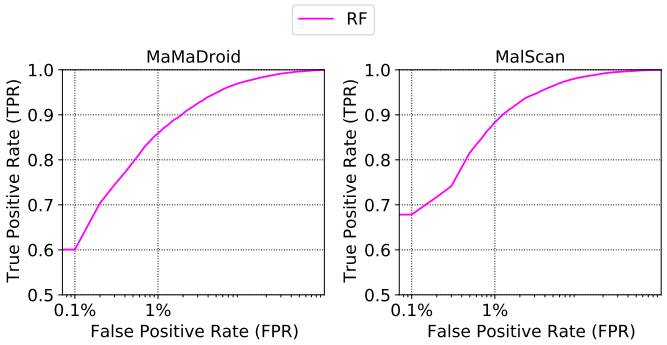

Figure 3: ROC curves of RF using MaMaDroid and MalScan features on the large-scale test.

Random Projections [29] to reduce the input dimensionality to 24,000 before feeding it into the NN. The NN is trained for 5,000 epochs using the Adam optimizer [30] to minimize cross-entropy loss. Similarly, we evaluated MaMaDroid and MalScan with several models suggested in prior work [16], [17]. However, based on empirical performance, we ultimately selected a Random Forest (RF) architecture with 300 trees for these extractors. We apply Drebin features exclusively in the small-scale experiment and utilize all three feature types in the large-scale experiment. For both $\mathcal{D}_s$ and $\mathcal{D}_l$, the data is split into 80% training and 20% test sets. To account for training uncertainty, we repeat the train-test split process 10 times. We report the mean and standard deviation of the resulting metrics, visualizing the Receiver Operating Characteristic (ROC) curves for Drebin-based models in Fig. 2, and for MaMaDroid and MalScan in Fig. 3.

Performance Metrics. To evaluate the performance of our classifiers, we report two primary metrics: *(i)* the False Positive Rate (FPR), which quantifies the volume of false alarms generated; and *(ii)* the True Positive Rate (TPR), which measures the proportion of correctly identified malware samples. For each classifier, we calculate the classification thresholds required to maintain fixed FPR levels of 0.1% and 1%. Finally, we utilize the Attack Success Rate (ASR) to quantify the efficacy of integrity attacks, defined as the frequency with which the attack successfully triggers a misclassification of the targeted benign APK.

Countermeasure. We also consider a scenario in which the ML service provider adopts defense mechanisms to mitigate potential poisoning noise within the training data. Specifically, we evaluate the Deep k-NN defense proposed by Peri et al. [31], which is designed to identify mislabeled training samples. This method performs anomaly detection by examining the labels of the k-nearest neighbors [32] for each sample in the feature space. We demonstrate the effectiveness of this countermeasure against both Denial of Service and targeted misclassification attacks in Sect. IV-G.

B. Feasibility of Label Spoofing Attacks

We demonstrate the feasibility of label spoofing attacks by utilizing the malicious samples selected in Sect. III and investigating: *(i)* whether their injection via repackaging is viable; *(ii)* the potential side effects of such perturbations; and *(iii)* whether AV engines are effectively coerced into erroneous misclassifications.

Injection through Repackaging. Inserting a file into an APK requires a repackaging process, as described in Sect. II. This process may fail due to limitations in the Apktool library or defensive mechanisms implemented within the APK to prevent tampering [33]. Therefore, we first assessed the success rate of repackaging by applying Apktool to all samples in our *Goodware Dataset*. Our results show that 97.7% (20,284/20,769) of the benign APKs were successfully repackaged without errors. We refer to this subset as the *Repack-able Goodware Dataset*.

To comply with Android OS requirements and ensure the APKs remain installable, we digitally signed each sample using a custom-generated certificate. To maintain ethical standards and prevent misuse, we included metadata within the certificate explicitly stating it was generated for scientific research purposes.

Side Effects of the Injection. We next quantified the side effects resulting from the injection process. To do so, we constructed a *Spoofed Dataset* by repackaging each APK in the *Repack-able Goodware Dataset* with a randomly selected malicious sample from those listed in Table I. As specified in Sect. III, the malicious file is inserted into the `res` folder before the APK is repackaged.

We measured the size variation of these repackaged APKs relative to their original versions to determine if the process results in noticeably larger archives. Our analysis indicates that the impact on file size is negligible; on average, the APK size increased by only 2–3%, with a median increase of approximately 1.4%.

C. Influence on the Decisions of AVs.

In this section, we designed two experiments to balance realism with ethical responsibility. The first experiment utilizes VirusTotal to demonstrate the feasibility of label spoofing on a real-world AV aggregation platform. However, since the repeated submission of synthetic malware can pollute publicly

scraped datasets, we limited the number of VirusTotal uploads. To evaluate the attack mechanism at scale without further contamination, we designed a second experiment that replicates VirusTotal's behavior within a controlled environment using multiple commercial Android AV products. This dual-pronged approach allows for a large-scale study of label spoofing while avoiding additional public data pollution.

First Experiment. We randomly sampled 160 APKs from the *Spoofed Dataset* and uploaded them to VirusTotal. While this sample size is relatively small compared to our primary datasets, and despite the fact that uploading synthetic malware for research is generally tolerated [5], [34]–[36], we deliberately restricted the volume of submissions to minimize the risk of contaminating public datasets. For transparency, we added a comment to each VirusTotal submission page stating that the APK was synthetic malware generated for research purposes. All 160 APKs were flagged as malicious, with an average detection count of 13 ± 2 out of the 62 AV engines available at the time of submission (a detection rate of $21 \pm 2.9\%$). This detection range falls within the stable thresholds identified by Zhu et al. [5], which are largely insensitive to short-lived label fluctuations and engine-level noise, thereby indicating a high-confidence malicious classification. Furthermore, applying AVClass [26] to the reports consistently attributed the samples to the same malware family as the injected malicious component.

To confirm that these detections were triggered by the injected payloads rather than the repackaging process itself, we conducted a second control experiment. We submitted the entire set of 20,284 APKs from the *Repack-able Goodware Dataset* to VirusTotal. In this case, *none* of the samples was flagged. This serves as a negative control, validating that repackaging alone does not induce false positives and that the previous detections are exclusively attributable to the injected malicious artifacts. Finally, as an ethical precaution, we requested the deletion of our 160 synthetic samples from VirusTotal; our request was promptly accepted.

Second Experiment. To avoid further VirusTotal pollution, we sought to replicate its behavior in a local environment. However, fully replicating VirusTotal is infeasible, as it relies on proprietary, cloud-based AV engines that cannot be deployed locally. Consequently, we selected the 14 Android AVs evaluated in July 2025 by AV-Test [37], representing the most established solutions on the market. We prepared 14 Android emulators on a physical server equipped with four Intel Xeon Platinum 8160 CPUs (2.10 GHz, 96 cores total) and 512 GB of RAM. Each emulator was configured with one of the 14 AVs and its corresponding signature database. Subsequently, the server was disconnected from the Internet to prevent the emulators from communicating with vendor servers, ensuring no samples were inadvertently uploaded.

We observed that real-time protection functioned in only 9 out of the 14 AVs without Internet access. This limitation is well-documented and motivates the use of on-device ML models [38], [39]. Using a known Joker malware sample, we confirmed that upon installation, the AV issues a warning notification that can be programmatically retrieved via `dumpsys` [40] through the Android Debug Bridge (ADB).

In this setup, we successfully installed 19,067 out of 20,284 (93%) APKs from the *Spoofed Dataset* across the 9 functioning emulators. The primary cause of installation failure was architecture incompatibility (ARM-only APKs on x86 emulators). Notably, *all* successfully installed APKs were flagged as malicious by at least one AV. However, the average detection rate was lower than in the VirusTotal experiment ($2 \pm 0.02\%$). This reduction is expected, as mobile AV versions typically lack the full signature sets available to cloud-based aggregators, likely due to local storage constraints, and often rely on remote lookups that were disabled in our experiment to prevent polluting the companies' datasets.

Despite these environmental differences, the results remain representative of realistic deployment conditions and support **RQ2** by demonstrating that label spoofing attacks are both feasible and effective in practical scenarios.

D. Case Studies

Case Study 1. We consider a benign APK belonging to the Games category, originally undetected by all AV engines on VirusTotal. We injected the 55-byte `image/gif` malware sample of the known Chopper (see Table I) into the `res/drawable` directory. After repackaging and resigning, the APK was flagged as malicious by 22/62 AVs on VirusTotal. AVClass consistently attributed the detection to the Chopper family. Re-submitting the same APK without the injected file resulted in zero detections, confirming that the alert was exclusively caused by the injected artifact.

Case Study 2. We injected the malicious 92-byte `text/html` file into `assets/www/index.html`, a path commonly used by Apache Cordova hybrid apps, corresponding to the Scrinject malware family, into an otherwise benign productivity APK. On VirusTotal, 22/62 AVs flagged the modified APK as malicious, while it did not flag the same APK without the injected file. Interestingly, when evaluating the APK in an on-device/emulator setting, 3 out of 9 AV products detected the sample, although none of their corresponding engines on VirusTotal had flagged it in the previous experiment. These results highlight that identical injected artifacts can trigger heterogeneous detection behaviors across AV engines and deployment settings, yet still consistently succeed in inducing malicious labels.

E. Results of DoS Attacks

We now show how AndroVenom can induce Denial of Service (DoS) against ML models trained spoofed data on both $\mathcal{D}_s$ and $\mathcal{D}_l$. For each train-test split, we randomly select a percentage $p = 1\%, 2\%, 5\%, 10\%, 20\%$ of the benign APKs in the training set to produce misclassifications, amounting to 8, 16, 40, 80, 160 APKs in $\mathcal{D}_s$, and 328, 657, 1643, 3287, 6575 APKs in $\mathcal{D}_l$. Since we have already proved the feasibility of label spoofing through repackaging, and since we do not want to pollute the data sources of VirusTotal, we will *simulate* our attacks, without sending any more samples to be analyzed.

Effect on Small-scale Dataset. We show the efficacy of DoS against Drebin-based models in Fig. 4, highlighting a consistent drop in TPR with the increasingly larger poisoning ratio for all three classifiers. In particular, as shown in Table II,

This article has been accepted for publication in IEEE Transactions on Information Forensics and Security. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2026.3671128

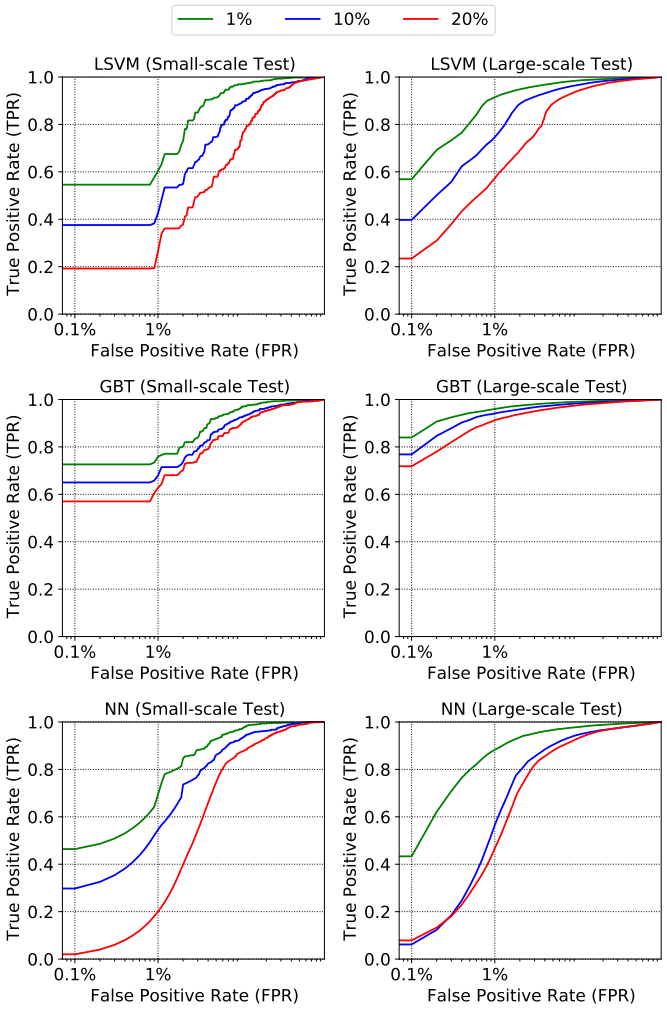

Figure 4: ROC curves of LSVM, GBT, NN using Drebin features on the small-scale and large-scale test with the poisoning ratio 1%, 10%, and 20%.

with only 1% of label-spoofed APKs (8 samples out of 800), the TPR of the poisoned classifier drop by 5.9%, 5.1%, 20.4% for LSVM, GBT, and NN respectively, compared to the poison-free baseline when FPR is 0.1%. We perform Mann-Whitney U tests to compare TPR values at consecutive poisoning ratio levels—such as 0% vs. 1% and 1% vs. 2%, for FPR thresholds of 0.1% and 1%. In most cases, the test rejects the null hypothesis ($p < 0.05$), indicating a statistically significant drop in TPR as the poisoning ratio is increased. Among the three classifiers, NN is the most vulnerable to the DoS attack, with a drop of TPR below 50% at 0.1% FPR for all attacks that we evaluate in the experiments.

Although LSVM preserves relatively more classification accuracy when the poisoning ratio is low, its TPR still has a large decrease when the poisoning ratio rises to 5%, (40 samples out of 800). Among all the three ML models, GBT is the most robust classifier against DoS attacks, with only

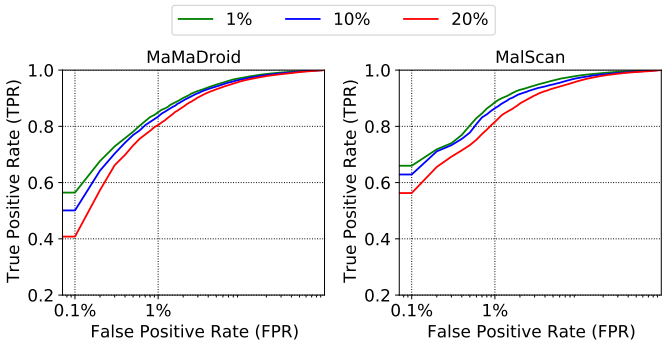

Figure 5: ROC curves of RF using MaMaDroid and MalScan features on the large-scale test with the poisoning ratio 1%, 10%, and 20%.

a 16.3% TPR drop at 1% FPR with 20% of label-spoofed samples. As GBT uses an ensemble of decision trees, each tree minimizes the errors of the previous ones. As a result, GBT reduces the impact of DoS attacks, benefiting from multiple boosted training iterations.

Effect on Realistic Large-scale Dataset. The best malware classification performance of ML models, trained on the three feature representations and on the large-scale dataset, are increased, thanks to the abundance of training data. However, we observe the same trend highlighted from the small-scale experiment: *TPR of malware classification presents sharp decreases in the presence of attacks*.

We show ROC curves and results at different poisoning ratios against Drebin-based, MaMaDroid-based, and MalScan-based models in Fig. 4, Fig. 5, Table V. In the case of Drebin, when the poisoning ratio is over 2% for NN and 5% for LSVM, TPR drops significantly compared to the poison-free baseline. With 0.1% FPR, the label spoofing attack on 1643, 3287, and 328 benign APKs out of 32K training samples are already sufficient to cause a 10% drop of TPR for LSVM, GBT, and NN respectively. For MaMaDroid features, we observe a 7% drop of TPR at 0.1% FPR, but unable to reduce further the performance similar to GBT. For MalScan features, TPR drops less than 2% at both 0.1% and 1% FPR while the poisoning ratio is less than 5%, demonstrating its robustness compared to other features. Nevertheless, TPR decreases evidently after 5% poisoning ratio.

We also quantify the effect of poisoning by highlighting the increment of FPRs. To do so, we freeze TPR to 95% and observe the variation of FPR with an increasingly larger poisoning ratio. For Drebin-based classification, as we can see in Table III, FPR of GBT rises from 0.6% to 1.0%, 1.4% and 3.0% respectively with the poisoning ratio as 5%, 10% and 20%. We can observe similar trends for LSVM and NN. The Pearson correlation coefficients between FPR and the poisoning ratios for LSVM, GBT, and NN are all larger than 0.95, with p-values less than 0.05. This suggests a strong proportionality between the increase of the poisoning ratio and the rise of FPR. Besides, we show the FPR table for RF using

Table II: Mean TPR with standard deviation of models trained with Drebin features on $\mathcal{D}_s$ against DoS attacks at different poisoning ratios. We report performances 0.1% and 1% FPR. All numbers are in percentage.

		Poisoning Ratio					
		0	**1**	**2**	**5**	**10**	**20**
LSVM	**FPR=0.1**	60.5 ± 15.1	54.6 ± 21.7	54.3 ± 22.0	47.5 ± 17.3	37.6 ± 15.7	19.2 ± 9.6
	FPR=1	66.6 ± 17.4	60.5 ± 15.7	61.2 ± 18.1	50.5 ± 13.4	42.5 ± 15.3	26.6 ± 15.6
GBT	**FPR=0.1**	77.7 ± 14.9	72.6 ± 10.5	69.7 ± 10.5	71.7 ± 7.9	65.0 ± 10.6	57.0 ± 11.5
	FPR=1	79.0 ± 14.6	75.9 ± 7.4	74.9 ± 7.7	76.5 ± 8.0	67.9 ± 10.9	62.7 ± 13.8
NN	**FPR=0.1**	66.8 ± 25.4	46.4 ± 28.3	32.3 ± 27.8	33.5 ± 35.8	29.8 ± 32.0	2.0 ± 0.9
	FPR=1	78.3 ± 12.9	69.5 ± 16.9	69.2 ± 10.7	63.6 ± 23.4	54.6 ± 23.0	20.0 ± 9.2

Table III: Mean FPR of models trained with Drebin features on $\mathcal{D}_l$ against DoS attacks at different poisoning ratios. We report performances at 95% TPR. All numbers are in percentage.

	Poisoning Ratio					
	0	**1**	**2**	**5**	**10**	**20**
LSVM	1.7	2.2	2.4	3.6	6.0	12.0
GBT	0.6	0.7	0.8	1.0	1.4	3.0
NN	2.1	2.8	3.7	6.9	11.1	13.9

Table IV: Mean FPR of RF trained with MaMaDroid and MalScan features on $\mathcal{D}_l$ against DoS attacks at different poisoning ratios. We report performances at 90% TPR. All numbers are in percentage.

	Poisoning Ratio					
	0	**1**	**2**	**5**	**10**	**20**
MaMaDroid	1.9	2.0	2.0	2.1	2.3	2.9
MalScan	1.2	1.2	1.2	1.3	1.6	2.6

MaMaDroid and MalScan features in Table IV by freezing the TPR value to 90%. For MaMaDroid, FPR rises from 1.9% to 2.1%, 2.3% and 2.9% corresponding to the poisoning ratio of 5%, 10% and 20% respectively. For MalScan, FPR rises from 1.2% to 1.3%, 1.6% and 2.6% corresponding to the poisoning ratio of 5%, 10% and 20% respectively. The Pearson correlation coefficients between FPR and the poisoning ratios for RF using MaMaDroid and MalScan features are larger than 0.95, with p-values less than 0.005. This alignment with the attack objectives of AndroVenom underscores the effectiveness of the applied DoS attack methodology.

Final Remarks on DoS Attacks. Given the results we presented, we are able to answer **RQ3**. Regardless of the size over training set, AndroVenom influences consistently ML-based malware classifiers. Security entities may use a small or large collection of file samples to build their ML-based solutions for malware classification. Nevertheless, an attacker may exploit the AV engine-based annotation process and inject an excessively small number of misannotated training samples into the training set to cause drastic utility loss of the learned classifier. This vulnerability existing in the AV-based annotation technique raises a severe trustworthy concern for the current practices of ML-based malware classification.

F. Results of Integrity Attacks

We conduct the integrity label spoofing attack with the large-scale dataset $\mathcal{D}_l$, showing that we can actively misclassify specific benign APKs. As for the DoS large-scale experiments, we emulate the attack to avoid the unethical implications that such activity would cause. In particular, we target the TikTok [41] APK. The poisoning variants are generated within the feature space. Specifically, for Drebin features, each variant of the TikTok APK is obtained by introducing one distinct additional feature. For MaMaDroid and MalScan features, each poisoning variant is constructed by perturbing a unique transition probability by an increment of 0.1. After the train-test-split process, we generate $q = 5, 10, 50, 100, 200, 500, 1000$ unique variants of such benign APK, all labeled as malware. The attack results using Drebin, MaMaDroid, and MalScan features are shown in Table VI and Table VII. For Drebin features, compared to the poison-free baseline, TPR fluctuates by only 1% (1% FPR) and 5% (0.1% FPR) as the number of poisoning variants increases for both LSVM and GBT. Besides, the TPR of NN also fluctuates by only 1% and 7% when FPR is 1% and 0.1% respectively, compared to the baseline. Therefore, the targeted label spoofing attack has a very limited impact on the classification performance of the three classifiers on the whole test set. Among the three classifiers, GBT is still the most robust model for the targeted label spoofing attack, with almost no fluctuation of TPR when FPR is 1% and the number of poisoning variants increases. In terms of the classification accuracy over the targeted goodware, we find that only 5 poisoning variants (0.015% of the training set) are sufficient to misclassify TikTok as malware for all three classifiers at 1% FPR. At the low FPR level (FPR=0.1%), the integrity attack shows more impacts over GBT than LSVM and NN. With 10 and 50 poisoning variants (0.03% and 0.15% of the training set), ASR on GBT can reach 50%, significantly higher than LSVM and NN. One potential explanation is that the targeted attack is in nature a two-task learning problem, i.e., normal classification over non-targeted APKs and misclassification over the targeted APKs simultaneously. Since GBT has a better model capacity than LSVM and NN to describe the adaptive and piecewise non-linear decision boundaries (GBT is more accurate with the FPR level), it hence better learns the two learning tasks, triggering higher TPR and ASR at the same time. Since NN has complex and continuous non-linear decision boundaries, it is sensitive to each poisoning variant.

Table V: Mean TPR with standard deviation of models trained with Drebin, MaMaDroid, and MalScan features on $\mathcal{D}_l$ against DoS attacks at different poisoning ratios. We report performances 0.1% and 1% FPR. All numbers are in percentage.

		Poisoning Ratio					
		0	**1**	**2**	**5**	**10**	**20**
Drebin LSVM	**FPR=0.1**	60.1 ± 5.4	56.9 ± 5.9	55.1 ± 4.3	48.1 ± 8.4	39.7 ± 5.5	23.4 ± 5.0
	FPR=1	92.4 ± 1.1	91.4 ± 1.0	90.7 ± 0.9	86.5 ± 3.3	74.6 ± 2.1	57.3 ± 1.6
Drebin GBT	**FPR=0.1**	85.2 ± 4.3	83.9 ± 6.8	81.9 ± 6.1	78.1 ± 6.5	76.8 ± 6.4	71.8 ± 8.2
	FPR=1	96.5 ± 0.3	96.0 ± 0.4	95.7 ± 0.3	95.1 ± 0.4	94.1 ± 0.4	91.3 ± 0.7
Drebin NN	**FPR=0.1**	54.9 ± 13.5	43.3 ± 13.5	29.2 ± 11.1	14.7 ± 6.4	6.2 ± 2.2	7.9 ± 8.8
	FPR=1	90.3 ± 1.0	88.3 ± 1.1	85.7 ± 1.9	77.7 ± 3.8	56.6 ± 14.8	46.7 ± 17.1
MaMaDroid RF	**FPR=0.1**	60.0 ± 9.2	56.4 ± 8.5	56.0 ± 7.1	53.0 ± 9.1	50.1 ± 9.9	40.8 ± 12.0
	FPR=1	85.8 ± 1.2	85.0 ± 1.2	84.9 ± 1.1	84.7 ± 1.2	83.5 ± 1.3	80.5 ± 2.0
MalScan RF	**FPR=0.1**	67.8 ± 3.4	66.0 ± 10.0	66.4 ± 7.5	67.0 ± 5.6	62.8 ± 6.8	56.2 ± 8.4
	FPR=1	88.4 ± 1.7	88.5 ± 1.2	88.2 ± 1.3	87.6 ± 1.5	86.4 ± 1.8	81.6 ± 2.2

As different poisoning variants shift the decision regions differently, ASR of NN fluctuates. Regarding MaMaDroid, TPR of RF varies by 2% (1% FPR) and 10% (0.1% FPR) when the number of poisoning variants increases. Regarding MalScan, TPR of RF varies by 1% (1% FPR) and 5% (0.1% FPR) when the number of poisoning variants increases. When FPR is 1%, we find 5 poisoning variants induce 100% of ASR over the targeted TikTok APK for both MaMaDroid and MalScan features. Similarly, with FPR = 0.1%, only 5 poisoning variants can cause 90% and 80% of ASR for MaMaDroid and MalScan respectively. The results confirm the effectiveness of the integrity attack of AndroVenom over the targeted APK. In conclusion, the integrity attack of AndroVenom can misclassify the targeted goodware as malware with a very tiny ratio of poisoning variants yet keep an accurate classification over the rest of the test samples.

Final Remarks on Integrity Attacks. Given the results, we are able to answer **RQ4**. Regardless of the architecture of the classifiers, AndroVenom can make the targeted goodware misclassified as malware by ML-based malware classification models, while maintaining accurate classification on other testing samples. This observation unveils realistic unethical implications: *Only 5 variants of the benign APK produces excessive false alarms of ML-based malware classification*.

G. Defending against AndroVenom

We illustrate the defensive effect of the Deep KNN (described in Sect. IV-A) defense against AndroVenom on $\mathcal{D}_l$. We choose the number of nearest neighbors k as 3 and 11, which were empirically the best hyper-parameters we checked. We systematically varied the value of k from 1 to 15 in the Deep KNN-based classification method and evaluated the corresponding ASR on the test samples of the target APK. The results showed that ASR remains relatively stable for $k \geq 3$, with the lowest values observed at $k = 3$ and $k = 11$. We report the defense results against the DoS attack on Drebin features in Fig. 6. Also, we present the performance of DoS attacks against MaMaDroid and MalScan in presence of the defense in Fig. 7 and Fig. 8 respectively. Table VIII shows the defense results against the integrity attack on models trained on Drebin, MaMaDroid and MalScan features. For the DoS attack, we randomly select a percentage $p = 10\%$ of benign APKs in the training set, which are used to execute the label spoofing attack. For the integrity attack, we fix the number of poisoning variants q as 100. In the plots, we show ROC curves of the victim classifier compromised by the DoS attack with/without the Deep KNN defense strategy, noted as DoS+DEF and DoS respectively. In addition, we set up a control group to evaluate the impact of the Deep KNN defense strategy over the utility of the poison-free model. They include ROC curves of the poison-free classifier with/without the KNN defense, noted as CLN+DEF and CLN respectively. Regarding the defense results against the integrity attack, we provide TPR values of the victim classifiers with/without the defense, by freezing FPR = 0.1% and 1% respectively. The observation for ML models trained on Drebin features can be summarized in two perspectives. First of all, we find that Deep KNN does not provide consistent defense across different models, by working only on LSVM and NN models. However, GBT models seem to suffer from the inclusion of the defense, experiencing a deterioration of classification accuracy in the DoS attack scenario. Specifically, when FPR is 0.1% and k is 11, TPR of NN increases 20% but TPR of GBT drops by 20%. One possible reason could be that the filtering of training data by Deep KNN also causes a drift inside the training data distribution, which is not ideal for tree-like models like GBT. Second, we also observe that Deep KNN can not prevent integrity attacks of AndroVenom. As demonstrated in Table VIII, there is no defensive effect on LSVM and the ASR of NN even increases with the defense. Although the ASR of GBT decreases when FPR is 0.1%, the defense has no effect when FPR is 1%. As shown by Fig. 7 and Fig. 8, the Deep KNN defense causes significant TPR drop on both the poison-free and the attacked RF model in the DoS attack mode. In the integrity attack mode, with FPR = 0.1% and 100 poisoning variants, the Deep KNN defense can reduce ASR. However, it also causes the TPR to drop significantly. When FPR = 1%, it does not show mitigation effects. This observation denotes that the introduced defense method brings extra harms to the utility of the ML model, yet without offering a strong protection. A possible reason behind this suboptimal result is due to the proximity in the feature space of poisoned

Table VI: Mean TPR with standard deviation of models trained with Drebin features on $\mathcal{D}_l$, and Attack Success Rate (ASR) of the integrity attack targeting the TikTok, with an increasing number of poisoning samples. We report performances at 0.1% and 1% FPR. All values, except poisoning variants, are in percentage.

			Poisoning Variants							
			0	**5**	**10**	**50**	**100**	**200**	**500**	**1000**
LSVM	**FPR=0.1**	**TPR**	57.0 ± 6.6	56.9 ± 6.5	56.7 ± 7.4	56.3 ± 7.8	61.7 ± 4.9	61.0 ± 5.0	57.3 ± 5.3	57.4 ± 5.4
		ASR	0	0	0	0	0	0	0	0
	FPR=1	**TPR**	92.5 ± 1.1	92.1 ± 1.5	92.4 ± 0.8	91.8 ± 1.1	92.2 ± 1.2	92.9 ± 0.8	92.2 ± 1.1	92.0 ± 1.5
		ASR	0	100	100	100	100	100	100	100
GBT	**FPR=0.1**	**TPR**	88.2 ± 2.2	83.4 ± 4.3	87.0 ± 3.6	85.0 ± 4.2	86.8 ± 4.6	87.9 ± 3.6	86.1 ± 4.8	85.8 ± 4.4
		ASR	0	0	30	50	80	90	100	100
	FPR=1	**TPR**	96.6 ± 0.4	96.5 ± 0.2	96.6 ± 0.4	96.5 ± 0.2	96.4 ± 0.5	96.5 ± 0.2	96.6 ± 0.3	96.4 ± 0.4
		ASR	0	100	100	100	100	100	100	100
NN	**FPR=0.1**	**TPR**	51.9 ± 14.7	54.2 ± 13.2	59.6 ± 8.2	44.4 ± 16.2	54.1 ± 10.4	52.9 ± 11.1	49.3 ± 14.0	54.2 ± 10.9
		ASR	0	60	40	20	30	50	60	40
	FPR=1	**TPR**	90.4 ± 1.6	89.9 ± 1.2	90.8 ± 0.8	90.1 ± 1.7	91.4 ± 1.0	90.4 ± 0.9	90.1 ± 1.2	90.7 ± 1.1
		ASR	0	80	70	90	70	100	70	80

Table VII: Mean TPR with standard deviation of RF trained with MaMaDroid and MalScan features on $\mathcal{D}_l$, and Attack Success Rate (ASR) of the integrity attack targeting the TikTok, with an increasing number of poisoning samples. We report performances at 0.1% and 1% FPR. All values, except poisoning variants, are in percentage.

			Poisoning Variants				
			0	**5**	**10**	**50**	**100**
MaMaDroid	**FPR=0.1**	**TPR**	58.9 ± 6.6	55.6 ± 13.4	52.2 ± 9.8	59.4 ± 10.4	55.5 ± 7.0
		ASR	0	90	100	100	100
	FPR=1	**TPR**	85.4 ± 1.6	86.0 ± 0.9	85.6 ± 1.4	85.9 ± 0.9	84.6 ± 1.4
		ASR	0	100	100	100	100
MalScan	**FPR=0.1**	**TPR**	66.8 ± 3.6	64.8 ± 5.3	64.0 ± 11.7	68.0 ± 2.2	62.6 ± 5.5
		ASR	0	80	100	100	100
	FPR=1	**TPR**	88.6 ± 0.9	88.2 ± 1.7	88.5 ± 1.9	89.4 ± 1.0	88.4 ± 1.2
		ASR	0	100	100	100	100

samples and regular ones. The poisoned samples are all close to each other in the feature space, being consistently labeled as malicious. Consequently, most of its nearest samples still carry the attack-desired malicious label, which circumvents the defensive mechanism using Deep KNN.

V. Related Work

Inspired by Cinà et al. [13], we categorize in Table IX the existing poisoning attacks on malware classification into six dimensions: the attacker's goal, the attacker's knowledge, the attacker's manipulation surface, the attacker's capability, the operating system of the dataset, and the feature used for malware classification. *(1)* The attacker's goal can be either a DoS attack or an integrity attack, which is presented in Sect. II. *(2)* The attacker's knowledge can be either a white-box attack or a black-box attack. A white-box attack has complete knowledge about the targeted system, including the training data, the test data, and the targeted model. This attack performs a worst-case scenario. A black-box attack assumes that the attacker has no knowledge of the targeted model. *(3)* The attacker's manipulation surface can be either a feature space manipulation or a problem space manipulation. A feature space manipulation theoretically modifies only the feature space of samples, such as feature vectors. A problem space manipulation makes changes directly on real samples. Hence, the problem space manipulation can test the validity of the feature space manipulation. *(4)* The attacker's capability can be either a clean-label attack, a dirty-label attack, or a label-flip attack, which is introduced in Sect. II. *(5)* The main operating systems of the dataset for malware classification are Android and Windows. *(6)* Since our work focuses on Android malware detection, we only summarize the Android features used in previous works. Apart from Drebin and MaMaDroid feature set presented in Sect. II, some works use only permissions [42] or API call graphs [43] for their feature extractors.

Some works [43]–[52] report an integrity attack, which aims to activate misclassification on specific malware. Besides, the classifier still contributes normal classification behaviors over the other files. With an explanation AI-based technique, [44] can perform better search over important features to poison. Instead, [42], [53]–[56] involve the DoS attack mode. Notably, [56] is established based on Android malware classification use cases. By injecting label-flipping noise into the active learning process, the proposed attack achieves to harm the overall availability of the targeted ML system. Most works of Android malware classification choose Drebin or MaMaDroid as the features. While some previous poisoning attack methods in ML-based malware classifiers concentrate on tampering the feature space, our study focuses on a realistic problem space attack scenario. Specifically, we deliver the attack via injecting malicious files to activate misannotation of benign training samples, yet without compromising the utility of the modified apps or the extracted features.

VI. Conclusions

In this paper, we introduced potential and highly realistic threats originating from the blind trust afforded to Antivirus (AV) engines, which enables dangerous poisoning attacks against Machine Learning (ML) models. We presented AndroVenom, a framework for constructing samples that are consistently labeled as malicious while remaining entirely devoid of harmful functionality. By doing so, we have demonstrated a realistic poisoning attack capable of compromising *any* ML model trained on crowd-sourced AV data.

This is achieved by coercing various AV engines into generating false labels through the injection of specific malicious signatures into benign APKs. Such manipulation not only

Table VIII: Mean TPR with standard deviation of models trained with Drebin, MaMaDroid, and MalScan features on $\mathcal{D}_l$, and Attack Success Rate (ASR) of the integrity attack targeting the TikTok, with/without the Deep KNN defense strategy. We report performances at 0.1% and 1% FPR. All values are in percentage except the number of poisoning variants (q) and nearest neighbors (k).

			q=0			q=100		
			no defense	k=3	k=11	no defense	k=3	k=11
Drebin LSVM	**FPR=0.1**	**TPR**	57.0 ± 6.6	62.9 ± 6.9	60.8 ± 7.7	61.7 ± 4.9	61.3 ± 4.9	64.1 ± 3.8
		ASR	0	0	0	0	0	0
	FPR=1	**TPR**	92.5 ± 1.1	92.3 ± 1.0	89.1 ± 3.1	92.2 ± 1.2	91.7 ± 0.8	90.3 ± 1.4
		ASR	0	0	0	100	100	90
Drebin GBT	**FPR=0.1**	**TPR**	88.2 ± 2.2	85.3 ± 4.7	72.6 ± 7.9	86.8 ± 4.6	78.4 ± 7.5	69.3 ± 10.2
		ASR	0	0	0	80	30	0
	FPR=1	**TPR**	96.6 ± 0.4	95.6 ± 0.4	93.3 ± 0.5	96.4 ± 0.5	95.3 ± 0.4	92.8 ± 0.8
		ASR	0	0	0	100	100	100
Drebin NN	**FPR=0.1**	**TPR**	51.9 ± 14.7	58.8 ± 12.9	57.0 ± 15.0	54.1 ± 10.4	54.1 ± 11.7	57.9 ± 10.0
		ASR	0	0	0	30	60	80
	FPR=1	**TPR**	90.4 ± 1.6	91.0 ± 1.0	90.1 ± 1.0	91.4 ± 1.0	91.1 ± 1.2	90.5 ± 0.7
		ASR	0	0	0	70	100	100
MaMaDroid RF	**FPR=0.1**	**TPR**	58.9 ± 6.6	35.3 ± 10.6	12.1 ± 3.7	55.5 ± 7.0	30.1 ± 10.6	9.2 ± 1.2
		ASR	0	0	0	100	70	0
	FPR=1	**TPR**	85.4 ± 1.6	81.1 ± 1.9	70.1 ± 2.9	84.6 ± 1.4	80.7 ± 2.1	67.9 ± 4.3
		ASR	0	0	0	100	100	100
MalScan RF	**FPR=0.1**	**TPR**	66.8 ± 3.6	18.5 ± 3.6	14.3 ± 3.4	62.6 ± 5.5	15.2 ± 3.0	12.7 ± 2.2
		ASR	0	0	0	100	0	0
	FPR=1	**TPR**	88.6 ± 0.9	85.1 ± 1.4	78.6 ± 2.2	88.4 ± 1.2	83.3 ± 2.5	77.3 ± 3.1
		ASR	0	0	0	100	100	100

Table IX: Poisoning attacks on malware classification categorized by the attacker's goal (**S** = DoS / **I** = Integrity), knowledge (**W** = White-box / **B** = Black-box), manipulation surface (**F** = Feature space / **P** = Problem space), capability (**CL** = Clean-Label / **DL** = Dirty-Label / **LF** = Label-Flip), operating system (**AD** = Android / **WD** = Windows), and feature (**D** = Drebin / **M** = MaMaDroid / **G** = API call graph / **E** = Permission).

WORK	YEAR	GOAL	KNOW	MANI	CAPA	OS	FEAT
Chen et al. [53]	2018	S	W	F	DL	AD	D
Demontis et al. [54]	2019	S	W	F	DL	AD	D
Severi et al. [44]	2021	I	B	P	CL	AD	D
GTA [43]	2021	I	B	P	DL	AD	G
Kuppa et al. [45]	2021	I	B	P	CL	WD	-
Shan et al. [46]	2022	I	B	F	CL	WD	-
Li et al. [47]	2022	I	B	P	DL	AD	D + M
Aryal et al. [55]	2022	S	B	-	LF	WD	-
Jigsaw Puzzle [48]	2023	I	B	P	CL	AD	D
Noppel et al. [49]	2023	I	W	F	DL	AD	D
Tian et al. [50]	2023	I	W + B	P	CL	AD	D
Qi et al. [51]	2023	I	B	F	CL	WD	-
Taheri et al. [42]	2024	S	W	F	DL	AD	E
McFadden et al. [56]	2024	S	B	-	LF	AD	D
Zhan et al. [52]	2025	I	B	P	CL	WD	-

triggers a malicious misclassification but also grants attackers control over the malware family to which these samples are attributed. Our study reveals that these label spoofing attacks can be orchestrated against a wide spectrum of ML models for Android malware classification that rely on static features.

At the time of writing, none of the prominent state-of-the-art ML-based Android malware classification methods include an analysis of the `res` folder within the APK [7], [8], [16], [38], [47], [57], [58]. This omission makes our injection invisible to all current preprocessing and feature extraction algorithms. Furthermore, the sheer volume of crowd-sourced data makes manual verification of AV-based annotations practically infeasible, presenting a significant challenge for mitigation.

Attack Impact and Distribution Risks. We evaluated both Denial-of-Service (DoS) and integrity attacks. While we observed a consistent drop in test-time performance across various models, the efficacy of our integrity attacks is particularly noteworthy: only 5 label-spoofed samples are required to force the misclassification of a specific benign APK. The consequences extend beyond model accuracy. Given the time and cost required to manually verify these false alarms, users may hesitate to download APKs affected by AndroVenom for extended periods. Consequently, the reputation of the targeted APK may suffer, significantly impeding legitimate distribution.

Future Improvements and Limitations. This work demonstrates the feasibility of label spoofing and raise concerns regarding the trustworthiness of AV-based annotation pipelines. As shown in Sect. III, attackers possess significant flexibility regarding the types of malicious files they can inject. This scenario can be made even more stealthy by leveraging diverse malicious patterns to produce multiple file variants. We argue that there is much to explore regarding how these attacks can be further optimized. For instance, while we injected entire files easily recognizable by their hash, future work could identify the minimum indispensable modification required to maintain control over label spoofing and family attribution. Such advanced attack techniques would be highly specific to the targeted operating system and file format.

One limitation of our current attack is the necessity of modifying the APK to inject the malicious file, and this process inevitably invalidates the digital signature. AV companies could potentially mitigate this for a subset of legitimate apps by creating allow-lists based on original digital signatures. Furthermore, even if an attacker modifies an APK without altering its behavior (as with AndroVenom), developers may implement integrity checks within the execution logic to detect tampering. In such cases, the modified app might exhibit

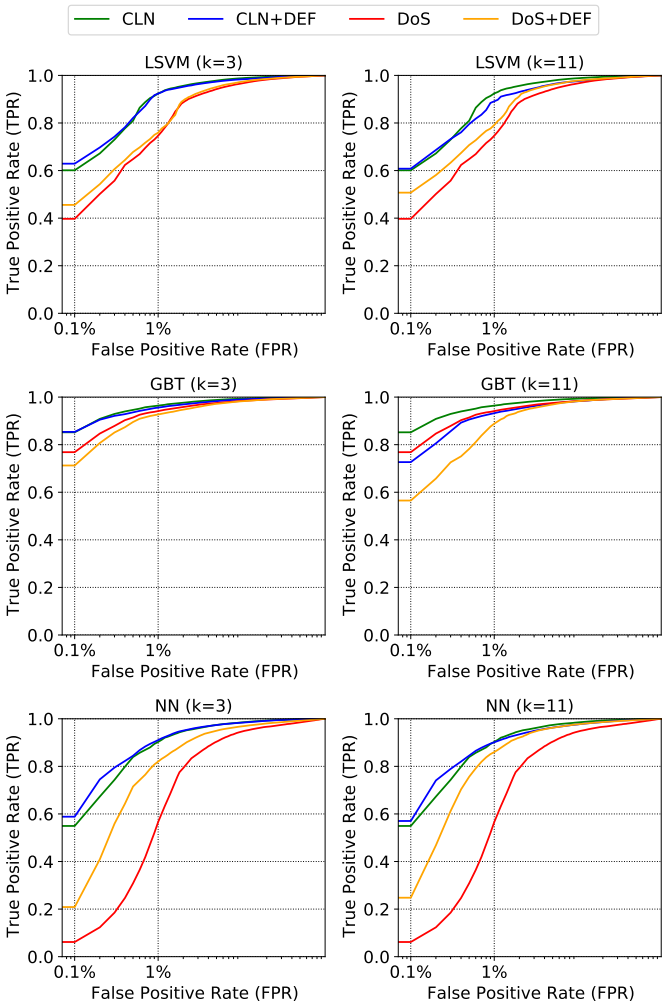

Figure 6: ROC curves of LSVM, GBT, NN using Drebin features with/without the Deep KNN defense strategy (**DEF**) on $\mathcal{D}_l$ in the attack-free (**CLN**) and DoS attack (**DoS**) scenarios. The number of nearest neighbors k is 3 and 11. The poisoning ratio of the DoS attack p is 10%.

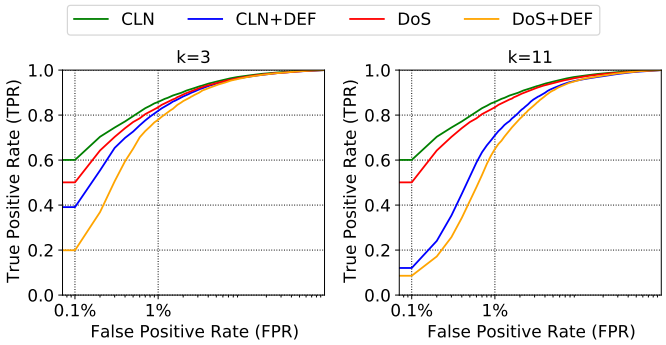

Figure 7: ROC curves of RF using MaMaDroid features with/without the Deep KNN defense strategy (**DEF**) on $\mathcal{D}_l$ in the attack-free (**CLN**) and DoS attack (**DoS**) scenarios. The numbers of nearest neighbors k are 3 and 11. The poisoning ratio of the DoS attack p is 10%.

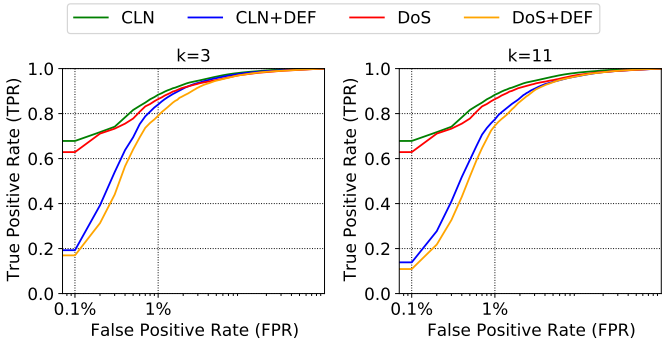

Figure 8: ROC curves of RF using MalScan features with/without the Deep KNN defense strategy (**DEF**) on $\mathcal{D}_l$ in the attack-free (**CLN**) and DoS attack (**DoS**) scenarios. The numbers of nearest neighbors k are 3 and 11. The poisoning ratio of the DoS attack p is 10%.

different feature profiles during dynamic analysis.

Finally, we consciously chose not to emphasize remediation techniques for the following reasons. Although certain ML techniques can suppress label-flipping noise during training [31], [59]–[63], our core message is to highlight the inherent vulnerability of relying blindly on crowd-sourced AV annotations. These ML-based defenses are typically applied during the training process and do not address the root cause: the submission of modified files to AV-based platforms.

The vulnerability in the AV-based annotation process is difficult to curb in practice, as mitigation requires full support from the entire AV vendor community. Our findings suggest that researchers should enrich the feature sets used for classification. For example, integrating resource files into the feature set could dilute or flag the impact of misannotated samples. Future research will explore further countermeasures, such as clustering or ensemble learning-based defenses [59], [64], to identify suspicious samples that share near-identical static features yet are assigned conflicting labels. While these methods help detect outliers, establishing a certifiable, untamperable annotation process remains the ultimate bottleneck.

VII. Acknowledgement

The work of the first author was conducted under the supervision of Prof. Farid Nait-Abdesselam and was primarily supported by a Fellowship Grant from Université Paris Cité. The work was also partially supported by the French National Research Agency (Agence Nationale de la Recherche) via grants “ANR-22-PECY-0007” (DefMal) and “ANR-23-IAS4-0001” (CKRISP).

REFERENCES

[1] Malware statistics. https://www.av-test.org/en/statistics/malware/, Accessed February 28, 2026.

[2] Antonio Ruggia, Dario Nisi, Savino Dambra, Alessio Merlo, Davide Balzarotti, and Simone Aonzo. Unmasking the veiled: A comprehensive analysis of android evasive malware. In *Proceedings of the 2024 ACM Asia conference on Computer and Communications Security (ASIACCS)*, 2024.

[3] Savino Dambra, Yufei Han, Simone Aonzo, Platon Kotzias, Antonino Vitale, Juan Caballero, Davide Balzarotti, and Leyla Bilge. Decoding the secrets of machine learning in malware classification: A deep dive into datasets, feature extraction, and model performance. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 60–74, 2023.

[4] Virustotal. https://www.virustotal.com/, Accessed February 28, 2026.

[5] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. Measuring and modeling the label dynamics of online {Anti-Malware} engines. In *29th USENIX Security Symposium*, pages 2361–2378, 2020.

[6] Virustotal community contributors. https://support.virustotal.com/hc/en-us/articles/115002146809-Contributors, Accessed February 28, 2026.

[7] Antonio Ruggia, Andrea Possemato, Savino Dambra, Alessio Merlo, Simone Aonzo, and Davide Balzarotti. The dark side of native code on android. *ACM Transactions on Privacy and Security, 2025*, 2025.

[8] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. Drebin: Effective and explainable detection of android malware in your pocket. In *NDSS*. The Internet Society, 2014.

[9] Simone Aonzo, Yufei Han, Alessandro Mantovani, and Davide Balzarotti. Humans vs. machines in malware classification. *Proc. of USENIX-23*, 2023.

[10] Joxean Koret and Elias Bachaalany. *The antivirus hacker's handbook*. John Wiley & Sons, 2015.

[11] EICAR. Antimalware test felipe. https://www.eicar.org/download-anti-malware-testfile/, Accessed February 28, 2026.

[12] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2154–2156, 2018.

[13] Antonio Emanuele Cinà, Kathrin Grosse, Ambra Demontis, Sebastiano Vascon, Werner Zellinger, Bernhard A Moser, Alina Oprea, Battista Biggio, Marcello Pelillo, and Fabio Roli. Wild patterns reloaded: A survey of machine learning security against training data poisoning. *ACM Computing Surveys*, 55(13s):1–39, 2023.

[14] OWASP. Owasp machine learning security top ten. https://owasp.org/www-project-machine-learning-security-top-10/, Accessed February 28, 2026.

[15] Operating system market share worldwide. https://gs.statcounter.com/os-market-share/, Accessed February 28, 2026.

[16] Lucky Onwuzurike, Enrico Mariconti, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version). *ACM Transactions on Privacy and Security (TOPS)*, 22(2):1–34, 2019.

[17] Yueming Wu, XiaoDi Li, Deqing Zou, Wei Yang, Xin Zhang, and Hai Jin. Malscan: Fast market-wide mobile malware scanning by social-network centrality analysis. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*, ASE '19, page 139–150, 2020.

[18] Av comparatives. https://www.av-comparatives.org/, Accessed February 28, 2026.

[19] Clamav. https://www.clamav.net/, Accessed February 28, 2026.

[20] Byungho Min, Vijay Varadharajan, Udaya Tupakula, and Michael Hitchens. Antivirus security: naked during updates. *Software: Practice and Experience*, 44(10):1201–1222, 2014.

[21] VirusTotal. Yara. https://virustotal.github.io/yara/, Accessed February 28, 2026.

[22] Nitin Naik, Paul Jenkins, Roger Cooke, Jonathan Gillett, and Yaochu Jin. Evaluating automatically generated yara rules and enhancing their effectiveness. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1146–1153. IEEE, 2020.

[23] Connor Tumbleson. Apktool. https://apktool.org/, Accessed February 28, 2026.

[24] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

[25] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. Avclass: A tool for massive malware labeling. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 230–253. Springer, 2016.

[26] Silvia Sebastián and Juan Caballero. Avclass2: Massive malware tag extraction from av labels. In *Annual Computer Security Applications Conference*, pages 42–53, 2020.

[27] Hojjat Aghakhani, Dongyu Meng, Yu-Xiang Wang, Christopher Kruegel, and Giovanni Vigna. Bullseye polytope: A scalable clean-label poisoning attack with improved transferability. In *2021 IEEE European symposium on security and privacy (EuroS&P)*, pages 159–178. IEEE, 2021.

[28] Hyrum S Anderson and Phil Roth. Ember: an open dataset for training static pe malware machine learning models. *arXiv preprint arXiv:1804.04637*, 2018.

[29] Ping Li, Trevor J. Hastie, and Kenneth W. Church. Very sparse random projections. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 287–296, 2006.

[30] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *3rd International Conference for Learning Representations (ICLR)*, 2017.

[31] Neehar Peri, Neal Gupta, W. Ronny Huang, Liam Fowl, Chen Zhu, Soheil Feizi, Tom Goldstein, and John P. Dickerson. Deep k-nn defense against clean-label data poisoning attacks. In *ECCV Workshops*, page 55–70, 2020.

[32] Belur V Dasarathy. Nearest neighbor (nn) norms: Nn pattern classification techniques. *IEEE Computer Society Tutorial*, 1991.

[33] Sajal Rastogi, Kriti Bhushan, and BB Gupta. Android applications repackaging detection techniques for smartphone devices. *Procedia Computer Science*, 78:26–32, 2016.

[34] Simone Aonzo, Gabriel Claudiu Georgiu, Luca Verderame, and Alessio Merlo. Obfuscapk: An open-source black-box obfuscation tool for android apps. *SoftwareX*, 11:100403, 2020.

[35] Zhewei Huang, Yin Minn Pa Pa, and Katsunari Yoshioka. Exploring the impact of llm assisted malware variants on anti-virus detection. In *2024 IEEE Conference on Dependable and Secure Computing (DSC)*, pages 76–77. IEEE, 2024.

[36] Marcus Botacin. Gpthreats-3: Is automatic malware generation a threat? In *2023 IEEE Security and Privacy Workshops (SPW)*, pages 238–254. IEEE, 2023.

[37] AVTest. The best antivirus software for android: July 2025. https://www.av-test.org/en/antivirus/mobile-devices/android/july-2025/, Accessed February 28, 2026.

[38] Simone Aonzo, Alessio Merlo, Mauro Migliardi, Luca Oneto, and Francesco Palmieri. Low-resource footprint, data-driven malware detection on android. *IEEE Transactions on Sustainable Computing*, 5(2):213–222, 2017.

[39] Tao Peng, Bochao Hu, Junping Liu, Junjie Huang, Zili Zhang, Ruhan He, and Xinrong Hu. A lightweight multi-source fast android malware detection model. *Applied Sciences*, 12(11):5394, 2022.

[40] Google. dumpsys. https://developer.android.com/tools/dumpsys, Accessed February 28, 2026.

[41] Tiktok. https://www.tiktok.com/about?lang=en, Accessed February 28, 2026.

[42] Rahim Taheri, Mohammad Shojafar, Farzad Arabikhan, and Alexander Gegov. Unveiling vulnerabilities in deep learning-based malware detection: Differential privacy driven adversarial attacks. *Computers & Security*, 146:104035, 2024.

[43] Zhaohan Xi, Ren Pang, Shouling Ji, and Ting Wang. Graph backdoor. In *30th USENIX Security Symposium*, pages 1523–1540, 2021.

[44] Giorgio Severi, Jim Meyer, Scott E. Coull, and Alina Oprea. Explanation-guided backdoor poisoning attacks against malware classifiers. In Michael D. Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium*, pages 1487–1504, 2021.

[45] Aditya Kuppa and Nhien-An Le-Khac. Adversarial XAI methods in cybersecurity. *IEEE Trans. Inf. Forensics Secur.*, 16:4924–4938, 2021.

[46] Shawn Shan, Arjun Nitin Bhagoji, Haitao Zheng, and Ben Y. Zhao. Poison forensics: Traceback of data poisoning attacks in neural networks. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium*, pages 3575–3592.

[47] Chaoran Li, Xiao Chen, Derui Wang, Sheng Wen, Muhammad Ejaz Ahmed, Seyit Camtepe, and Yang Xiang. Backdoor attack on machine learning based android malware detectors. *IEEE Trans. Dependable Secur. Comput.*, 19(5):3357–3370, 2022.

[48] Limin Yang, Zhi Chen, Jacopo Cortellazzi, Feargus Pendlebury, Kevin Tu, Fabio Pierazzi, Lorenzo Cavallaro, and Gang Wang. Jigsaw puzzle: Selective backdoor attack to subvert malware classifiers. In *44th IEEE Symposium on Security and Privacy*, pages 719–736. IEEE, 2023.

[49] Maximilian Noppel, Lukas Peter, and Christian Wressnegger. Disguising attacks with explanation-aware backdoors. In *44th IEEE Symposium on Security and Privacy*, pages 664–681. IEEE, 2023.

[50] Jianwen Tian, Kefan Qiu, Debin Gao, Zhi Wang, Xiaohui Kuang, and Gang Zhao. Sparsity brings vulnerabilities: Exploring new metrics in backdoor attacks. In *32nd USENIX Security Symposium*, pages 2689–2706, 2023.

[51] Xiangyu Qi, Tinghao Xie, Jiachen T. Wang, Tong Wu, Saeed Mahloujifar, and Prateek Mittal. Towards A proactive ML approach for detecting backdoor poison samples. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium*, pages 1685–1702, 2023.

[52] Dazhi Zhan, Kun Xu, Xin Liu, Tong Han, Zhisong Pan, and Shize Guo. Practical clean-label backdoor attack against static malware detection. *Computers & Security*, 150:104280, 2025.

[53] Sen Chen, Minhui Xue, Lingling Fan, Shuang Hao, Lihua Xu, Haojin Zhu, and Bo Li. Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. *COSE*, 73:326–344, 2018.

[54] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *28th USENIX Security Symposium*, pages 321–338, 2019.

[55] Kshitiz Aryal, Maanak Gupta, and Mahmoud Abdelsalam. Analysis of label-flip poisoning attack on machine learning based malware detector. In *IEEE International Conference on Big Data, Big Data 2022*, pages 4236–4245, 2022.

[56] Shae McFadden, Mark Kan, Lorenzo Cavallaro, and Fabio Pierazzi. The impact of active learning on availability data poisoning for android malware classifiers. In *Proceedings of the Annual Computer Security Applications Conference Workshops (ACSAC Workshops)*, 2024.

[57] Seungho Jeon and Jongsub Moon. Malware-detection method with a convolutional recurrent neural network using opcode sequences. *Information Sciences*, 535:1–15, 2020.

[58] Jerry Cheng, Starsky H.Y. Wong, Hao Yang, and Songwu Lu. Smartsiren: virus detection and alert for smartphones. In *Proceedings of the 5th International Conference on Mobile Systems, Applications and Services*, MobiSys '07, page 258–271, 2007.

[59] Alexander Levine and Soheil Feizi. Deep partition aggregation: Provable defense against general poisoning attacks. *arXiv preprint arXiv:2006.14768*, 2021.

[60] Wenxiao Wang, Alexander J Levine, and Soheil Feizi. Improved certified defenses against data poisoning with (Deterministic) finite aggregation. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 22769–22783. PMLR, 17–23 Jul 2022.

[61] Ruoxin Chen, Zenan Li, Jie Li, Junchi Yan, and Chentao Wu. On collective robustness of bagging against data poisoning. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 3299–3319. PMLR, 17–23 Jul 2022.

[62] Keivan Rezaei, Kiarash Banihashem, Atoosa Malemir Chegini, and Soheil Feizi. Run-off election: Improved provable defense against data poisoning attacks. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 29030–29050. PMLR, 2023.

[63] Chulin Xie, Yunhui Long, Pin-Yu Chen, Qinbin Li, Sanmi Koyejo, and Bo Li. Unraveling the connections between privacy and certified robustness in federated learning against poisoning attacks. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 1511–1525, 2023.

[64] Jinyuan Jia, Xiaoyu Cao, and Neil Zhenqiang Gong. Intrinsic certified robustness of bagging against data poisoning attacks. In *AAAI Conference on Artificial Intelligence*, 2020.