MalWhiteout: Reducing Label Errors in Android Malware Detection

Liu Wang[*]
School of Cyber Science and Engineering, Huazhong University of Science and Technology
Wuhan, China
w_liu@bupt.edu.cn

Haoyu Wang[†]
School of Cyber Science and Engineering, Huazhong University of Science and Technology
Wuhan, China
haoyuwang@hust.edu.cn

Xiapu Luo
The Hong Kong Polytechnic University
Hong Kong, China
csxluo@comp.polyu.edu.hk

Yulei Sui
University of Technology Sydney
Sydney, Australia
yulei.sui@uts.edu.au

ABSTRACT

Machine learning based Android malware detection has attracted a great deal of research work in recent years. A reliable malware dataset is critical to evaluate the effectiveness of malware detection approaches. Unfortunately, existing malware datasets used in our community are mainly labelled by leveraging existing anti-virus services (i.e., VirusTotal), which are prone to mislabelling. This, however, would lead to the inaccurate evaluation of the malware detection techniques. Removing label noises from Android malware datasets can be quite challenging, especially at a large data scale. To address this problem, we propose an effective approach called MalWhiteout to reduce label errors in Android malware datasets. Specifically, we creatively introduce Confident Learning (CL), an advanced noise estimation approach, to the domain of Android malware detection. To combat false positives introduced by CL, we incorporate the idea of ensemble learning and inter-app relation to achieve a more robust capability in noise detection. We evaluate MalWhiteout on a curated large-scale and reliable benchmark dataset. Experimental results show that MalWhiteout is capable of detecting label noises with over 94% accuracy even at a high noise ratio (i.e., 30%) of the dataset. MalWhiteout outperforms the state-of-the-art approach in terms of both effectiveness (8% to 218% improvement) and efficiency (70 to 249 times faster) across different settings. By reducing label noises, we show that the performance of existing malware detection approaches can be improved.

CCS CONCEPTS

• **Software and its engineering → Development frameworks and environments**.

[*]Liu Wang (w_liu@bupt.edu.cn) is also affiliated with School of Computer Science, Beijing University of Posts and Telecommunications.
[†]Haoyu Wang is the corresponding author (haoyuwang@hust.edu.cn).

ASE '22, October 10–14, 2022, Rochester, MI, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9475-8/22/10…$15.00
https://doi.org/10.1145/3551349.3560418

KEYWORDS

label noise, Android malware detection, confident learning

ACM Reference Format:
Liu Wang, Haoyu Wang, Xiapu Luo, and Yulei Sui. 2022. MalWhiteout: Reducing Label Errors in Android Malware Detection. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3551349.3560418

1 INTRODUCTION

Android malware continues to grow and poses an increasing threat to users (e.g., privacy leakage, financial losses, etc.). Accordingly, Android malware detection and classification have attracted massive research efforts from our community in recent years. According to statistics [3], more than 10K papers have focused on Android malware detection in the past decade, most of which are based on machine learning techniques. In such work, both model training and evaluation rely on a set of apps (i.e., samples) and their associated labels (i.e., benign or malicious). Typically, the labels can be either automatically flagged by anti-virus services such as VirusTotal [4] or manually flagged by security experts.

However, existing malware labelling methods have their limitations. On the one hand, the results returned by anti-virus services (i.e., VirusTotal) are shown to be unreliable and are prone to mislabelling (see § 2). On the other hand, manually labelling is labour- and time-intensive, requires extensive expert knowledge and is difficult to scale to large datasets. Thus, most existing malware datasets [8, 23, 39, 42] that are widely used by related studies were created using the detection results of VirusTotal. As there are over 60 engines that report the detection results on VirusTotal, researchers usually use ad-hoc methods to select different thresholds (e.g., 1, 5, or 10) of detection engines to label malware samples.

The noisy labels in the dataset, i.e. mislabelled samples, have a considerable impact on machine learning based malware detection research. First, noisy labels in the training set can degrade the performance of malware detection models trained with them, making the models less effective in practice. Second, noisy labels in the test/validation set can lead to misjudgement about the true performance of existing malware detection methods. Given the current state of common malware labelling methods, the noisy label problem is inevitable/inherent in machine learning based malware detection and is difficult to deal with. Moreover, this situation will become more acute with the growing size of the datasets used in the research. Yet, this problem has not been effectively addressed. Many studies still use noisy datasets for model training and testing, which to some extent introduces evaluation bias.

Actually, beyond Android malware, label noise is a common problem in machine learning datasets (e.g., image datasets). There is a plethora of research and techniques to address noisy labels in academia, e.g., noise estimation [11, 12, 28], and robust training from noisy labels [16, 29, 34]. Much work deals with noisy labels by means of noise estimation and data cleaning, i.e., excluding samples whose labels are considered potentially corrupted and training with the cleaned dataset. There is also work focusing on robust training from noisy labels, i.e., instead of explicitly pinpointing which sample is likely mislabelled, it focuses on improving the robustness of the model against noise so that it can be trained well even when noisy labels exist in the training data. For example, recent studies [14, 25] design noise-tolerant loss functions to improve the robustness of learning neural networks under label noise.

Thus, one straightforward way to handle noisy malware labels is to borrow the ideas from the general machine learning community (i.e., remove the noisy image labels). However, based on our preliminary exploration, existing techniques have much room for improvement when migrating them to the malware detection domain. The underlying reason is that apps are fundamentally different from images. Mobile apps have rich semantic information in code, a wide variety of metadata, complex processing methods, multiple file compositions, and many relationships between apps through attributes, which are much more complicated than images. To the best of our knowledge, the only existing work in the research community targeting noise reduction in Android malware detection is Differential Training [47]. It relies on a heuristic to differentiate wrongly-labelled samples and correctly-labelled samples based on their loss values in two deep learning models. The noise estimation is completed by multiple iterations, with each iteration finding outliers and revising their labels. The revised labels would be used for malware detection. However, it requires multiple iterations thus falls short in terms of efficiency and practicality, especially when dealing with large-scale malware datasets (see § 5.5).

To overcome the problem of malware detection with noisy datasets, we propose MalWhiteout, a lightweight yet effective noise detection system that can reduce label errors in machine learning based Android malware detection. We have migrated Confident Learning [28], an advanced noise estimation technique, to the domain of Android malware detection. To mitigate the bias introduced by model itself, we further incorporate the idea of ensemble learning and app-feature based calibration to improve the robustness of MalWhiteout. Based on a large-scale crafted malware dataset, we show that MalWhiteout can achieve an accuracy of over 94% for the dataset with 30% wrong labels. For samples mislabeled by VirusTotal, MalWhiteout can identify most of them. Besides, MalWhiteout outperforms the state-of-the-art approach [47] in terms of both effectiveness (8% to 218% improvement) and efficiency (70 to 249 times faster) across different settings.

In summary, this paper makes the following contributions:

- We perform a large-scale study to show the potential label errors introduced by existing malware labelling method, and observe that VirusTotal's scan results have significant uncertainty for labelling malware. We collected over 870K samples which were uploaded to VirusTotal three times over a long period. Over 30% of the samples were found to suffer from label changes in the rescan after two years.
- We propose an accurate and efficient approach, MalWhiteout, for pinpointing the noisy labels in malware datasets. Our approach creatively introduces Confident Learning (CL) method to detect the noisy labels in Android datasets, and incorporates the idea of ensemble learning and app relation based adjustment to achieve more robust results.
- We perform extensive experiments to evaluate the effectiveness of MalWhiteout. Experimental results show that MalWhiteout is capable of detecting label noise with an accuracy of over 94% and F1 of over 91% at varying noise ratios up to 30%, and can identify most of the mislabels from VirusTotal labeling. Compared to the state-of-art, MalWhiteout achieves much better results (8% to 218% improvement) with significantly shorter time (70 to 249 times faster). We further show that the performance of malware detectors can be improved after removing noise by MalWhiteout.

To facilitate further research, MalWhiteout is released at:

https://github.com/MalTools/MalWhiteout

2 THE UNCERTAINTY OF ANDROID MALWARE LABELLING

2.1 Malware Labelling based on VirusTotal

Due to the prohibitive cost of manually labelling, relying on the scan results of existing anti-virus services (e.g., as collected in VirusTotal) is the most common solution to label malware in the community.

However, a common challenge of using VirusTotal is that different security engines often diverge in their detection results for a given app. In this context, researchers have to work out a strategy to aggregate the results and assign a binary label (i.e., malicious or benign) to the app. Typically, researchers will define a voting threshold, and the scanned app will be considered malicious once the number of positive engines exceeds (or equals to) this threshold. However, *the threshold is mostly set based on the researchers' intuition and experience, without convincing reasons.* Considering that there are over 60 anti-virus engines on VirusTotal, different researchers can take different thresholds. For example, a number of studies [23, 36, 40] used 1 engine as the threshold, TESSERACT [30] set the threshold to 4 for malware labelling, and Wang et al. [38] set the threshold to 10. Wei et al. [43] considered over half of the antivirus products' decisions to be reliable, and used 28 engines (at least 50% of the 55 specific antivirus products) as the threshold to create an Android malware dataset. Previous work [51] surveyed 115 academic papers that used VirusTotal for malware labelling, and found 82 of them clearly stated taking a threshold-based method and using varying thresholds (ranging from 1 to 40) to determine whether a file is malicious or benign. Thus, *there are no standards on how to label apps based on the detection results of various engines.*

Worse still, *the detection results of VirusTotal could change over time, making it more difficult to infer the true label of an app.* Existing work [51] reveals that label flips widely exist across engines in

VirusTotal (i.e., individual engines can flip their labels on a given file in a short period of time), and they do not necessarily disappear even after a year. Hence, such limitations (i.e., variable thresholds and volatility of results) make VirusTotal-based malware labelling method subject to significant uncertainty.

2.2 Preliminary Estimation of the Uncertainty

It is known that VirusTotal and its third-party vendors keep updating their anti-malware engines, so the label of a given sample may change over time. Recent efforts [21, 51] tried to measure the label dynamics of VirusTotal, but they are limited in providing a solid estimation of how unreliable VirusTotal is, and most of the studies are focused on PE files. In this regard, we carried out a straightforward preliminary study to explore the dynamics and uncertainty of VirusTotal's labelling on Android malware.

We had collected over 876K samples from Androzoo [7], which were sourced from Google play and detected as malicious by at least one engine from VirusTotal.[1] The samples were collected in August 2019, at which time we uploaded them to VirusTotal for rescanning. After two years, we uploaded these samples to VirusTotal for rescanning again in August 2021. As such, we obtained three snapshots of the scanning reports, i.e., scan results provided by Androzoo, scan results in 2019.08 and scan results in 2021.08, respectively. Then we compared the results of these three scans. Figure 1 portrays the flow of app labels across multiple scans, where a benign app refers to one that is not flagged by any engine, otherwise is malicious (i.e., set the threshold as 1). For the first two scans, we found that 21% of the samples turned from malicious to completely benign, with none of the engines flagged positive (i.e., returned a "malicious" label). This means that 21% samples had their labels changed (if we use 1 engine as the threshold to label malware). For the last two scans, we found that overall about 64% of the samples gained different scanning results, varying between 1 and 39 engines. In 2019.8, about 78.8% of the samples were flagged positive by at least one engine, and the other 21.2% were not flagged positive by any of the engines. In 2021.8, the two percentages became 65.9% and 34.1% respectively. We can see that if we set the threshold to 1, i.e., label a sample as malicious if at least one engine thinks it is malicious, there are approximately 17% of the samples were given different labels by VirusTotal over the two years (about 2% of the samples originally seen as benign shifted to malicious and about 15% of the samples originally seen as malicious shifted to benign). If we compare the first and third scans, the number of apps whose labels changed is even more (34.1%). Thus, using VirusTotal for malware labelling is prone to generating error/noisy labels.

3 KEY IDEAS OF MALWHITEOUT

In general, our key idea is to borrow existing noise-handling methods to deal with the noise problem in machine learning based Android malware detection. However, our preliminary exploration revealed that while existing methods can be ported to deal with our task, they still present some challenges (idea-1). In response, we come up with two ideas for improvement (idea-2 and idea-3).

[1]AndroZoo is a growing collection of Android apps collected from several sources including the official Google Play market. Each app is analysed by different antivirus products from VirusTotal, and the scanned results are provided.

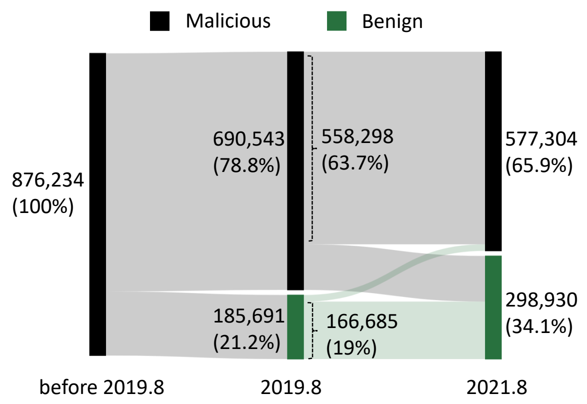

Figure 1: The uncertainty of Android malware labelling.

3.1 Idea-1: The Applicability of Traditional Noise Estimation Techniques

As many noise estimation techniques have been proposed in other fields (e.g., image classification), we first tried such methods to see if they also work well for addressing the noise problem in Android malware detection. Specifically, we picked a few open source efforts (e.g., Confident Learning [28], Co-Teaching [18], Decoupling [24]), ran them on a crafted noisy Android dataset (i.e., the dataset with 10% noise in § 5.1) and verify their effectiveness in detecting label noise. Unfortunately, we observe a number of challenges when migrating these methods. First, although some methods (e.g., Co-Teaching [18], Decoupling [24]) can detect some noises in malware datasets, they can also generate intolerable number of false positives (i.e., identify the correctly labeled samples as noises). These numerous false positives cannot be ignored because if we correct the labels by flipping them, these false positives can become new noises. Second, some methods [19] require a clean dataset for training. However, since the technologies that make up malware are highly sophisticated and constantly evolving, it is challenging for us to ensure the correctness of an Android dataset, and our initial goal is to make MalWhiteout work on any given malware datasets. Third, all the methods utilize features that are based on separate entities, without taking into account that many apps are related to each other (e.g., a malicious developer can release a number of similar malware).

Nevertheless, among them, Confident Learning (CL) [28] seems a viable method, as it performed best during our preliminary exploration (i.e., it gets rid of the highest amount of noises). This inspired us to leverage CL for noise handling in Android malware detection.

Confident Learning for Noise Estimation. Confident Learning (CL) [28] is an emerging framework for characterizing and identifying label errors in datasets and learning with noisy labels. CL works by estimating the joint distribution between the (noisy) observed labels and the (true) latent labels. The central idea of CL to estimate the true labels is that when the predicted probability of a sample is greater than a per-category-threshold, the sample is confidently counted as actually belonging to that category. Specifically, the thresholds for each category are the average predicted probability of samples in that category. This equips CL with robustness to heterogeneous class probability distributions and class-imbalance. To find label errors, CL requires only two inputs: (1) an $N \times K$

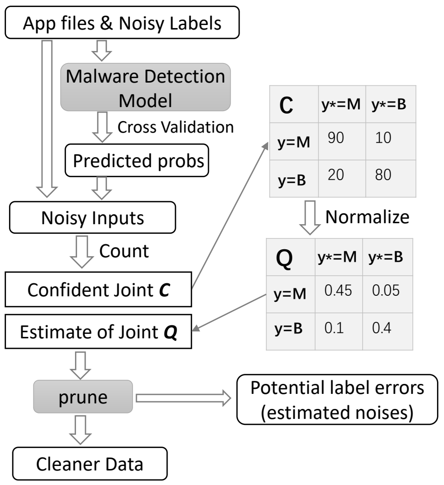

Figure 2: The working process of Confident Learning (CL).

matrix of out-of-sample predicted probabilities for N samples and K categories which is obtained using a model (by cross-validation) beforehand ; (2) an array of noisy/original labels for each sample. As shown in Figure 2, CL counts the examples that are labeled as y and have a high probability of belonging to $y*$, constructing the confident joint C where diagonals capture correct labels and non-diagonals capture asymmetric label error counts. Then C is used to estimate the joint distribution Q, characterizing the noise rates of each case. Following the estimation of C and Q, any rank and prune approach can be used to clean data (e.g., prune by probability ranking and noise rate). CL decouples the model and data cleaning procedure by working with the model's outputs (i.e., predicted probabilities), so that any model that produces predicted probabilities can be used. Moreover, it is non-iterative (thus running fast) and does not rely on any set of true labels that is guaranteed to be uncorrupted. The implemented CL framework can be wrapped with any malware classifier (e.g., Drebin [8], CSBD [6]) to find potential label errors in a noisy dataset.

However, it is not perfect due to the presence of false positives in the results and the neglect of some app-specific features (e.g., app relation). We thus turn our attention to handle these two issues.

3.2 Idea-2: The Power of Multiple Models

CL works with a machine learning based malware detection model to estimate noise, which can be any malware detection model here. In fact, there are quite a lot of malware detection methods in the research community, which use different features and classification models. To prevent the possible bias introduced by the method itself, in our preliminary exploration, we chose two open source malware detection methods, i.e., CSBD [6] and Drebin [8], to verify the noise detection effect of CL. We modified their code and implemented CL's method in an attempt to find label errors on the crafted noisy dataset. Specifically, the ground truth dataset we use consists of 5,794 malware samples and 5,800 benign samples (described in detail in § 5.1), on the basis of which we randomly select 10% of samples whose labels are flipped and thus become noises.

Figure 3 shows the noise distribution for the three cases of true noises, noises detected by the CL-wrapped CSBD model, and noises detected by the CL-wrapped Drebin model. We gave each app a

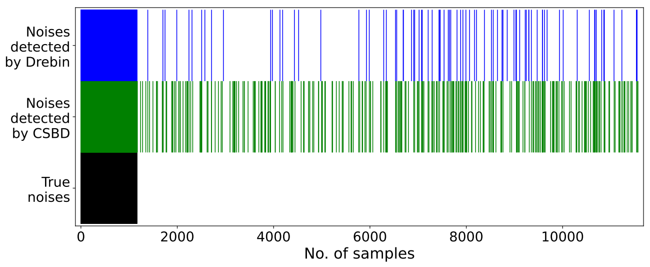

Figure 3: Noise detection results based on different models.

number and ranked the noise samples first. So each app is a data point on the x-axis shown in solid colors when identified as noise. We can observe that both models can successfully detect many true noises (dense vertical lines on the left), but also have some false positives (sparse vertical lines on the right). Overall the Drebin model performs better than the CSBD model with CL. A closer look reveals that the false positives identified by the two models are usually inconsistent (the blue and green lines rarely overlap on a straight line). This suggests that the samples in which the decisions of the two models diverge are prone to be false positive. In contrast, the samples detected as noisy by more than one model are more inclined to be true noises.

For the sake of statistical evidence, we calculate the percentage of true noises in the two scenarios, i.e., in the noises predicted by two models and in the noises predicted by only one of the models. As a result, of the 989 samples identified as noises by the two models, 969 samples (98%) were actually true noises, compared to 170 of the 500 samples (34%) identified as noises by only one of the models. We suspect that this may arise from the bias introduced by the method itself which is different for each method. Thus, each method is able to flag many true noises, while the generated false positives vary widely. This suggests that relying on multiple models for noise detection may be more effective than a single model.

3.3 Idea-3: App Specific Features are Helpful

As aforementioned, the CL models use the semantic features of the individual apps like the features of other entities. Some app-specific features, i.e., the relations between apps, are ignored. As is well known, each app has a wealth of attributes (e.g. developer, signature) that enable association between apps. Such association sometimes can play an important role in mobile analysis, such as deciding the app's label. For example, if two apps have the same private signature (i.e., from the same developer), their labels are likely to be the same, as a malicious developer/group often publishes more than one malicious app. Previous work [31] made use of Indicators of compromise (IoCs)[2] in mobile markets to identify malicious developer accounts and potential unreported malware. They revealed that the connections between app IoCs can be leveraged for attribution inference, and the developer identifiers from app markets (e.g., developer ID, developer name, company name) can be used to source the app.[3]

[2]In the field of cyber-security, an Indicator of compromise (IoC) is an object or activity observed on a network or device that indicates a high probability of unauthorised access to the system. Such indicators are used to detect malicious activity at an early stage as well as to prevent known threats.

[3]Note that, such information can be parameterized by the market, and is not guaranteed to be collected for every app.

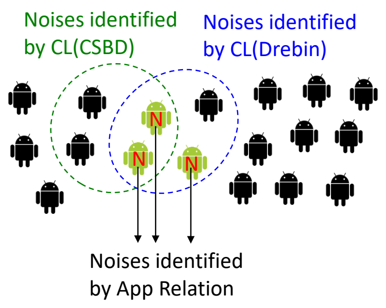
Noises identified by CL(CSBD)
Noises identified by CL(Drebin)
N
N
N
Noises identified by App Relation

Figure 4: An example of app relation based calibration.

In view of this, we carried out a simple preliminary investigation to see if such app relation could help us fine-tune noisy labels. Given an app, suppose we infer its true label based on the labels of the majority of apps that share the same developer with it, and decide whether it is a noise based on the difference between its original label and the inferred label. By this means, on our noisy dataset, we can successfully identify many true noises and very few false positives. The unidentified noises were mostly due to the fact that that there were no apps in the dataset sharing the same developer with the given app. More importantly, if we zoom in to this portion of the samples in which the noise can be identified by app relation (i.e., samples that share developer with at least one other sample), this method yielded better results than CL. As shown in Figure 4, the 18 apps shared the same developer[4], and 3 of them had original labels of benign, which were actually noisy labels. Among them, 4 apps were identified as noises by the CL-wrapped CSBD model, of which 2 were false positives. And 4 apps were identified as noises by the CL-wrapped Drebin model with 1 was false positive. Whereas, the developer-based app relation could correctly identify all the noises, better than both models. This shows that app relation can facilitate noise detection. However, as such relation cannot cover the full range of apps, it might be a good idea to adjust the noise detection results relying on the app relation.

4 THE DESIGN OF MALWHITEOUT

Enlightened by the key ideas, we propose a lightweight yet effective system called MalWhiteout, aiming at reducing label noises for machine learning based Android malware detection. Figure 5 illustrates the overall architecture of MalWhiteout. MalWhiteout is built atop CL, and it further introduces two major components: 1)*ensemble-based noise detection*, which adopts ensemble learning to improve the robustness of noise estimation based on CL; 2) *app relation based adjustment*, which further uses an inter-app relation based on app developers to adjust the noise detection results. Apps and their original labels (which may be incorrect) are fed into the system, where MalWhiteout detects potential label errors in the dataset and revises/flips their labels, leading to a cleaner dataset.

4.1 Ensemble-based Noise Detection

MalWhiteout implements noise detection based on Confident Learning. To address the dependency of CL on malware detection models in our preliminary exploration (see § 3.2), we incorporate the idea of ensemble learning for a more comprehensive and accurate assessment on the label noises. Specifically, we take advantage of Stacking (also known as Stacked Generalization) [1], an ensemble learning method that combines predictions from multiple machine learning algorithms on the same dataset via a meta-learner to output the final prediction. In practice, the architecture of the Stacking method is usually two-level, involving multiple base learners (level-0 learners), and a meta-learner (level-1 learner) that combines the predictions of the base learners. For this purpose, we first need to properly select multiple base learners and a meta-learner.

4.1.1 Learner Selection and Feature Extraction. For base learners, we need to select several Android malware detection approaches. Each sample is transformed into a multi-dimensional numeric feature vector as specified by the corresponding malware detection approach. In general, the criterion for selection is that the malware detection approaches should be "good and different". A good malware detection approach ensures the accuracy of the prediction, and different malware detection approaches help to compensate for the bias of each approach. For this, we have shortlisted three malware detection efforts that are widely used in academia, i.e., Drebin [8], CSBD [6], MalScan [45], all of which report promising results and use very different features and methods for malware detection. Drebin [8] is a lightweight approach to Android malware detection that extracts features (e.g. permissions, Android components, API calls, network addresses, etc.) from the app's code and Manifest file. The features are further fed into a Support Vector Machine (SVM) classifier for malware detection. CSBD [6] constructs a Control Flow Graph (CFG) of the app's bytecode and builds a set of textual features extracted from the CFG. It uses Random Forest for malware detection. MalScan [45] is a detection system that relies on social-network-based centrality analysis of sensitive API calls. It treats function call graphs of apps as social networks and extracts the semantic features of the graphs. These three malware detection approaches serve as base learners in our system.[5] All the three approaches are applied in parallel to the dataset, extracting features that are specific to each approach for each sample. For the meta-learner, it is common to use a simple and linear model to learn how to harness the variety of predictions made. We adopt the Logistic Regression model as it is a linear model that is typically used for binary classification tasks.

4.1.2 Noise Estimation. Figure 6 shows the workflow of ensemble-based noisy label detection. First, each base learner works with its own features and classifier to output predictions, i.e., the predicted probabilities of each sample under each category, through a 5-fold cross-validation. As such, a matrix of size $N \times K$, where N refers to the number of samples and K to the number of categories, is output by each base learner. Next, for each sample, its probabilities predicted by the different base learners are concatenated into a new feature vector, which is fed into the meta-learner along with its original label. Meanwhile, the meta-learner performs noise detection by being wrapped into CL. Finally, it outputs the indices of potential mislabeled samples, i.e. telling us which samples in the dataset are likely to be mislabelled.

[4]signature: 27196e386b875e76adf700e7ea84e4c6eee33dfa

[5]Note that, any other malware detection approaches can be incorporated to MalWhiteout directly, and we only use these three approaches as a case study to show the performance of MalWhiteout.

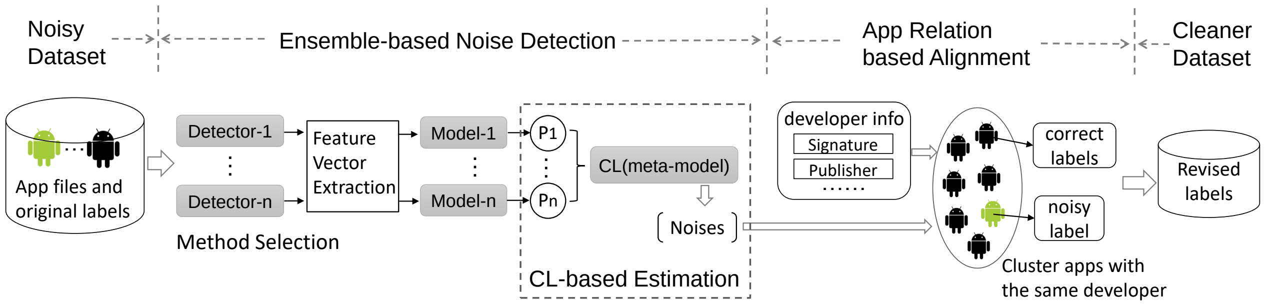

Figure 5: System architecture of MalWhiteout.

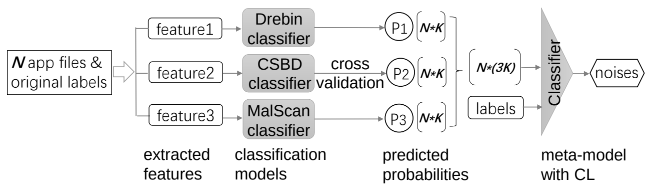

Figure 6: Workflow of ensemble-based noisy label detection.

4.2 App Relation Based Adjustment

Up to now, the aforementioned noise detection approach is essentially a selection based on the uncertain estimation of a single sample, and the identified "error labels" are not guaranteed to be true errors. We then seek to improve the system's correctness by taking into account the apps' relations.

As discussed in § 3.3, we primarily leverage the app's developer information to infer its true label. Besides the developer identifiers collected from app markets, we also carry out the extraction of each app's signature which symbolizes a particular developer. Note that, some malware developers may use the known common keys in the community (such as TestKey, the generic default key for packages with no key specified) to sign apps in order to conceal themselves. For this, we identify those signatures that belong to the generic keys and discard them. After that, we treat apps with the same developer as belonging to a cluster, and determine the true labels of the samples in each cluster by *majority voting* (if there is more than one sample in the cluster). Here, we set the threshold to 2/3 based on some experimental measurements, i.e. if more than 2/3 of the samples in a cluster are malicious (or benign), then all sample labels in this cluster would be presumed to be malicious (or benign). In this way, the few apps in the cluster whose labels are inconsistent with the presumed label will be regarded as noise. Based on this decision, the noise detection results obtained in the previous phase will be adjusted. Note that this phase is a fine-tuning for only the portion of the samples for which it is possible to infer whether they are noisy or not through developer-based app relations. At last, based on the adjusted results, the label of each sample that is identified as a noise will be revised, resulting in a cleaner dataset.

5 EVALUATION

In this section, we conduct experiments to evaluate the performance of MalWhiteout. Our experiments were conducted on a server running Ubuntu 20.04.1 LTS operating system with 6-core Intel(R) Xeon(R) CPU E5-2603 v4 @ 1.70GHz and 64.0GB memory. In particular, we investigate the following research questions (RQs):

- RQ1 How effective is MalWhiteout in reducing label noises?
- RQ2 How important are the design decisions we made for MalWhiteout (i.e., ensemble-based noise detection and app relation based alignment) on the basis of CL?
- RQ3 How much can MalWhiteout improve the performance of existing malware detection efforts?
- RQ4 How well does MalWhiteout perform compared with the state-of-the-art approach?

5.1 Datasets and Metrics

To answer the RQs, we require a credible ground truth dataset including both malware and benign apps. On this basis, we artificially create noisy labels to obtain datasets with varying level of noise.

5.1.1 Malware Dataset. Due to the unreliability of the commonly used malware labeling methods aforementioned (see § 2), some malware datasets that largely rely on the VirusTotal scan results are not used by us, even though they are widely used by the research community (e.g., AMD [42] and Drebin [8]). To the best of our knowledge, the most reliable and trustworthy malware dataset in our community is MalGenome [50], which was created by carefully examining Android related security announcements, threat reports and blog contents from existing mobile antivirus companies and active experts. However, it is old (created in 2011) and has only 1,260 malware samples from 49 families. To expand the dataset, we use a reliable way similar to that of MalGenome to collect malware samples, i.e., relying on the analysis reports of security experts. Usually, the security companies publish security reports to reveal their new identified malware. These reports are manually analyzed by security experts and often list the Indicators of compromise (IoCs), such as the file hash which could be used for us to create malware dataset. Thus, we make efforts to create an Android malware dataset called MalRadar [41], which contains a total of 4,534 samples belonging to 148 malware families. Specifically, we first use a keyword-based method to automatically search and crawl security reports from leading security company websites, obtaining 178 high-quality security reports which were released from 2014 to 2021. Then we extract the IoCs (app hashes) and family labels from the collected reports. At last, we successfully download a total of 4,534 binary files from Koodous [2], one of the most popular platforms that provided mobile malware downloading services. The detailed collection process can be found at the paper [41]. This collected dataset can serve as a ground truth malware dataset, along with MalGenome. As MalGenome was created in 2011 with 1,260 samples ranging from 2010-2011 and MalRadar samples were distributed in 2014-2021, these two datasets contain a wide variety of malicious apps covering a time span of almost a decade. As a result, our ground truth malware set consists of 5,794 malware samples in total. They are distributed in 196 unique families, with the number of samples per family ranging from 1 to 796.

5.1.2 Benign Dataset. In addition, we collect and download benign apps from AndroZoo [7]. To ensure that the benign apps we use are indeed benign, we upload each benign candidate to VirusTotal for rescanning two times (i.e., three snapshots in total). Only samples that are not flagged as positive by any of the engines in all these three snapshots are eligible. To eliminate the bias that may be introduced by an unbalanced dataset, we keep the number of benign apps comparable to the number of malicious apps. As a result, we obtain a total of 5,800 benign apps as the ground truth benign set.

5.1.3 Crafting Label Noises. To examine MalWhiteout's capability on detecting label noise, we manually create some noisy labels in the dataset. The noisy labels come from two ways: (1) We randomly select a certain percentage of the samples i.e., 5%, 10%, 15%, 20%, 25%, 30%, respectively, as "noises" whose labels are manually revised/flipped. (2) We label the samples based on VirusTotal's scan results (using a threshold-based method), and take the mislabeled ones as "noises". We assume that the noise ratio in a dataset used for malware detection is restrained, as typically the samples are selected by the researchers based on certain criteria. Thus we consider that a noise ratio limit of 30% is adequate. Note that, we assume that the noise ratio in a dataset is less than 50%, because if this is not the case it means that the quality of the dataset is even lower than randomly labelled, which is rarely possible in practice. In addition, to prevent possible bias due to the unbalanced distribution of noises in benign and malicious apps, we keep the noise ratio in malware set and benign set at an equal level.

5.1.4 Metrics. The noise detection problem addressed by MalWhiteout is essentially a binary classification task, thus we use the widely used metrics (including precision, recall, accuracy, etc.) to measure the effectiveness of MalWhiteout. Besides, to more directly demonstrate MalWhiteout's ability in reducing noise, we also list the number and percentage of wrongly-labelled samples left after MalWhiteout's process (# of Noise Left and % of Noise Left), as well as the percentage of wrong labels being reduced (% of Noise Reduced). Table 1 presents the metrics used. Note that in all our experiments, we achieve noise reduction by the strategy of revising/flipping labels for each identified noise, in order not to compromise the size of the dataset.[6]

5.2 RQ1: Overall Effectiveness

5.2.1 Detect label noises from random flipping. We first evaluate how well MalWhiteout detects label noises at various noise ratios. To this end, we conduct experiments on noisy datasets containing 5%, 10%, 15%, 20%, 25%, 30% noises respectively. Table 2 presents the detection results of MalWhiteout on datasets at different noise ratios. We can see that after MalWhiteout's processing, the percentage of noisy labels in the dataset is reduced from 5% to 0.43%, from 10% to 0.62%, from 15% to 1.3%, from 20% to 2.04%, from 25% to 3.28%, from 30% to 5.16%. As the noise ratio varies from 5% to 30%, the F1 ranges from 96.8% to 91.3%, the accuracy ranges from 99.6% to 94.8%, and the percentage of noises reduced by MalWhiteout ranges from 93.8% to 82.8%. The results suggest that the effectiveness of MalWhiteout is promising and generally stable at different noise ratios. However, if we compare side-by-side, we can notice that the effect of MalWhiteout seems to decrease gradually as the noise ratio increases from 5% to 30%. The possible reason is that too many wrong labels in the dataset can affect the predictions obtained by each base learner through cross-validation, thus affecting the correctness of noise detection by CL. Additionally, we fed the post-processing results back into MalWhiteout for the 30% noise ratio case, which contains a total of 598 (5.16%) noises, and found that the noise was reduced by almost half again, cutting the noise ratio down to 2.84% (with 330 left). Thus, for datasets with heavy label noise, a second pass can yield a much cleaner dataset. Anyhow, MalWhiteout has good noise detection capability when the noise ratio of the dataset is within 30%.

Considering that obfuscation is one of greatest challenges in malware detection domain, we are interested in whether obfuscation may affect MalWhiteout's ability to properly label errors. Since there exist a number of samples using obfuscation techniques in our collected malware samples, we take an additional check on how MalWhiteout performed on these obfuscated samples. We find that MalWhiteout is still able to identify label noise despite the use of some obfuscation techniques. For example, for the case of 10% noise ratio, we found that 37 noisy samples were actually obfuscated and all of them were correctly identified by MalWhiteout. We argue that this is attributed to the resistance to obfuscation by the three malware detectors selected. Theoretically, MalWhiteout inherits the detection capability of its internal malware detectors, thus the effect of obfuscation on MalWhiteout basically depends on the anti-obfuscation capabilities of the selected detectors. This also reflects the importance of model selection.

5.2.2 Detect label noises from VirusTotal labeling. To better understand the practical benefits of MalWhiteout in the context of VirusTotal labeling, which motivated this work, we evaluate MalWhiteout using the actual mislabels from VirusTotal. Specifically, for the ground truth dataset we used, we inferred labels based on the commonly used threshold-based method (i.e., set a voting

Table 1: Descriptions of metrics used in our experiments.

Metrics	Abbreviation	Definition
True Positive	TP	# samples correctly classified as noise
False Positive	FP	# samples incorrectly classified as noise
Ture Negative	TN	# samples correctly classified as non-noise
False Negative	FN	# samples incorrectly classified as non-noise
Precision	Pre	$TP/(TP+FP)$
Recall	Rec	$TP/(TP+FN)$
F-measure	F1	$2*Pre*Rec/(Pre+Rec)$
Accuracy	Acc	$(TP+TN)/(TP+TN+FP+FN)$
# of Noises	# Noises	# of wrongly-labelled samples in the dataset
# of Noise Left	# Left	$\#Noises-TP+FP$
% of Noise Left	% Left	$\#Left/(TP+TN+FP+FN)$
% of Noise Reduced	% Reduced	$(\#Noises-\#Left)/\#Noises$

[6]An alternative strategy is to simply discard all the identified noisy samples. It avoids generating new noises but causes the loss of some samples (i.e. false positives). In practice, this strategy can certainly be used if sufficient samples are available.

Table 2: Noise detection results at different noise ratios.

Noise Ratio	5%	10%	15%	20%	25%	30%
# Noises	580	1160	1740	2320	2900	3480
TP	556	1130	1661	2196	2690	3142
FP	26	42	72	112	170	260
Pre	0.956	0.964	0.959	0.952	0.941	0.924
Rec	0.973	0.973	0.954	0.946	0.927	0.902
F1	0.964	0.968	0.956	0.948	0.934	0.913
Acc	0.996	0.994	0.987	0.980	0.968	0.948
% Reduced	93.1%	93.8%	91.3%	89.8%	86.9%	82.8%
# Left	50	72	151	236	380	598
% Left	0.43%	0.62%	1.3%	2.04%	3.28%	5.16%

threshold t and if t or more engines return a "malicious" label, then the sample is labeled as malicious). In this way, the mislabeled samples were actual label noises generated by VirusTotal. Evaluating MalWhiteout on these noises can be important as it is designed to mitigate the unreliability of VirusTotal labeling.

Once we collected the malware dataset, we uploaded them to VirusTotal for rescanning to see how many engines managed to flag them. Figure 7 shows the CDF distribution of the number of flagged engines in VirusTotal for each sample. It can be seen that there are indeed a portion of malware samples that were flagged by a limited number of engines, e.g., 50 samples were flagged by less than 10 engines. Thus, if we set the threshold t to 10, these 50 samples will be mislabeled by VirusTotal. Overall, the samples that were flagged by less than 25 engines account for 33%, a quite high percentage of noise that is not common in VirusTotal labeling. We thus consider thresholds ranging from 2 to 20. Table 3 presents the detection results of MalWhiteout on noisy datasets generated by different thresholds. Assuming we set the threshold to 2, 5, 10 and 20 respectively, there will be 8, 50, 137 and 643 mislabels generated. These samples are arguably the most evasive malware samples. Taking them as noises, MalWhiteout can identify most of them. For example, MalWhiteout can find all of the 8 noises in the case of threshold=2 and 46 out of 50 noises in the case of threshold=5. The percentage reduced by MalWhiteout are 100%, 92%, 87.6% and 64.9% respectively, with 0, 4, 17 and 226 noises left, at various thresholds (see the last two rows, i.e., *% Reduced** and *# Left**). However, MalWhiteout also generates some false positives that can be introduced as new noises in the dataset (by flipping labels). With the introduction of new noises, MalWhiteout can reduce the noises by more than half. The results look slightly inferior to the ones in Table 2, probably because of the unbalanced distribution of noises in the two classes (i.e., only some of the malware labels are flipped to become noises) and the number of noises is quite small. Nevertheless, MalWhiteout can still reduce at least half of the noises, much better than state-of-the-art (see §5.5). Overall, MalWhiteout can effectively reduce the number of incorrectly labelled samples from VirusTotal, thereby countering the unreliability of VirusTotal labeling, which is the practical relevance of MalWhiteout in the context of VirusTotal labeling.

Answer to RQ1: MalWhiteout achieved an accuracy over 94% and F1 over 91% in detecting noises at different noise ratios up to 30%. Besides, MalWhiteout can identify most of the actual mislabels from VirusTotal at different thresholds, reflecting its practical benefits in the context of threshold-based VirusTotal labeling.

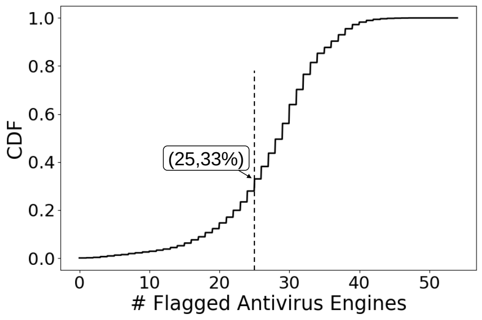

Figure 7: CDF of the number of flagged engines on VirusTotal.

Table 3: Noise detection results at different thresholds.

Threshold	2	5	10	20
# (%) Noises	8 (0.1%)	50 (0.9%)	137 (2.4%)	643 (11%)
TP	8	46	120	417
FP	4	12	27	44
Pre	0.667	0.793	0.816	0.904
Rec	1.0	0.92	0.876	0.649
F1	0.8	0.852	0.845	0.755
Acc	≈ 1.0	0.999	0.996	0.977
% Reduced	50%	68%	67.9%	58%
# Left	4	16	44	270
% Reduced*	100%	92%	87.6%	64.9%
# Left*	0	4	17	226

5.3 RQ2: The Effectiveness of Design Decisions

In the design of MalWhiteout, we have incorporated the idea of ensemble learning and considered inter-app relations based on app developers, aiming to improve its accuracy of noise detection. However, it is open to investigation how useful these design decisions can be. Thus, we study the result of each case in order to understand the effect of each component. First, we have selected three malware detection approaches and extracted features for each. In fact, for each approach, we also use CL to wrap its classification model and find potential noises, as practiced in the preliminary experiment (see § 3.2). To enable ensemble learning, we use a meta-learner to combine the out-of-sample predicted probabilities made by base learners and use CL to find potential noises. By comparing the results of the above models (i.e., three separate models and an ensemble model), we can learn whether and to what extent ensemble learning has made a difference. At last, we perform a partial adjustment of the noise detection results using developer-based app relation. Thereby, we can understand whether and to what extent the app relation has contributed by comparing the results before and after the adjustment.

Table 4 shows the comparison results at different noise ratios, where the optimal value of each metric is bolded. Column 3-5 present the results of noise detection using each of the three individual methods, which exhibit their varying detection abilities. Overall, Drebin works best. The Column 6 shows the detection results using the ensemble learning approach, which exhibits a significant improvement compared to when using a single model. This shows that ensemble learning can mitigate the inherent bias from each model, making the noise estimation capability of CL more robust. The last column shows the detection results after noise adjustment using developer-based app relation. Compared to Column 6, it can be seen that app relation helps reduce the number of false positives, improving the precision of noise detection.

Table 4: Noise detection results obtained at each case.

NoiseRatio (# Noises)	Metric	Separate Models			Ensemble	App Relation
		MalScan	CSBD	Drebin		
5% (580)	TP	515	479	561	558	**566**
	FP	154	137	53	38	**26**
	F1	0.823	0.800	0.938	0.948	**0.964**
	Acc	0.981	0.979	0.994	0.995	**0.996**
10% (1160)	TP	1000	991	1118	1116	**1130**
	FP	273	298	84	63	**42**
	F1	0.821	0.809	0.946	0.953	**0.968**
	Acc	0.963	0.960	0.989	0.991	**0.994**
15% (1740)	TP	1482	1466	1653	1654	**1661**
	FP	456	511	145	114	**72**
	F1	0.804	0.788	0.934	0.943	**0.956**
	Acc	0.938	0.932	0.980	0.983	**0.987**
20% (2320)	TP	1963	1971	2177	2195	**2196**
	FP	634	761	190	159	**112**
	F1	0.798	0.780	0.929	0.939	**0.949**
	Acc	0.914	0.904	0.971	0.975	**0.980**
25% (2900)	TP	2334	2372	2681	**2706**	2690
	FP	853	936	282	229	**170**
	F1	0.767	0.764	0.914	0.927	**0.934**
	Acc	0.877	0.874	0.957	0.963	**0.967**
30% (3480)	TP	2532	2676	3164	**3169**	3142
	FP	1184	1220	374	314	**260**
	F1	0.704	0.725	0.901	0.910	**0.913**
	Acc	0.816	0.825	0.940	0.946	**0.948**

Answer to RQ2: Both the design of ensemble learning based noise detection and app relation based adjustment have made a positive contribution to MalWhiteout. Ensemble learning significantly improves detection over using separate models, and app relation further helps reduce false positives.

5.4 RQ3: Improvement of Malware Detection

As aforementioned, a correctly labelled dataset is highly important for machine learning based Android malware detection. We next verify the impact of training sets of varying quality on malware detection efforts to see how much MalWhiteout can help improve the performance of existing techniques.

We first divide the ground truth dataset into two parts: 70% of it serves as the training set and 30% as the test set. For the training set, we make three separate training sets of varying quality: (1) Correctly-labelled dataset, i.e., the 70% ground truth training set. (2) Noisily-labelled dataset, i.e., we randomly select a certain percentage of apps from the training set and flip their labels to generate a noisily-labelled dataset. Here we set the noise ratio to 10%. (3) Processed dataset, i.e., after MalWhiteout revises the labels in noisily-labelled dataset, we refer to it as the processed dataset. Next, we train a machine learning based Android malware detection approach using either the correctly-labelled dataset, the noisily-labelled dataset, or the processed dataset, and evaluate the performance of each dataset using the 30% ground truth test set.

In this experiment, we also use Drebin, CSBD and MalScan, as the malware detection approach, respectively, for evaluation. For each approach, it is trained on each of the three datasets and tested on the same test set (using its specific features and classifier). This allows us to understand the impact of the quality of the training set on the performance of a machine learning based malware detection approach. We use the composite metric F-measure to characterize the performance of the classification model, with a higher F1 value indicating better classification ability. Figure 8 shows the performance (F1) of the models trained on the correctly-labelled, noisily-labelled and processed training sets separately, for each malware detection approach. We can observe that all the three approaches yielded minimum F1 values on the noisily-labelled dataset, which were 94.55%, 86.12%, and 92.45%, respectively, indicating that the noisy datasets can impair the capacity of existing machine learning based malware detection efforts. Besides, when trained with the processed dataset (i.e., after the labels being revised by MalWhiteout), the models all achieved significant improvements in malware detection. The improved F1 scores are closer to the upper bounds, i.e., the results obtained by training with the correctly-labelled dataset. Specifically, Drebin was improved 94.55% to 96.28% (upper bound 97.04%), CSBD from 86.12% to 92.82% (upper bound 95.38%) and MalScan from 92.45% to 93.91% (upper bound 95.43%). The F1 of the three models improved by 1.83% (Drebin), 7.78% (CSBD) and 1.58% (MalScan) respectively. Likewise, if we set the noise ratio to 20% in the training set, we observe that the F1 of the three models improved by 2.4% (Drebin), 7.89% (CSBD), and 7.21% (MalScan) respectively. This demonstrates MalWhiteout's positive contribution to the existing machine learning based Android malware detection efforts in our research community.

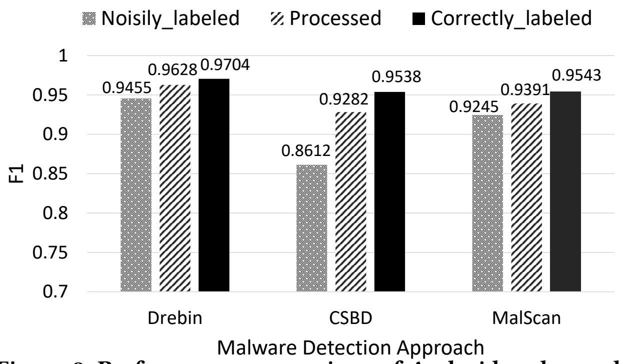

Figure 8: Performance comparison of Android malware detection approaches trained on different datasets.

Answer to RQ3: A noisy training set can negatively impact the performance of existing machine learning based malware detection techniques. MalWhiteout can improve the performance of these methods by dealing with noise in the dataset.

Table 5: Comparison results for noises from random flipping.

Noise Ratio (# Noises)	Metric	MalWhiteout	MalWhiteout (Drebin)	DT
5% (580)	TP	**566**	564	402
	FP	**26**	39	232
	% Reduced	**93.1%**	90.5%	29.3%
	# Left	**40**	55	410
	# Hours	**0.42**	0.42	29.8h
10% (1160)	TP	**1130**	1125	967
	FP	**42**	62	334
	% Reduced	**93.8%**	91.6%	54.6%
	# Left	**72**	97	527
	# Hours	**0.5**	0.5	58.2h
15% (1740)	TP	**1661**	1651	1535
	FP	**72**	91	376
	% Reduced	**91.3%**	89.7%	66.6%
	# Left	**151**	180	581
	# Hours	**0.67**	0.67	88.3h
20% (2320)	TP	**2196**	2170	2066
	FP	**112**	130	331
	% Reduced	**89.8%**	87.9%	74.8%
	# Left	**236**	280	585
	# Hours	**0.64**	0.64	109.3h
25% (2900)	TP	**2690**	2659	2503
	FP	**170**	202	324
	% Reduced	**86.9%**	84.7%	75.1%
	# Left	**380**	443	721
	# Hours	**0.64**	0.64	145.3h
30% (3480)	TP	**3142**	3125	3014
	FP	**260**	303	355
	% Reduced	**82.8%**	81.1%	76.4%
	# Left	**598**	658	821
	# Hours	**0.68**	0.68	170h

Table 6: Comparison results for noises from VirusTotal.

Threshold (# Noises)	Metric	MalWhiteout	MalWhiteout (Drebin)	DT
2 (8)	TP	**8**	8	4
	FP	**4**	15	257
	% Reduced	**50%**	-	-
	# Left	**4**	15	261
	# Hours	**0.5h**	0.5h	6.9h
5 (50)	TP	**46**	46	31
	FP	**12**	18	289
	% Reduced	**68%**	56%	-
	# Left	**16**	22	308
	# Hours	**0.52h**	0.52h	25.8h
10 (137)	TP	**120**	118	67
	FP	**27**	34	344
	% Reduced	**67.9%**	61.3%	-
	# Left	**44**	53	414
	# Hours	**0.56h**	0.56h	35.4h
20 (643)	TP	**414**	409	247
	FP	**44**	96	369
	% Reduced	**58%**	48.7%	-
	# Left	**270**	330	765
	# Hours	**0.6h**	0.6h	38.9h

5.5 RQ4: Compare with State-of-the-art

To the best of our knowledge, Differential Traning [47] (DT) is the only existing work on noise reduction for Android malware datasets. Therefore, we use DT as the baseline and compare with it in terms of noise detection effectiveness and efficiency.

The core idea of DT is to build two deep learning classification models (i.e., noise detection models), one is trained using the whole training set of apps (called WS model) and the other is trained on the randomly down-sampled set of apps (called DS model). It uses a heuristic method to distinguish wrongly-labeled samples from correctly-labeled samples by outlier detection on loss vectors generated by the intermediate states of the two models. The labels of the outlier samples are flipped with a certain probability in each iteration, until the fluctuation of the estimated noise ratio becomes smaller than a certain threshold over several iterations. Thus it achieves noise reduction usually requiring a number of iterations. In the pre-processing phase of DT, a machine learning based malware detection approach is selected to transform the raw app files into numeric feature vectors. In our experiments conducted in this paper, we select Drebin malware detection to extract feature vectors which are fed into DT, and use Multi-Layer Perceptron (MLP) with two hidden layers as noise detection models, following the paper [47].

In this experiment, we seek to compare the performance of DT and MalWhiteout. Given that DT uses feature vectors specified by Drebin, we also add a control experiment using only Drebin's features (i.e. omitting the ensemble learning component) for the sake of fairness. The comparative results are presented in Table 5 and Table 6. We bold the optimal value of each metric.

5.5.1 The Effectiveness of Noise Detection. We first focus on the effectiveness of noise detection. Based on data setup in §5.2, we also design two sets of experiments, i.e., detect label noises from (1) random flipping and (2) VirusTotal labeling. As shown in Table 5 and Table 6, MalWhiteout achieves much better results than DT in all cases, resulting in more true positives, less false positives, a greater proportion of noise being reduced and a smaller amount of noise left. Even with only Drebin's features, i.e. omitting the ensemble learning process, our approach can still achieve better results than DT (see the last two columns). For noises from random flipping (see Table 5), the difference in effect is particularly noticeable when the noise ratio is relatively low. For example, at a noise ratio of 5%, DT can only reduce noises by less than 30%, whereas MalWhiteout is able to reduce by more than 90% (achieving 218% improvement). With the noise ratio increases, DT works better. At a noise ratio of 30%, DT reduces 76.4% of the wrong labels, which is getting closer to MalWhiteout's 81.1% and 82.8% (about 8% improvement). For mislabels from VirusTotal labeling (see Table 6), MalWhiteout performs remarkably better than DT. For example, when threshold=2, MalWhiteout performed worst reducing the noise by 50%, while DT produced an intolerable number of false positives and even exacerbated the original noise level. Therefore, MalWhiteout is much more effective than DT.

5.5.2 The Efficiency of Noise Detection. We next take a look at the time cost (# Hours) of the two approaches, which indicates the efficiency and the feasibility in practice. For MalWhiteout, the operations can be divided into three phases: pre-processing (including feature extraction and signature extraction for samples), ensemble-based noise detection and app relation based adjustment. Figure 9 shows the timeline of MalWhiteout's workflow. The most time-consuming part lies in the feature extraction, with Drebin taking the longest time of 22.5 hours in total. Note that during the pre-processing phase we process the three malware detection methods in parallel, thus the pre-processing takes 22.5 hours in total. Nevertheless, since the same feature extraction (by Drebin)

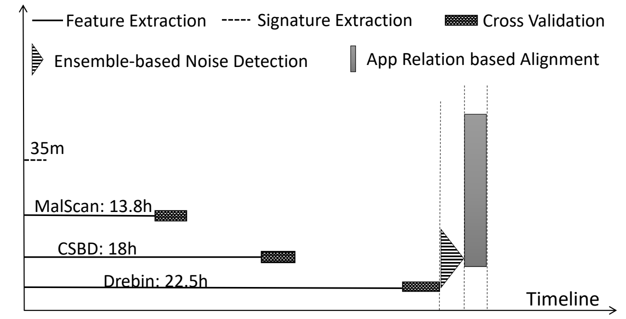

Figure 9: Timeline of MalWhiteout's workflow.

step is also required for DT, the two methods take equal time for preprocessing and we do not take this into account in the comparison. Therefore, the time duration in Table 5 and Table 6 (# Hours) refers to the time spent on the noise detection process that is performed after the pre-processing step has been completed. For DT, it uses a stop criterion that is similar to the early stopping. Since it reduces the label noises in a gradual way in multiple iterations, the more noises in the dataset the more iterations it requires and the longer time it takes. We can see that MalWhiteout has a stable time cost within 40 minutes (0.68 hour), while DT takes 29.8-170 hours as the noise ratio increase from 5% to 30%. Overall, MalWhiteout is much more efficient compared to DT, being 70 to 249 times faster.

Answer to RQ4: MalWhiteout outperforms the state-of-the-art (i.e., DT) in terms of both the detection effectiveness and efficiency (time cost). This makes MalWhiteout more appealing for practical usage.

6 THREATS TO VALIDITY

Despite the promising results, this work has three potential threats to validity. First, we make an assumption that the majority of samples in the dataset used for malware detection are correctly labelled, thus the maximum noise ratio is taken to be 30% in our experiments. There is a chance that the assumption could be violated but we believe it fits the vast majority of situations. Second, we leverage Confident Learning to estimate noise for malware detection due to its encouraging performance in our preliminary study. However, there are actually other noise handling methods in the research community that could be ported over. We will try more approaches in our future work. Third, we use the developer-based app relation to assist in noise detection, using a majority voting mechanism to infer the samples' true labels. This allows most of the samples to be labeled correctly. However, we acknowledge that there can exist corner cases where the labels are mistakenly revised, e.g., a real benign app released by a developer who releases a lot of malicious apps will also be considered as malware. Besides, there can be extreme situations where no apps or very few apps share a developer. In that case the app relation based adjustment would not work. Nevertheless, our experiment indicates that even without this step, the detection results produced by ensemble learning look good.

7 RELATED WORK

7.1 Android Malware Detection

Many Android malware detection studies incorporated traditional machine learning methods such as K-nearest Neighbours (KNN), Support Vector Machines (SVM), Decision Trees, Random Forests (RF) and Naive Bayesian (NB) [5, 6, 8, 13, 17, 26, 32, 44, 45, 48]. For example, *DroidAPIMiner* [5] extracted features relying on the API, package, and parameter level information from apps and used different algorithms (e.g., RF, KNN, SVM, etc.) for classification. In addition, an increasing number of researchers are using deep learning methods for Android malware detection. For example, *MalDozer* [22] uses features such as the raw sequences of app's API method calls and relies on deep learning techniques to automatically learn patterns to detect Android malware.

7.2 Label Noise Reduction

A large number of approaches have been proposed to manage noisy labels using traditional machine learning techniques (e.g., KNN, outlier detection) [11, 12, 37]. Besides, researchers have devoted significant effort to improve the robustness of conventional models (e.g., SVM, DT) to the noisy labels [9, 15]. In recent years, an increasing number of studies are using deep learning techniques to overcome the issue of noisy labels [19, 27, 35]. Many efforts [18, 20, 24, 33, 49] focused on the sample selection research direction, i.e., identifying correctly-labeled examples from noisy data via multi-network or multi-round learning. A number of studies [10, 16, 34, 46] have attempted to modify the architecture to model the label transition matrix of noisy datasets, e.g., adding a noise adaptation layer at the top of the softmax layer or designing a new dedicated architecture. Despite all these efforts to deal with noisy labels, there are many challenges in applying them to solve the noise problem in malware detection, such as intolerant false positives. Nevertheless, we admit that advanced noise reduction techniques in the machine learning community can be complementary to this work.

8 CONCLUSION

In this paper, we present a novel noise detection method, MalWhiteout, to reduce label noises in datasets for machine learning based Android malware detection. We exploit existing noise handling techniques, migrating Confident Learning method to the malware detection domain. We further incorporate the idea of ensemble learning and app relation based adjustment to improve MalWhiteout's robustness. We conduct a comprehensive evaluation on a curated reliable dataset. Experimental results indicate that MalWhiteout is capable of detecting noise with high accuracy and is more effective and greatly faster than the state-of-the-art. Moreover, MalWhiteout can improve the performance of existing machine learning based Android malware detection efforts.

ACKNOWLEDGMENTS

This work was supported in part by National Key R&D Program of China (2021YFB2701000), the National Natural Science Foundation of China (grant No.62072046), the Fundamental Research Funds for the Central Universities (HUST 3004129109), and Hong Kong RGC Projects (No. PolyU15219319, PolyU15222320, PolyU15224121).

REFERENCES

[1] 2019. Stacking in Machine Learning. https://www.geeksforgeeks.org/stacking-in-machine-learning/.

[2] 2022. Koodous. https://koodous.com.

[3] 2022. Publication Trends. https://app.dimensions.ai/discover/publication.

[4] 2022. VirusTotal. https://www.virustotal.com/.

[5] Yousra Aafer, Wenliang Du, and Heng Yin. 2013. Droidapiminer: Mining api-level features for robust malware detection in Android. In *International conference on security and privacy in communication systems*. Springer, 86–103.

[6] Kevin Allix, Tegawendé F Bissyandé, Quentin Jérome, Jacques Klein, State Radu, and Yves Le Traon. 2016. Empirical assessment of machine learning-based malware detectors for Android. *Empirical Software Engineering* 21, 1 (2016), 183–211.

[7] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2016. Androzoo: Collecting millions of Android apps for the research community. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 468–471.

[8] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and CERT Siemens. 2014. Drebin: Effective and explainable detection of Android malware in your pocket. In *NDSS*, Vol. 14. 23–26.

[9] Battista Biggio, Blaine Nelson, and Pavel Laskov. 2011. Support vector machines under adversarial label noise. In *Asian conference on machine learning*. PMLR, 97–112.

[10] Xinlei Chen and Abhinav Gupta. 2015. Webly supervised learning of convolutional networks. In *Proceedings of the IEEE international conference on computer vision*. 1431–1439.

[11] Sarah Jane Delany, Nicola Segata, and Brian Mac Namee. 2012. Profiling instances in noise reduction. *Knowledge-Based Systems* 31 (2012), 28–40.

[12] Dragan Gamberger, Nada Lavrac, and Saso Dzeroski. 2000. Noise detection and elimination in data preprocessing: experiments in medical domains. *Applied artificial intelligence* 14, 2 (2000), 205–223.

[13] Joshua Garcia, Mahmoud Hammad, Bahman Pedrood, Ali Bagheri-Khaligh, and Sam Malek. 2015. Obfuscation-resilient, efficient, and accurate detection and family identification of Android malware. *Department of Computer Science, George Mason University, Tech. Rep* 202 (2015).

[14] Aritra Ghosh, Himanshu Kumar, and PS Sastry. 2017. Robust loss functions under label noise for deep neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 31.

[15] Aritra Ghosh, Naresh Manwani, and PS Sastry. 2017. On the robustness of decision tree learning under label noise. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 685–697.

[16] Jacob Goldberger and Ehud Ben-Reuven. 2016. Training deep neural-networks using a noise adaptation layer. (2016).

[17] Alessandra Gorla, Ilaria Tavecchia, Florian Gross, and Andreas Zeller. 2014. Checking app behavior against app descriptions. In *Proceedings of the 36th international conference on software engineering*. 1025–1035.

[18] Bo Han, Quanming Yao, Xingrui Yu, Gang Niu, Miao Xu, Weihua Hu, Ivor Tsang, and Masashi Sugiyama. 2018. Co-teaching: Robust training of deep neural networks with extremely noisy labels. *Advances in neural information processing systems* 31 (2018).

[19] Dan Hendrycks, Mantas Mazeika, Duncan Wilson, and Kevin Gimpel. 2018. Using trusted data to train deep networks on labels corrupted by severe noise. *Advances in neural information processing systems* 31 (2018).

[20] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. 2018. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In *International Conference on Machine Learning*. PMLR, 2304–2313.

[21] Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Brad Miller, Vaishaal Shankar, Rekha Bachwani, Anthony D Joseph, and J Doug Tygar. 2015. Better malware ground truth: Techniques for weighting anti-virus vendor labels. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*. 45–56.

[22] ElMouatez Billah Karbab, Mourad Debbabi, Abdelouahid Derhab, and Djedjiga Mouheb. 2018. MalDozer: Automatic framework for Android malware detection using deep learning. *Digital Investigation* 24 (2018), S48–S59.

[23] Li Li, Daoyuan Li, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, David Lo, and Lorenzo Cavallaro. 2017. Understanding Android app piggybacking: A systematic study of malicious code grafting. *IEEE Transactions on Information Forensics and Security* 12, 6 (2017), 1269–1284.

[24] Eran Malach and Shai Shalev-Shwartz. 2017. Decoupling "when to update" from "how to update". *Advances in Neural Information Processing Systems* 30 (2017).

[25] Naresh Manwani and PS Sastry. 2013. Noise tolerance under risk minimization. *IEEE transactions on cybernetics* 43, 3 (2013), 1146–1151.

[26] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. 2017. MAMADROID: Detecting Android malware by building markov chains of behavioral models. In *Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS)*.

[27] Aditya Krishna Menon, Ankit Singh Rawat, Sashank J Reddi, and Sanjiv Kumar. 2019. Can gradient clipping mitigate label noise?. In *International Conference on Learning Representations*.

[28] Curtis Northcutt, Lu Jiang, and Isaac Chuang. 2021. Confident learning: Estimating uncertainty in dataset labels. *Journal of Artificial Intelligence Research* 70 (2021), 1373–1411.

[29] Giorgio Patrini, Alessandro Rozza, Aditya Krishna Menon, Richard Nock, and Lizhen Qu. 2017. Making deep neural networks robust to label noise: A loss correction approach. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1944–1952.

[30] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *28th USENIX Security Symposium (USENIX Security 19)*. 729–746.

[31] Silvia Sebastian and Juan Caballero. 2020. Towards attribution in mobile markets: Identifying developer account polymorphism. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 771–785.

[32] Jingya Shen, Zhenxiang Chen, Shanshan Wang, Yuhui Zhu, and Muhammad Umair Hassan. 2018. DroidDetector: a traffic-based platform to detect Android malware using machine learning. In *Third International Workshop on Pattern Recognition*, Vol. 10828. International Society for Optics and Photonics, 108280N.

[33] Yanyao Shen and Sujay Sanghavi. 2019. Learning with bad training data via iterative trimmed loss minimization. In *International Conference on Machine Learning*. PMLR, 5739–5748.

[34] Sainbayar Sukhbaatar, Joan Bruna, Manohar Paluri, Lubomir Bourdev, and Rob Fergus. 2014. Training convolutional networks with noisy labels. *arXiv preprint arXiv:1406.2080* (2014).

[35] Ryutaro Tanno, Ardavan Saeedi, Swami Sankaranarayanan, Daniel C Alexander, and Nathan Silberman. 2019. Learning from noisy labels by regularized estimation of annotator confusion. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 11244–11253.

[36] Kurt Thomas, Juan A Elices Crespo, Ryan Rasti, Jean-Michel Picod, Cait Phillips, Marc-André Decoste, Chris Sharp, Fabio Tirelo, Ali Tofigh, Marc-Antoine Courteau, et al. 2016. Investigating Commercial Pay-Per-Install and the Distribution of Unwanted Software. In *25th USENIX Security Symposium (USENIX Security 16)*. 721–739.

[37] Jaree Thongkam, Guandong Xu, Yanchun Zhang, and Fuchun Huang. 2008. Support vector machine for outlier detection in breast cancer survivability prediction. In *Asia-Pacific Web Conference*. Springer, 99–109.

[38] Haoyu Wang, Zhe Liu, Jingyue Liang, Narseo Vallina-Rodriguez, Yao Guo, Li Li, Juan Tapiador, Jingcun Cao, and Guoai Xu. 2018. Beyond Google play: A large-scale comparative study of Chinese Android app markets. In *Proceedings of IMC 2018*. 293–307.

[39] Haoyu Wang, Junjun Si, Hao Li, and Yao Guo. 2019. RmvDroid: Towards a reliable Android malware dataset with app metadata. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 404–408.

[40] Liu Wang, Ren He, Haoyu Wang, Pengcheng Xia, Yuanchun Li, Lei Wu, Yajin Zhou, Xiapu Luo, Yulei Sui, Yao Guo, et al. 2021. Beyond the virus: a first look at coronavirus-themed Android malware. *Empirical Software Engineering* 26, 4 (2021), 1–38.

[41] Liu Wang, Haoyu Wang, Ren He, Ran Tao, Guozhu Meng, Xiapu Luo, and Xuanzhe Liu. 2022. MalRadar: Demystifying Android Malware in the New Era. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 6, 2 (2022), 1–27.

[42] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. 2017. Deep ground truth analysis of current Android malware. In *International conference on detection of intrusions and malware, and vulnerability assessment*. Springer, 252–276.

[43] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. 2017. Deep ground truth analysis of current Android malware. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 252–276.

[44] Dong-Jie Wu, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu. 2012. Droidmat: Android malware detection through manifest and api calls tracing. In *2012 Seventh Asia Joint Conference on Information Security*. IEEE, 62–69.

[45] Yueming Wu, Xiaodi Li, Deqing Zou, Wei Yang, Xin Zhang, and Hai Jin. 2019. Malscan: Fast market-wide mobile malware scanning by social-network centrality analysis. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 139–150.

[46] Tong Xiao, Tian Xia, Yi Yang, Chang Huang, and Xiaogang Wang. 2015. Learning from massive noisy labeled data for image classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2691–2699.

[47] Jiayun Xu, Yingjiu Li, and Robert H Deng. 2021. Differential training: A generic framework to reduce label noises for Android malware detection. (2021).

[48] Suleiman Y Yerima, Sakir Sezer, and Igor Muttik. 2014. Android malware detection using parallel machine learning classifiers. In *2014 Eighth international conference on next generation mobile apps, services and technologies*. IEEE, 37–42.

[49] Xingrui Yu, Bo Han, Jiangchao Yao, Gang Niu, Ivor Tsang, and Masashi Sugiyama. 2019. How does disagreement help generalization against label corruption?. In

International Conference on Machine Learning. PMLR, 7164–7173.
[50] Yajin Zhou and Xuxian Jiang. 2012. Dissecting Android malware: Characterization and evolution. In *2012 IEEE symposium on security and privacy*. IEEE, 95–109.
[51] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. 2020. Measuring and modeling the label dynamics of online anti-malware engines. In *29th USENIX Security Symposium (USENIX Security 20)*. 2361–2378.