Drift Forensics of Malware Classifiers

Theo Chow
theo.chow@kcl.ac.uk
King's College London
United Kingdom

Zeliang Kan
zeliang.kan@kcl.ac.uk
King's College London
University College London
United Kingdom

Lorenz Linhardt
l.linhardt@campus.tu-berlin.de
TU Berlin, BIFOLD
Germany

Lorenzo Cavallaro
l.cavallaro@ucl.ac.uk
University College London
United Kingdom

Daniel Arp
d.arp@tu-berlin.de
TU Berlin
Germany

Fabio Pierazzi
fabio.pierazzi@kcl.ac.uk
King's College London
United Kingdom

ABSTRACT

The widespread occurrence of mobile malware still poses a significant security threat to billions of smartphone users. To counter this threat, several machine learning-based detection systems have been proposed within the last decade. These methods have achieved impressive detection results in many settings, without requiring the manual crafting of signatures. Unfortunately, recent research has demonstrated that these systems often suffer from significant performance drops over time if the underlying distribution changes—a phenomenon referred to as concept drift. So far, however, it is still an open question which main factors cause the drift in the data and, in turn, the drop in performance of current detection systems.

To address this question, we present a framework for the in-depth analysis of dataset affected by concept drift. The framework allows gaining a better understanding of the root causes of concept drift, a fundamental stepping stone for building robust detection methods. To examine the effectiveness of our framework, we use it to analyze a commonly used dataset for Android malware detection as a first case study. Our analysis yields two key insights into the drift that affects several state-of-the-art methods. First, we find that most of the performance drop can be explained by the rise of two malware families in the dataset. Second, we can determine how the evolution of certain malware families and even goodware samples affects the classifier's performance. Our findings provide a novel perspective on previous evaluations conducted using this dataset and, at the same time, show the potential of the proposed framework to obtain a better understanding of concept drift in mobile malware and related settings.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Knowledge representation and reasoning**; • **Security and privacy** → **Intrusion/anomaly detection and malware mitigation**.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
AISec '23, November 30, 2023, Copenhagen, Denmark.
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0260-0/23/11…$15.00
https://doi.org/10.1145/3605764.3623918

KEYWORDS

Machine Learning; Malware; Concept Shift; Explainable AI;

ACM Reference Format:
Theo Chow, Zeliang Kan, Lorenz Linhardt, Lorenzo Cavallaro, Daniel Arp, and Fabio Pierazzi. 2023. Drift Forensics of Malware Classifiers. In *Proceedings of the 16th ACM Workshop on Artificial Intelligence and Security (AISec '23), November 30, 2023, Copenhagen, Denmark.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3605764.3623918

1 INTRODUCTION

The relentless proliferation of Android malware has emerged as a critical concern, posing a substantial security risk to billions of Android users worldwide [16, 18]. More specifically, as individuals rely on applications to enhance the convenience of their daily lives, they face an ongoing threat to the privacy of their personal information and critical data. Researchers have employed various datasets and Artificial Intelligence (AI) models to detect and combat these malicious applications [5, 24, 29, 34]. Unfortunately, it is difficult for AI models to maintain a high level of detection performance when they are deployed in real-world scenarios. As shown in [35], models trained on malware datasets that consist of a temporal component will decay over time. Although there are strategies that suggest methods for dealing with this temporal bias [23], few have attempted to explain the reason for this decay.

Back in 2017, a report from Securelist [41] suggested the growth of mobile ransomware and trojans aimed at stealing personal information and abusing super-user rights. Whereas a review in 2020 [42] suggested two-thirds of malicious applications mainly collect expense consumption data from their users. This difference suggests a slight change in the behaviour of malicious applications.

With the countless malware families in Android malware datasets [8, 23, 35, 44], our intuition is that malware families are one of the causes of concept drift. While this was hinted in prior work [23, 35], this hypothesis has not been tested so far. Concept drift refers to the temporal changes in malware characteristics and behaviours, which can undermine the effectiveness of detection and classification systems over time. To investigate the relationship between malware families and concept drift, we explore the following research question:

How and why does concept drift impact the performance of Artificial Intelligence (AI) classifiers in detecting and classifying multi-family Android malware over time?

In this paper, we contribute to the field of Drift Forensics, a newly established direction of post-hoc analysis of drifted data [17, 38, 48, 50]. Our primary objective is to explore the intricate relationship between concept drift and the distribution of malware families. By leveraging explanations, we conduct an in-depth investigation into the individual contributions of each malware family to the performance of the AI classifier. Through this comprehensive examination, we aim to enhance our understanding of evolving malware trends and empower security practitioners with valuable insights for proactive defence against Android malware threats.

By contributing to Drift forensics in the context of malware family distribution, this paper extends the existing body of knowledge in digital forensics and malware analysis. Hence, we list the contributions this paper provides:

- We propose a new framework for drift forensics towards post-hoc analysis of concept drift based on Explainable AI (XAI) methods.
- To make the analysis feasible, we present a way to automatically identify a small number of Points of Interest (POI) promising for further analysis.
- We demonstrate the effectiveness of the framework on a large mobile malware dataset. In particular, our framework allows us to obtain new insights into the drift root causes.
- To foster future research, we make our code publicly available at https://github.com/isneslab/DriftAnalysis.

The insights gained from our investigation provide a foundation for developing adaptive and resilient detection and classification systems that can effectively counter the ever-changing Android malware landscape.

2 METHODOLOGY

While previous work has shown that concept drift can significantly affect the performance of current learning models, only a few focus on explaining the root causes of the drift in the data. Zola et al. [50] showcased a detailed post-hoc analysis on portable executables (PE) malware for digital forensics. We build on this work but focus our analysis on drift and identifying individual features using XAI. Yang et al. [48] developed a visualisation tool for detecting and explaining drift, however, their method is computationally expensive and difficult to scale. Demšar and Bosnić [17] used interaction-based methods for explanations to detect drift, but is unable to differentiate different types of drift and relies on tuning parameters. We argue that precise knowledge about the root causes of the drift is essential to build robust defenses and mitigations.

Our framework aims to close this gap and provides a systematic post-hoc methodology for analyzing malware datasets affected by concept drift. While it is commonly assumed that the concept drift is caused by the evolution of malware families [23, 35], this hypothesis has not been systematically examined so far. To build a suitable framework to check this hypothesis, we thus guide our research along the following investigative steps:

(1) Automatically identify relevant POI for analysis
(2) Investigate the contributions of benign applications to the performance decay.
(3) Investigate the contributions of *emerging* malware families to the performance decay.
(4) Investigate the contributions of *evolving* malware families to the performance decay.

Since this is a post-hoc analysis on dataset drift, we assume that ground truth labels are available. We discuss this limitation in section §4. Our framework provides a systematic approach for practitioners to analyze their data by focusing on the influence of families. By following a series of experimental steps, researchers gain insights into the impact of families on classifier performance decay.

2.1 Identifying Points of Interest

Due to the large amounts of applications and features, conducting a detailed analysis for all months is infeasible. Also, variations in the number of test samples collected in different months can impact the performance. In order to reduce the required effort to a manageable amount, we first need a method to select months that are likely to provide essential information on the underlying drift. We refer to this set of points as Points of Interest (POIs).

Oracle models. The main intuition behind our approach is the following: a performance drop occurs if the learned classifier does not sufficiently capture the test distribution anymore. In order to find characteristic points of the drift, we compare our original classification model (without knowledge of the test distribution) with an additional model that incorporates knowledge about the test distribution and encodes (some of) the information the original model was unable to capture. We refer to the latter as the *oracle model*. We train a separate oracle model for every month and compare the performance of each oracle with the performance of the original model. By analyzing those months where the performance of the oracle and the original model start to diverge, we should be able to better understand the drift that caused this difference.

Training oracles. Figure 1 illustrates our oracle experimental setup. We first train our reference model, C_{Base}, on the first months of the training data and examine its performance in the remaining time span. This model does not have additional information about the test distribution. We develop an oracle classifier, C_1, trained on the same training data but including additional data from the test month in question. These additional data include both malware and goodware samples. This enables C_1 to learn not only from the same samples as C_{Base} but also from samples belonging to the same month it is tested on. We also develop an oracle classifier, C_2, trained on the same training data as C_1 but include only malware samples from the test month in question. This is targeted for the analysis of goodware. To ensure no training samples are reused, we split each test month in half. Half of the samples are used for testing C_{Base}, C_1 and C_2, while the other half is used for training C_1 and C_2 only, with C_2 only using malware samples.

Selecting POIs. By simulating a classifier operating at test time and an oracle classifier that samples test data, we can plot the performance of both classifiers and compare it with the distribution

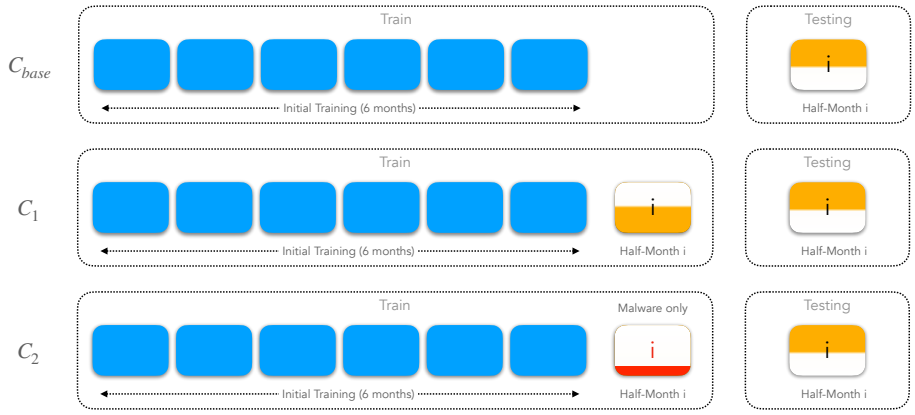


Figure 1: Oracle setup to identify points of interest. The first 6 months of training, represented in blue rounded boxes, contain both goodware and malware applications. Half of each test month i is used for training against all oracles, for making them comparable. C_{Base} is the classifier without future knowledge, whereas C_1 and C_2 represent the oracles containing part of the data from the test month i in the training phase. The differences in performance between C_{Base}, C_1, and C_2 can help us to prioritize points that require further investigation.

of families. Now to identify the set of POIs P, we first take the difference $d_{F1}(t)$ of F1 scores between both classifiers C_{Base} and C_1 for every month t in the test data.

$$d_{F1}(t) = F1(C_1)_t - F1(C_{Base})_t$$

As we are interested in those months where the performance of both classifiers begin to diverge significantly, we calculate the gradient for all points t with $d_{F1}(t) > 0$, i.e., all months in which the oracle C_1 outperforms the original classifier C_{Base}. Finally, we select the top-k points t with the steepest slope for further investigation.

$$P = \{t | t \in top_k(D)\} \tag{1}$$

$$\text{with } D = \{d_{F1}(t) - d_{F1}(t-1), \forall t \in T\},$$

where T denotes all months in the test set and top-k a function that returns the set of the k largest values of a given set. Note that we set $k = 3$ for the experiments described in this paper.

2.2 Identifying Candidate Families

After selecting POIs, we proceed to hypothesize which families contribute to the distribution change between the training and testing data. To achieve this, we compare for each family the number of samples by C_{Base} of that family, denoted as $TP(C_{Base})$, with the true positives of C_1, denoted as $TP(C_1)$, and normalize these values based on the total number of samples N in each month:

$$m = max\left(0, \frac{TP(C_1) - TP(C_{Base})}{N}\right) \tag{2}$$

By plotting the normalized number of missed samples m, we can identify which family contributed to the decay in performance at the identified POIs. This allows us to form hypotheses about the families that may have contributed to the performance decay in specific months of our data.

2.3 Explaining the Drift

Equipped with the Points of Interest (POI) and the candidate families, we can finally explore the underlying drift in detail. Here, we divide our analysis into two parts. In the first part, we examine the impact of emerging families on the classifier's performance. In the second part, we investigate how the evolution of different malware families contributes to the shift.

2.3.1 Analyzing emerging malware families. From the previous steps, we have identified families that appear in the training set and not in the test set. In order to verify whether this family is indeed the main contributor to the drift observed, we reintroduce samples of that family into the training set. Our intuition is a classifier trained on these samples of the drifted families will not be impacted greatly when similar samples of that family appear in test time. To further explain the reason an identified family affects the classifier's performance, we employ explanation methods. As suggested by Warnecke et al. [43], for white-box explanation methods in cybersecurity, Integrated Gradients (IG) [40] and Layer-wise Relevance Propagation (LRP) [10, 31] are the most ideal choice due to the criteria they proposed [20, 40]. However, in the case of a linear Support Vector Machine (SVM), IG gives the same result as Gradient × Input [40], where the Gradient corresponds to the classifier weights vector. In this paper, we utilize Gradient × Input as the representative explanation method due to its reliability and simplicity. The samples C_{Base} missed but C_1 captured are the samples we are most interested in. By sampling the given test month, C_1 learned features that it deems important for capturing those missed samples. Hence, we take a more detailed look at them using explanations.

We do this by comparing the top-n features used by the original classifier C_{Base} with the oracle classifier C_1 for the missed samples. By comparing the differences in feature attribution values and/or

appearance of new top features, we can give detailed reasoning on the features that contributed to drift.

2.3.2 Analyzing the evolution of malware families. To examine the malware evolution of each family in detail, we must isolate them individually and analyze their performance with respect to each other. Firstly, we train a classifier only on samples of one family from the training set. We then test our classifier on the test set. If malware evolution exists, we should observe the true positives of the trained family to decrease over time. However, if there is no malware evolution, then the classifier can capture all samples of the same family regardless of time.

We first statistically examine the familiesby using t-SNE, a dimensionality reduction technique to visualize data points in a lower dimensional space. We do this by using t-SNE on malware samples labelled according to their families. This will ensure each data point is with respect to each other for comparable visualization. By observing the t-SNE plots, we should gain an intuition on the drifting families.

The second method once again uses XAI techniques. By examining POIs we have identified before, we should expect families that drifted due to malware evolution to have different top-n features. Whereas families that do not evolve should have consistent top-n features. Note that we set $n = 5$ for the experiments described in this paper, as our preliminary experiments showed that is sufficient for meaningful results.

3 EVALUATION

Equipped with the framework described in §2, we can finally explore the root causes of the concept drift within the data. To this end, we guide our experiments along three research questions that we aim to answer throughout the evaluation:

- RQ1: How does the concept drift of benign applications affect the classifier's performance?
- RQ2: How does the *emergence* of new malware families affect the classifier's performance?
- RQ3: How does the *evolution* of existing malware families affect the classifier's performance?

3.1 Dataset

We conduct our analysis on the Transcendent dataset [11], which extends the dataset used by Pendlebury et al. [35] to study the impact of concept drift in the mobile malware domain. The dataset covers a time span of 5 years, ranging from 2014 until 2018. Barbero et al. [11] showed that a concept drift also exists in the extended data. So far, however, the exact reasons for the drift in this dataset are still unknown, making it an excellent candidate to analyze with our framework.

To better understand the drift reported in the original paper, we first obtain ground-truth labels for the samples in this dataset. In particular, we rescan the entire dataset using the VirusTotal service[1] and use AVClass2 [37] to obtain family labels. In total, we find that the complete dataset contains 177 different families. However, we notice that most of the malware consists only of a few families. To simplify our analysis, we therefore decided to select only the five largest families of the dataset and examine the drift on this data. More information about individual families can be found in Appendix A. We re-evaluate the performance of the classifier on the reduced dataset to ensure that the reduction does not affect the drift. Figure 5 shows similar performance as presented in [11].

[1] https://www.virustotal.com

3.2 Classifier

While there are several different learning methods that can be used for a two-class malware classification problem, we consider a Linear SVM [12, 15] for the remainder of this paper, with hyper-parameter C=1. This is to be consistent with the papers from which the dataset originated [11, 35] .

3.3 Results

RQ1 *— How does the concept drift of benign applications affect performance?* To answer the first research question, we compare the two oracle classifiers C_1 and C_2. Although both classifiers are trained on additional data, C_1 samples from both malware and goodware. Meanwhile, C_2 samples only from malware (cf. Figure 1). This means C_2 does not have any information about the goodware of the test month while C_1 does. By comparing C_1 and C_2, we can observe how goodware affects the classifier performance.

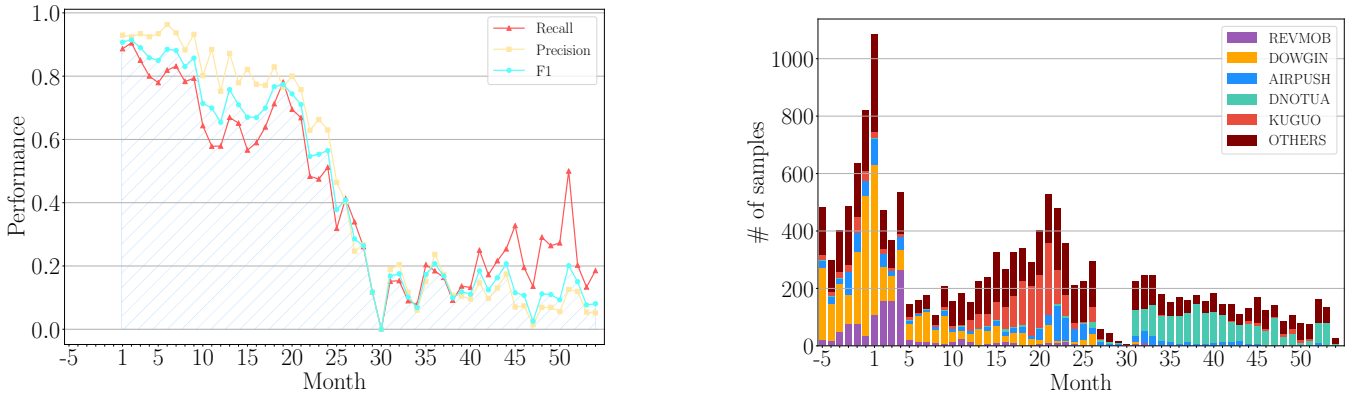
Figure 3 shows the difference in the performance of these two classifiers. It is evident that the performance is not affected greatly by the goodware samples. This suggests that there is minimal drift in goodware and the samples in the training data are representative of those in the test set, confirming results in prior research [35]. We can further confirm this by analysing the breakdown between the true positives of both classifiers. Figure 4a shows that the proportion of goodware samples that changed due to C_1 is very low.

> Goodware samples have little contribution to the performance decay and do not affect the classification of malware samples.

RQ2 *— How does the emergence of new malware families affect the classifier's performance?* We once again follow the steps of our framework by first identifying POIs. By comparing the F1 scores of the oracle classifier C_1 with our original classifier C_{Base}, one can determine POIs for further investigation. Figure 5 shows the performance of both classifiers and the POIs identified by using the gradient of the differences of the F1 scores. These include months 25, 31 and 52.

Then, we identify candidate families by comparing the true positives of C_{Base} and C_1 (see §2.2). Analyzing Figure 4b, we observe that for month 25, *Airpush* seems to be the only family that helped C_1 improve; for months 31 and 52 *Dnotua* and *Airpush* are the families that C_1 managed to capture that C_{Base} missed.

From the above, we can expect *Dnotua* and *Airpush* to be the main drivers of performance decay for a given month. Following the identification of *Dnotua* and *Airpush* as an impactful family to drift in the reduced dataset. We train a classifier on Transcendent with the first 6 months of data and *Dnotua* samples of month 31 to 36. Figure 6 shows the detection performance of such classifier, which has a significant improvement compared to results shown in Figure 2.



(a) Detection performance over time

(b) Family distribution over time

Figure 2: Detection performance and distribution of TRANSCENDENT dataset with all families.

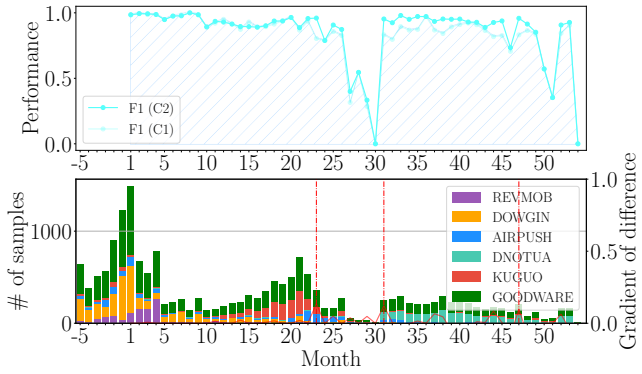


Figure 3: Performance of oracle classifier with (C_1) and without goodware (C_2).

Table 1: Mean of the weights of the top-3 features in missed *Dnotua* samples.

Feature	Weight C_{Base}	Weight C_1	Change
googletagmanager.com	0.692	41.650	↑ +6022.226 %
startService	0.613	33.972	↑ +5545.382 %
intent_LAUNCHER	13.307	3.612	↓ -72.856 %

We find that a classifier trained on *Dnotua* samples is able to maintain a significantly higher detection performance. However, there are still drops in the F1 score, which suggests that *Dnotua* is not the sole reason for performance decay. *Airpush*, despite it existing in the training set, is still affecting the classifier's performance.

To identify specific features that contribute to this result, we examine the difference in top-5 features for missed samples in both C_{Base} and C_1 Surprisingly, we find that our classifiers only consider three features as important for detecting *Dnotua*. All other features have a weight of 0. However, the two top features are heavily upweighted in C_1 when compared to C_{Base}. We show these three features in Table 1.

To get a better understanding of this finding, we look more closely into the two features and why they are used by the classifier to detect the *Dnotua* family. The first feature, the URL *googletagmanager.com*, shows that the malware is using the Google Tag Manager, which is a tool commonly used for tracking and advertising. While it is a legitimate service, it has been reported to be misused by malicious actors in the past.[2] In our dataset, this feature is present

Table 2: The appearance of top-3 *Dnotua* features in Goodware, Malware and *Dnotua* samples

Feature	Appearances		
	Malware	Goodware	Dnotua
googletagmanager.com	37.15%	39.21%	88.60%
startService	22.97%	6.09%	87.96%
intent_LAUNCHER	99.09%	99.44%	100.00%

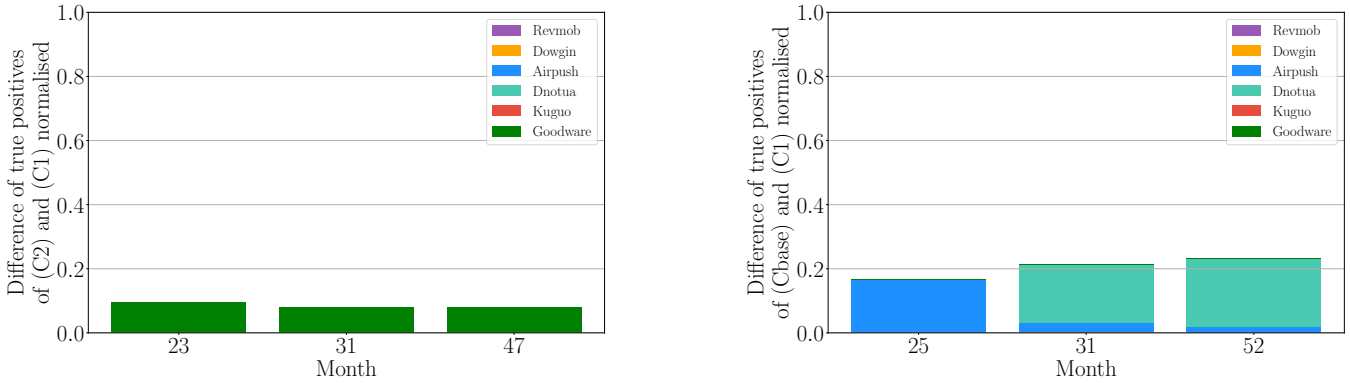
Table 3: Mean of weights of top-3 features of missed *Airpush* samples.

Feature	Weight C_{Base}	Weight C_1	Change
market.android.com	0.956	86.399	↑+9038.947 %
googletagmanager.com	0.785	21.608	↑+2751.322 %
play.google.com	12.84	14.161	↑ +110.287 %

Table 4: The appearance of top-3 *Airpush* features in Goodware, Malware and *Airpush* samples

Feature	Appearances		
	Malware	Goodware	Airpush
market.android.com	7.37%	0.74%	51.51%
googletagmanager	37.15%	39.21%	64.32%
play.google.com	10.2%	0.89%	72.74%

[2]https://blog.sucuri.net/2018/04/malicious-activities-google-tag-manager.html



(a) Goodware samples C_2 missed that C_1 captured.

(b) Malware family samples C_{Base} missed that C_1 captured.

Figure 4: Comparison of samples missed and captured in different oracle scenarios, to infer families which could have been captured with future knowledge of the dataset. All values are normalized to the total number of samples available in each month.

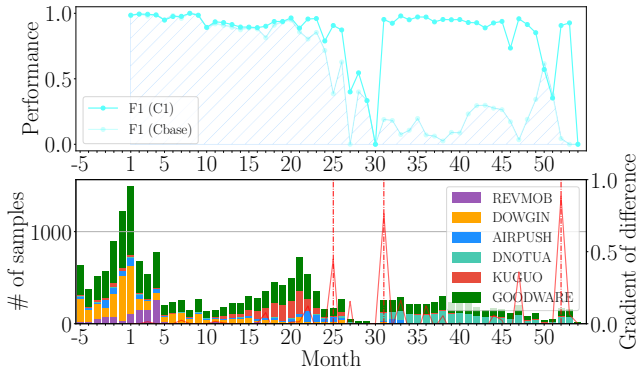


Figure 5: Oracle experiment on Transcendent dataset for POI selection. The vertical, red, dashed lines represent the points of interest with the three highest points of difference between the base classifier C_{Base} (without future knowledge) and the oracle C_1.

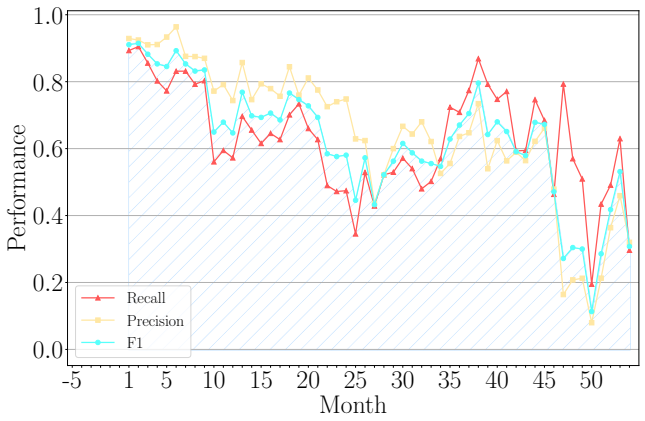


Figure 6: Classifier trained on first 6 months of Transcendent merged with *Dnotua* samples from month 31. This plot demonstrates that the forensic analysis identified the major cause of drift (cf. Figure 2(a)).

in 88.6% of the *Dnotua* samples and in 37.15% of the benign samples. This is shown in Table 2

The second feature *startService()* is also not a malicious feature but simply indicates that the app starts a service component at run-time. However, as malware commonly runs its malicious functionality as a service in the background, the model has identified this feature as an indicator for malware. The feature is present in 87.96% of the *Dnotua* samples and 6.09% of the benign applications.

A similar conclusion can be said for the *Airpush* family. Although there are more features that have an increase in mean weight, one particular feature stands out amongst the rest. Table 3 shows the top-3 features of missed *Airpush* samples. *market.android.com* appears in 51.51% of *Airpush* samples as shown in Table 7, and 97.37% of samples with this feature are from the *Airpush* family. Furthermore, 17.5% of all *Airpush* samples with this feature appear in month 25, the month where we see C_1 has a drastic increase in the detection of *Airpush*. Both the other two features *googletagmanager* and *play.google.com* have high presence in *Airpush* but a low appearance in the other malware families. This demonstrates the significance of these two *Airpush* features.

> The drift is substantially driven by two malware families—Dnotua and Airpush.

RQ3 — *How does the evolution of existing malware families affect the classifier's performance?* Although the emergence of *Dnotua* is the phenomenon that has one of the largest effect in terms of classifier performance, it is also important to investigate individual families—to understand the accuracy drop that may be induced by their evolution over time. To do this, we train a classifier for each family, which only has access to goodware samples

Table 5: Detection rate of training on one family (rows) and testing on rest (columns). This corresponds to the results in Figure 7. Each row corresponds to a different training family scenario. Each column reports the class-specific Recall assuming the column class as the "positive" target when computing TPs and FNs.

	Dowgin	Dnotua	Kuguo	Airpush	Revmob	Goodware
Dowgin	84.00%	1.12%	93.7%	0.97%	1.99%	97.87%
Dnotua	54.13%	98.72%	41.06%	16.55%	1.99%	96.22%
Kuguo	61.24%	1.17%	36.78%	18.99%	7.14%	96.77%
Airpush	2.92%	0.16%	2.96%	51.41%	3.04%	97.33%
Revmob	0.06%	0.00%	0.00%	11.3%	100.0%	99.64%

and malware samples of the respective family. These classifiers are trained on samples of the first 6 months and evaluated on all remaining months. Unfortunately, there are no samples of *Dnotua* in the first 6 months. Hence, we instead train on *Dnotua* samples from months 31-37 (i.e., the first 6 months where *Dnotua* predominantly appears in the dataset) and evaluate them on the remaining months. Figure 7 shows the class-specific Recall of each malware family if a classifier is trained solely on samples of the respective family. Table 5 reports the class-specific Recall values in each column reports the class-specific Recall assuming the column class as the "positive" target when computing TPs and FNs. For example, the first column reports the Recall of the Dowgin class, where $Recall_{Dowgin} = \frac{TP_{Dowgin}}{TP_{Dowgin}+FN_{Dowgin}}$. Each row corresponds to the malware family on which the classifier was trained on.

The first surprising observation is by training on the first 6 months of *Revmob* samples, a classifier is able to achieve perfect detection for all remaining *Revmob* samples. This indicates there is little malware evolution in *Revmob*. On the other hand, there seems to be a drop in prevalence for both *Airpush* and *Kuguo* samples. This suggests that both these families experience some form of malware evolution. Lastly, training solely on *Dnotua* results in near perfect detection for *Dnotua*, indicating that there is no significant evolution of this family that would impede its detection.

Simply observing the t-SNE plots gives us an intuition about the families that experience malware evolution. For example, *Dowgin* does not seem to evolve much whereas *Airpush* and *Kuguo* do. However, this analysis is not sufficient and can only give surface-level intuitions. Hence, we also analyze the top-5 important features of the classifier for all families. Note that due to certain months not containing any samples of a particular malware family, we are unable to directly use the POIs we identified previously. Instead, we sort the POIs in descending value and pick the first two where there are malware samples of that family present, and in addition consider the first month of appearance. This is commonly month 1 with the exception of *Dnotua* being month 31. This way we focus on the most impactful months that are available for analysis. Here, we present interesting results from two families, *Revmob* and *Airpush*; for completeness, we report results of all other families in Appendix B.

By examining the top-5 features of *Revmob* shown in Table 6, we notice that the *revmob.com* and *android.revmob.com* is consistently the most important feature for the classification of this family at different months. This explains why we do not observe any drift

Table 6: Top-5 explanations for *Revmob* samples at different months. We observe that the two most important features remain consistent over time, which motivates the high detection accuracy over time.

#	Month 1	Month 25	Month 31
1	revmob.com	revmob.com	revmob.com
2	android.revmob.com	android.revmob.com	android.revmob.com
3	GET_ACCOUNTS	PERMS_SHORTCUT	googletagmanager.com
4	INTENT_LAUNCHER	INTENT_LAUNCHER	Cipher(DES)
5	INTENT_PACKAGE	googleapis.com	startService

Table 7: Top-5 explanations for *Airpush* samples at different months. We see that most features change from the starting month, hence this motivates the drop in accuracy and the in-family evolution of *Airpush*.

#	Month 1	Month 31	Month 52
1	schemas.android.resauto	market.android.com	market.android.com
2	PERMS_GET_ACCOUNTS	INTENT_PACKAGE	googletagmanager
3	INTENT_LAUNCHER	googletagmanager.com	schemas.android.resauto
4	revmob.com	apportal.airpush.com	startService
5	play.google.com.com	schemas.android.resauto	apportal.airpush.com

in the *Revmob* family as all samples of this family will contain this URL. In our dataset, *revmob.com* and *android.revmob.com* appear in *Revmob* samples 11.27% and 11.19% times respectively.

If we turn our attention to *Airpush* samples in Table 7, we notice that the top features changed in month 52 and month 31 compared to month 1. This indicates the classifier required different features to classify *Airpush* samples. This further confirms our analysis in the previous steps, explaining why *Airpush* was not detected well in later months.

> Malware evolution is apparent in some families and can significantly affect a classifier's performance.

4 LIMITATIONS AND DISCUSSIONS

Transferability. We have shown that our framework is capable of helping a practitioner in detecting and evaluating a dataset that is affected by concept drift. However, there are still some limitations that need to be addressed. In our experiments, we have demonstrated our framework to work well in the mobile malware domain. We expect this to be easily transferable to other domains due to minimal reliance on the intrinsic features of Android Malware.

Ground truth labels. Our current framework relies on the availability of ground truth labels for individual families every month. This is similar to Chen et al. [13] where they assume a monthly labeling budget for active learning. Although this is an appropriate assumption for a post-hoc forensics framework, ground truth labels are scarce in real-world deployments. For this reason, we plan to extend our work by reducing the reliance on ground truth labels so our framework can be used in a semi-supervised setting.

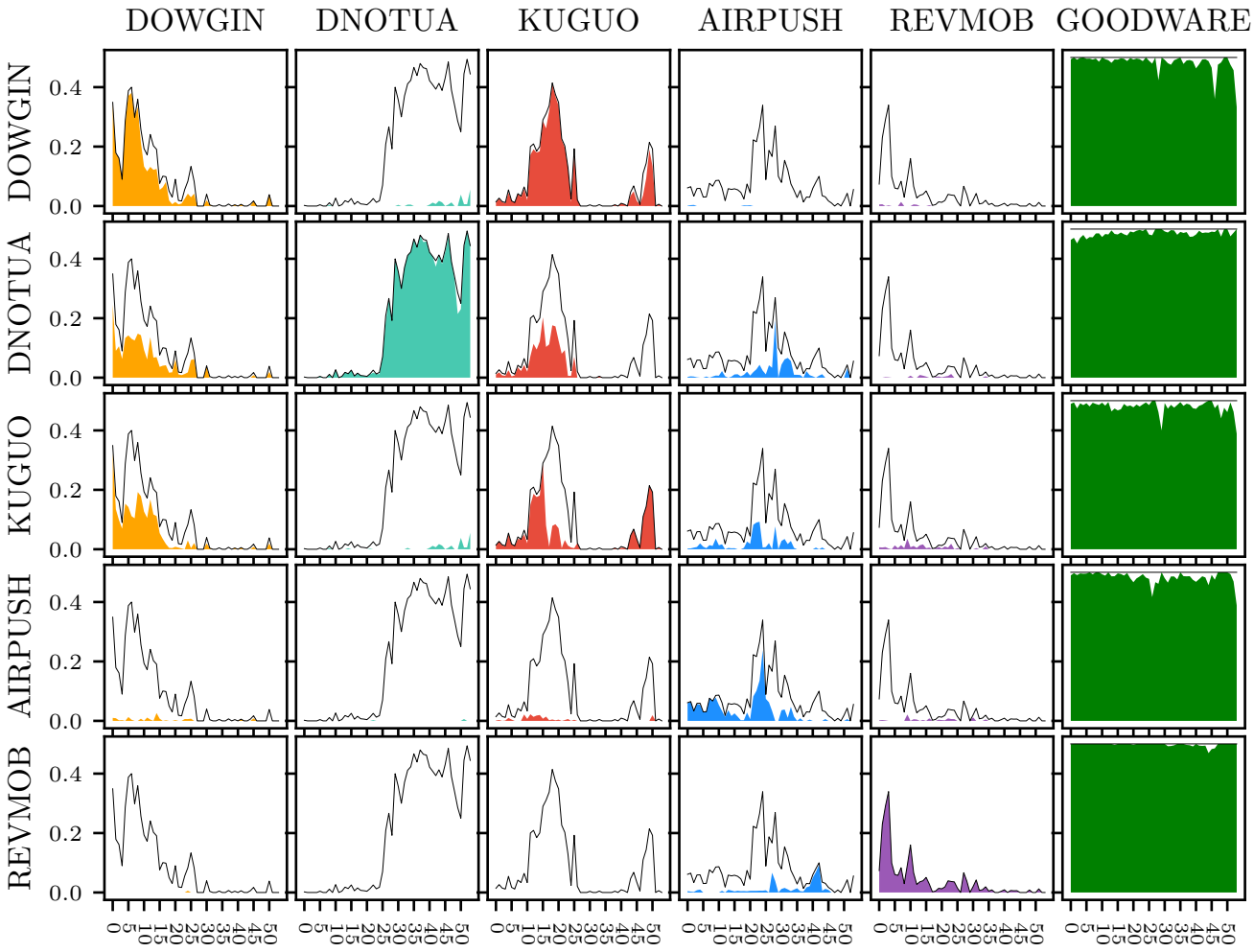


Figure 7: Classifier trained on one family (row) and tested on all families (columns). All families are trained using the first 6 months starting from the time of their first appearance, and tested on the remaining months. The x-axis tracks each month of the test set. The shaded region in each figure represents the recall and the black line total samples of test families.

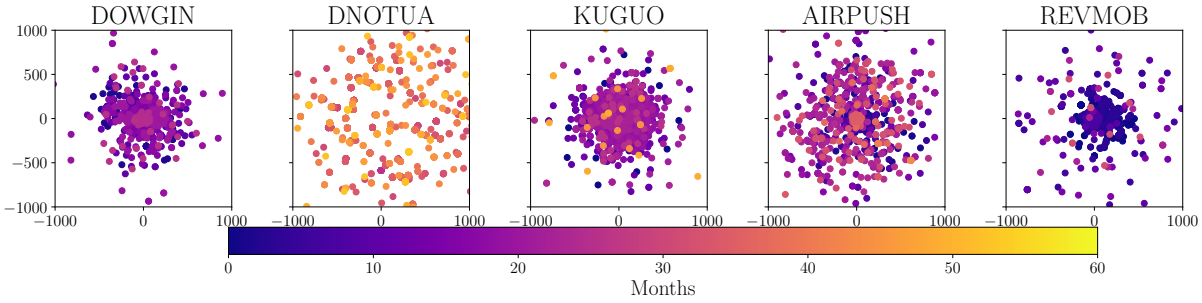


Figure 8: t-SNE plot of malware family samples over time. A different color corresponds to malware samples taken from a different month. We observe that there is little malware evolution for *Dowgin* samples compared to *Dnotua* and *Airpush*.

Focus on top-5 families. Restricting the analysis to only the top-5 family is reasonable, but it ignores trends of smaller families. Although we identified the main root cause of drift for the Transcendent dataset [11], future work should investigate better selection criteria for the families to analyze.

Classifier and Explanations. We instantiated our framework on a Linear SVM model with Gradients × Input explanations, with the Drebin [8] binary feature space. Due to the simplicity of both the classifier, the explanations, and the Drebin feature space, it is feasible to retrain the model and compare the results. However, in more complex classifiers (e.g., representation learning) features and explanations may not be easily interpretable, and retraining multiple times may become computationally unfeasible. Hence, future work should understand how to address and mitigate these challenges towards effective forensics tasks.

Other sources of bias. Although our framework is capable of identifying the root causes of concept drift in a dataset, it is heavily dependent on two factors. The accuracy of labeled samples and whether the samples collected reflect the true data distribution of the underlying security problem. As pointed out by Arp et al. [7], these are two of the ten common pitfalls that machine learning research tends to fall into. By analysing concept drift in a dataset using our framework, we in fact put them under a magnifying glass and try to explain a certain behaviour that is observed due to bias in data collection. Instead of falling into these pitfalls, our framework helps identify potential biases during data collection that a practitioner may have overlooked.

5 RELATED WORK

The work presented in this paper touches on different research fields by using Explainable AI (XAI) methods to analyze concept drift in the mobile malware domain. In the following, we provide a brief overview of the state of the art in these fields.

Explainable AI. Explainable AI (XAI) methods [36] attempt to make the decision process of machine learning models interpretable for humans [27]. They are often classified into *global* and *local* methods, with the former aiming to explain a whole model and the latter attempting to explain a model's decision for one particular data point. An orthogonal axis of classification is *ante-hoc* versus *post-hoc* methods. While ante-hoc XAI methods involve creating models that are explainable by design, post-hoc methods are applicable to more general model classes and are not involved in the training process.

In the past, many XAI methods have been proposed [36, 49]. In this work, we will make use of local post-hoc methods, in particular, attribution methods which assign a scalar importance value to each dimension of an input sample, indicating its contribution to the classification decision. This choice has been made since we attempt to create a general framework that does not hinge on a specific (ante-hoc) explanation method and model, and at the same time allows us to explain classification decisions for individual samples or groups of samples.

The natural choice for explaining linear classifiers on binary features is Gradient × Input [39] and we will make use of this method throughout the paper. It should be noted though that our proposed framework is not restricted to this setting and can easily be extended to more sophisticated models, such as deep neural networks, using XAI methods such as IG [40] or LRP [10, 31]. In fact, Gradient × Input has been proven to be a special case of LRP [6].

Concept Drift Analysis. In recent years, various researchers have started to explore the effect of different variants of distributional shift on learning-based detection methods [21, 32], including the security domain [9, 28, 35, 46, 47]. One of the early works that highlight problems due to concept drift in the mobile malware domain has been presented by Allix et al. [4]. Other researchers later confirmed and explored these findings further, demonstrating that even the detection performance of many state-of-the-art methods is affected by concept drift [23, 35].

Follow-up research thus proposed various approaches to alleviate the impact of concept drift on the classifier's performance, including methods for drift detection [e.g., 11, 22, 23] and online learning [13, 33, 35, 45]. While these methods compensate for the performance drop to some extent, none of them aims to analyze the root causes of the underlying drift.

The two most similar works in this direction have been done by Yang et al. [47] and Chen et al. [14]. Yang et al. [47] proposes a method for detecting and explaining individual drifting samples. However, the authors only examine artificially balanced datasets that solely contain malware. Our work extends this line of research by providing in-depth insights into a popular Android malware that is commonly used to explore concept drift in real-world settings [e.g., 11, 26].

6 CONCLUSION

In this paper, we have presented a post-hoc framework for performing a detailed analysis of concept drift for Android malware datasets. Our framework allows practitioners to automatically identify a small number of Points of Interest (POI). These POI determine the root cause of drift, from the associated malware to the precise feature (if explainable) causing such drift. In our settings, drift was mostly caused by a malware family (although which one or whether more were present is irrelevant to the problem at hand). This is either because of the actual evolution of malware behaviors, imprecise abstractions and representations of the datasets, or a combination thereof, which calls for further research on drift detection [11, 13, 25, 35, 47] and programs' representations.

We encourage researchers and practitioners to build on our framework to explore the root causes of distribution and concept drift in other domains, fostering new and exciting avenues toward a better understanding of one of the core challenges learning-based algorithms must face.

ACKNOWLEDGMENTS

The authors gratefully acknowledge funding from the German Federal Ministry of Education and Research (BMBF) under the grant BIFOLD23B and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under the projects 456292433; 456292463. This work has also been partially supported by the King's-China Scholarship Council Ph.D. Scholarship programme (K-CSC), a Google ASPIRE research award, and EPSRC Grant EP/X015971/1.

REFERENCES

[1] [n. d.]. Adware Dowgin. https://vms.drweb.com/virus/?i=21714828. Accessed: 2023-07-06.

[2] [n. d.]. Adware Kuguo. https://vms.drweb.com/virus/?i=17938587. Accessed: 2023-07-06.

[3] [n. d.]. Message Digest class. https://developer.android.com/reference/java/security/MessageDigest. Accessed: 2023-07-06.

[4] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2015. Are your training datasets yet relevant?. In *International Symposium on Engineering Secure Software and Systems*. Springer, 51–67.

[5] Brandon Amos, Hamilton Turner, and Jules White. 2013. Applying machine learning classifiers to dynamic android malware detection at scale. In *2013 9th international wireless communications and mobile computing conference (IWCMC)*. IEEE, 1666–1671.

[6] Marco Ancona, Enea Ceolini, Cengiz Öztireli, and Markus H. Gross. 2017. Towards better understanding of gradient-based attribution methods for Deep Neural Networks. In *International Conference on Learning Representations*.

[7] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2022. Dos and Don'ts of Machine Learning in Computer Security. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 3971–3988. https://www.usenix.org/conference/usenixsecurity22/presentation/arp

[8] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, and Konrad Rieck. 2014. Drebin: Effective and explainable detection of android malware in your pocket.. In *Proc. of the Network and Distributed System Security Symposium (NDSS)*, Vol. 14. 23–26.

[9] Erin Avllazagaj, Ziyun Zhu, Leyla Bilge, Davide Balzarotti, and Tudor Dumitras. 2021. When Malware Changed Its Mind: An Empirical Study of Variable Program Behaviors in the Real World. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 3487–3504. https://www.usenix.org/conference/usenixsecurity21/presentation/avllazagaj

[10] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. 2015. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one* 10, 7 (2015), e0130140.

[11] Federico Barbero, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. 2022. Transcending TRANSCEND: Revisiting Malware Classification in the Presence of Concept Drift. In *Proc. of the IEEE Symposium on Security and Privacy (S&P)*. IEEE. https://doi.org/10.1109/SP46214.2022.9833659

[12] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. 1992. A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. Association for Computing Machinery, New York, NY, USA, 144–152. https://doi.org/10.1145/130385.130401

[13] Yizheng Chen, Zhoujie Ding, and David Wagner. 2023. Continuous Learning for Android Malware Detection. arXiv:2302.04332 [cs.CR]

[14] Zhi Chen, Zhenning Zhang, Zeliang Kan, Jacopo Cortellazzi, Feargus Pendlebury, Fabio Pierazzi, Lorenzo Cavallaro, and Gang Wang. 2023. *Is It Overkill? Analyzing Feature-Space Concept Drift in Malware Detectors* (2023 ed.). IEEE.

[15] Nello Cristianini and John Shawe-Taylor. 2000. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods* (1 ed.). Cambridge University Press.

[16] Stefan Decker. [n. d.]. G DATA Mobile Malware Report: Criminals keep up the pace with Android malware. https://www.gdatasoftware.com/news/2021/10/37093-g-data-mobile-malware-report-criminals-keep-up-the-pace-with-android-malware. Accessed: 2023-06-19.

[17] Jaka Demšar and Zoran Bosnić. 2018. Detecting concept drift in data streams using model explanation. *Expert Systems with Applications* 92 (2018), 546–559.

[18] Dan Goodin. [n. d.]. Potentially millions of Android TVs and phones come with malware preinstalled. https://arstechnica.com/information-technology/2023/05/potentially-millions-of-android-tvs-and-phones-come-with-malware-preinstalled/. Accessed: 2023-06-239.

[19] Arash Habibi Lashkari Gurdip Kaur. [n. d.]. Understanding Android malware Families: Riskware - is it worth it? https://www.itworldcanada.com/blog/understanding-android-malware-families-riskware-is-it-worth-it-article-4/446692. Accessed: 2023-06-20.

[20] Joachim Herz, Dudley K Strickland, et al. 2001. LRP: a multifunctional scavenger and signaling receptor. *The Journal of clinical investigation* 108, 6 (2001), 779–784.

[21] T Ryan Hoens, Robi Polikar, and Nitesh V Chawla. 2012. Learning from streaming data with concept drift and imbalance: an overview. *Progress in Artificial Intelligence* 1, 1 (2012), 89–101.

[22] Cheng-Yu Hsieh, Chih-Kuan Yeh, Xuanqing Liu, Pradeep Ravikumar, Seungyeon Kim, Sanjiv Kumar, and Cho-Jui Hsieh. 2020. Evaluations and methods for explanation through robustness analysis. *arXiv preprint arXiv:2006.00442* (2020).

[23] Roberto Jordaney, Kumar Sharad, Santanu K Dash, Zhi Wang, Davide Papini, Ilia Nouretdinov, and Lorenzo Cavallaro. 2017. Transcend: Detecting concept drift in malware classification models. In *26th USENIX security symposium (USENIX security 17)*. 625–642.

[24] Mina Esmail Zadeh Nojoo Kambar, Armin Esmaeilzadeh, Yoohwan Kim, and Kazem Taghva. 2022. A survey on mobile malware detection methods using machine learning. In *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 0215–0221.

[25] Zeliang Kan, Feargus Pendlebury, Fabio Pierazzi, and Lorenzo Cavallaro. 2021. Investigating Labelless Drift Adaptation for Malware Detection. In *ACM Workshop on Artificial Intelligence and Security (AISec)*.

[26] Deqiang Li, Tian Qiu, Shuo Chen, Qianmu Li, and Shouhuai Xu. 2021. Can We Leverage Predictive Uncertainty to Detect Dataset Shift and Adversarial Examples in Android Malware Detection?. In *Proc. of the Annual Computer Security Applications Conference (ACSAC)*. https://doi.org/10.1145/3485832.3485916

[27] Zachary C Lipton. 2018. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue* 16, 3 (2018), 31–57.

[28] Federico Maggi, William Robertson, Christopher Kruegel, and Giovanni Vigna. [n. d.]. Protecting a Moving Target: Addressing Web Application Concept Drift. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID)*. https://doi.org/10.1007/978-3-642-04342-0_2

[29] Francesco Mercaldo and Antonella Santone. 2020. Deep learning for image-based mobile malware detection. *Journal of Computer Virology and Hacking Techniques* 16, 2 (2020), 157–171.

[30] Michael Mimoso. [n. d.]. Gunpoder Android Malware Hides Malicious Behaviors in Adware. https://threatpost.com/gunpoder-android-malware-hides-malicious-behaviors-in-adware/113654/. Accessed: 2023-06-19.

[31] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. 2019. Layer-wise relevance propagation: an overview. *Explainable AI: interpreting, explaining and visualizing deep learning* (2019), 193–209.

[32] Jose G Moreno-Torres, Troy Raeder, Rocío Alaiz-Rodríguez, Nitesh V Chawla, and Francisco Herrera. 2012. A unifying view on dataset shift in classification. *Pattern recognition* 45, 1 (2012), 521–530.

[33] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, and Yang Liu. 2017. Context-aware, adaptive, and scalable android malware detection through online learning. *IEEE Transactions on Emerging Topics in Computational Intelligence* 1, 3 (2017), 157–175.

[34] Fairuz Amalina Narudin, Ali Feizollah, Nor Badrul Anuar, and Abdullah Gani. 2016. Evaluation of machine learning classifiers for mobile malware detection. *Soft Computing* 20 (2016), 343–357.

[35] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, Lorenzo Cavallaro, et al. 2019. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *Proceedings of the 28th USENIX Security Symposium*. USENIX Association, 729–746.

[36] Wojciech Samek, Grégoire Montavon, Andrea Vedaldi, Lars Kai Hansen, and Klaus-Robert Müller. 2019. *Explainable AI: interpreting, explaining and visualizing deep learning*. Vol. 11700. Springer Nature.

[37] Silvia Sebastián and Juan Caballero. 2020. AVclass2: Massive Malware Tag Extraction from AV Labels. In *Annual Computer Security Applications Conference*. Association for Computing Machinery, New York, NY, USA, 42–53. https://doi.org/10.1145/3427228.3427261

[38] Shawn Shan, Arjun Nitin Bhagoji, Haitao Zheng, and Ben Y Zhao. 2022. Poison forensics: Traceback of data poisoning attacks in neural networks. In *31st USENIX Security Symposium (USENIX Security 22)*. 3575–3592.

[39] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning Important Features through Propagating Activation Differences. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70* (Sydney, NSW, Australia) *(ICML'17)*. JMLR.org, 3145–3153.

[40] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International conference on machine learning*. PMLR, 3319–3328.

[41] Roman Unuchek. [n. d.]. Mobile malware evolution. https://securelist.com/mobile-malware-evolution-2016/77681/. Accessed: 2022-05-22.

[42] Zhiqiang Wang, Qian Liu, and Yaping Chi. 2020. Review of android malware detection based on deep learning. *IEEE Access* 8 (2020), 181102–181126.

[43] Alexander Warnecke, Daniel Arp, Christian Wressnegger, and Konrad Rieck. 2020. Evaluating explanation methods for deep learning in security. In *2020 IEEE european symposium on security and privacy (EuroS&P)*. IEEE, 158–174.

[44] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. 2017. Deep ground truth analysis of current android malware. In *Detection of Intrusions and Malware, and Vulnerability Assessment: 14th International Conference, DIMVA 2017, Bonn, Germany, July 6-7, 2017, Proceedings 14*. Springer, 252–276.

[45] Ke Xu, Yingjiu Li, Robert Deng, Kai Chen, and Jiayun Xu. 2019. Droidevolver: Self-evolving android malware detection system. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 47–62.

[46] Limin Yang, Arridhana Ciptadi, Ihar Laziuk, Ali Ahmadzadeh, and Gang Wang. 2021. BODMAS: An Open Dataset for Learning based Temporal Analysis of PE Malware. In *2021 IEEE Security and Privacy Workshops (SPW)*. 78–84. https://doi.org/10.1109/SPW53761.2021.00020

[47] Limin Yang, Wenbo Guo, Qingying Hao, Arridhana Ciptadi, Ali Ahmadzadeh, Xinyu Xing, and Gang Wang. 2021. {CADE}: Detecting and Explaining Concept

Drift Samples for Security Applications. In *Proc. of the USENIX Security Symposium*. https://www.usenix.org/conference/usenixsecurity21/presentation/yang-limin

[48] Weikai Yang, Zhen Li, Mengchen Liu, Yafeng Lu, Kelei Cao, Ross Maciejewski, and Shixia Liu. 2020. Diagnosing concept drift with visual analytics. In *2020 IEEE conference on visual analytics science and technology (VAST)*. IEEE, 12–23.

[49] Yu Zhang, Peter Tiňo, Aleš Leonardis, and Ke Tang. 2021. A survey on neural network interpretability. *IEEE Transactions on Emerging Topics in Computational Intelligence* 5, 5 (2021), 726–742.

[50] Francesco Zola, Jan Lukas Bruse, and Mikel Galar. 2023. Temporal Analysis of Distribution Shifts in Malware Classification for Digital Forensics. In *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE Computer Society, 439–450.

A TOP-5 MALWARE FAMILIES

We collect information about the selected families online and on technical reports [1–3, 19, 30], to later verify the explanations we gather during our analysis. In the following, we provide a brief description of each family in our dataset.

Dowgin. Anti-virus vendors report that this malware tries to display advertisements on top of other apps. Thus, it is mainly considered as adware. In addition, some variants of this malware also attempt to access audio and video recording interfaces, gather information about the network, phone status and the apps installed on the device [1].

Dnotua. Dnotua may utilize system resources in an undesirable or annoying manner that may pose a security risk, such as stealing network information or asking for root privileges. Adversaries may also use Dnotua to install other apps that are malicious, keeping Dnotua undetected by Antivirus scanners [19]. One unique feature common in Dnotua samples is it updates MessageDigest, which is the class in Android for encrypting messages [3]. Dnotua is typically classified as a riskware.

Airpush. Airpush is an advertising library that has been misused by malicious apps to push fraudulent advertisements to the user or even hide its malicious code inside the library [30]. Furthermore, apps that use older versions of the library are also considered malicious by many antivirus engines, as they use push notifications for advertisements—which is against the Google guidelines (see https://dottech.org/75309/google-bans-notification-bar-ads-such-as-airpush-from-play-store/).

Kuguo. Kuguo is an adware that is known to be very similar to Dowgin. One unique feature of this family is its access to the ITelephony private interface that controls voice and video calls on an Android device. Since this is similar to Dowgin, we expect it to exhibit similar behaviour and not affect detection performance drastically [2].

Revmob. Another Adware, Revmob collects the personal information and browser history of the victim and redirects victims to malicious websites [19]. Furthermore, Revmob will display obnoxious ads and exhibit similar behaviour to Airpush.

B TOP-5 FEATURES OF A CLASSIFIER TRAINED ON ONE FAMILY

We present the top-5 features of a classifier trained on one family and tested on the remaining families.

We would have expected *Kuguo* to have drastic changes in the ranking of feature importance. Although the changes are surprisingly small, we notice that the top features do change slightly between months 1 and 25. This confirms are suspicion of malware evolution that is evident from Figure 7. Furthermore, the previous information collected about *Kuguo*'s behavior and its similarity to *Dowgin* lets us reason that *Kuguo* heavily relies on *Dowgin* for classification. The top features of Table 8 and Table 9 are also very similar.

Table 8: Top-5 explanations for *Kuguo.*

#	Month 1	Month 22	Month 25
1	INTENT_USER	INTENT_USER	INTENT_USER
2	INTENT_PACKAGE	PERMS_SHORTCUT	com.baidu.AppActivity
3	PERMS_LAUNCHER	INTENT_LAUNCHER	PERMS_SHORTCUT
4	INTENT_SHORTCUT	API:AudioManager	INTENT_LAUNCHER
5	INTENT_LAUNCHER	INTENT_PACKAGE	com.downloadservice

Dowgin's top features stay relatively consistent over month 1 to 25, and changes for month 31. This suggest that there is little malware evolution of this family. *Dowgin* has a high class-specific recall as shown in Figure 7, Hence this result is expected. The change of top features also aligns well with the drop in performance at month 31.

Table 9: Top-5 explanations for *Dowgin*

#	Month 1	Month 25	Month 31
1	INTENT_USER	INTENT_PACKAGE	INTENT_USER
2	INTENT_PACKAGE	INTENT_PACKAGE	com.qihoo.util
3	Cipher(DES)	PERMS_SHORTCUT	PERMS_SHORTCUT
4	PERMS_SHORTCUT	CipherDES	INTENT_LAUNCHER
5	INTENT_SHORTCUT	INTENT_LAUNCHER	com.unity3d

For *Dnotua*, We notice that the features stay consistent throughout months 31 to month 52. We reason this is the reason why Dnotua does not experience large drop in class-specific recall. This can be seen in Table 10.

Table 10: Top-5 explanations for *Dnotua*

#	Month 31	Month 47	Month 52
1	googletagmanager.com	googletagmanager.com	googletagmanager.com
2	startService	startService	startService
3	INTENT_LAUNCHER	INTENT_LAUNCHER	INTENT_USER
4	PERMS_INTERNET	PERMS_INTERNET	PERMS_INTERNET
5	activities:Mustach	INTENT_USER	activities:Mustach