# The Impact of Active Learning on Availability Data Poisoning for Android Malware Classifiers

Shae McFadden[†§‡], Zeliang Kan[†‡], Lorenzo Cavallaro[‡], Fabio Pierazzi[‡†]
[†]*King's College London,* [‡]*University College London,* [§]*The Alan Turing Institute*

***Abstract*—Can a poisoned machine learning (ML) model passively recover from its adversarial manipulation by retraining with new samples, and regain non-poisoned performance? And if passive recovery is possible, how can it be quantified? From an adversarial perspective, is a small amount of poisoning sufficient to force the defender to retrain more over time?**

**This paper proposes the evaluation of passive recovery from "availability data poisoning" using active learning in the context of Android malware detection. To quantify passive recovery, we propose two metrics: *intercept* to assess the speed of recovery, and *recovery rate* to quantify the stability of recovery. To investigate passive recovery, we conduct our experiments at different rates of active learning, in conjunction with varying strengths of availability data poisoning. We perform our evaluation on 259,230 applications from AndroZoo, using the DREBIN feature representation, with linear SVM, DNN, and Random Forest as classifiers. Our findings show the convergence of the poisoned models to their respective hypothetical non-poisoned models. Therefore, demonstrating that through the use of active learning as a concept drift mitigation strategy, passive recovery is feasible across the three classifiers evaluated.**

***Index Terms*—supervised learning, malware detection, poisoning, active learning, passive recovery.**

## 1. Introduction

Malware evolution and challenges in abstracting malware semantics affect the performance of learning-based algorithms over time, due to non-stationarity and concept drift [1], [2], [3], [4]. Therefore, it has become common practice to mitigate concept drift by retraining or updating models whenever new ground-truth labels become available. The quality of new ground-truth labels can be challenged by label instability [5], the use of unvetted datasets, and the intensive costs of labeling whenever pseudo-labels cannot be trusted [6]. A questionable ground-truth may, unfortunately, facilitate data poisoning [7], [8]. The literature on data and backdoor poisoning attacks is abundant in computer vision tasks, and recent work shows that the same threat also applies to malware classifiers [7], [8]. Current defenses aim to identify poisoned models [9], [10], or trace data poisoning attacks in specific settings under a forensic lens [11], [12]. However, understanding the temporal effects of concept drift mitigation strategies on data poisoning attacks in non-stationary contexts remains largely unexplored.

This paper investigates whether and how concept drift mitigation strategies help to passively recover model compromise induced by availability data poisoning attacks.[1] We propose the formalization of "*recovery*" as the convergence of the *poisoned model* performance with that of the hypothetical performance of the same model if it were not poisoned (which we refer to as the *hypothetical model*). We distinguish *passive recovery* as the recovery process that results as a by-product of the complete classification system, as opposed to *active recovery* that specifically seeks to mitigate the impact of poisoning through directed methods. We quantify passive recovery by introducing two new metrics: *intercept* and *recovery rate*. The intercept aims to determine *when* the performance of the poisoned model begins to converge with the performance of the hypothetical model. Since model performance exhibits fluctuations in non-stationary contexts [1], [2], the recovery rate aims to quantify the percentage of time that the poisoned model has a performance equal to or greater than the hypothetical model. Since exact recovery is unlikely, we introduce the concept of the *tolerance margin*, which defines *how much* approximation is tolerated when assuming that the model has at least the same performance as the hypothetical model. Then, we consolidate these metrics and methodologies to propose RPAL, an extensible framework to aid in the investigation of passive recovery for other ML classification systems given a set of parameters, classifiers, concept drift mitigations and poisoning strategies.

We evaluate RPAL in the context of Android malware detection to quantify passive recovery from dirty label (label-flip) and clean label (feature-flip) poisoning by using active learning (uncertainty sampling). We used the 5-year dataset from Barbero et al. [2], additionally used by Kan et al. [14], which is crawled from AndroZoo [15]. The dataset adopts the DREBIN [16], [1], [17] feature space, one of the most popular feature representations. This dataset allows for comparability with previous temporal-evaluation work, as well as allows us to evaluate the stability of passive recovery over a four-year test period. In our evaluation, we investigate the following three research questions. RQ-INTERCEPT investigates the impact of active learning and poisoning rates on the intercept, and shows the diminishing

1. This paper extends our previously published poster [13].

returns of increasing active learning rates against availability data poisoning during passive recovery. RQ-RECOVERY-RATE evaluates the overall stability of the recovery rate for passive recovery after the intercept, and shows that—when comparing across all settings—the classifiers evaluated are capable of passive recovery. RQ-CLASSIFIER determines the impact of how different classifiers using the same features affect passive recovery, and shows that—despite all classifiers being capable of passive recovery—their $F_1$ score and overall performance are affected by the classifier chosen.

Overall, we make the following key contributions:

- We identify passive recovery as a hidden factor that would impact the performance of an availability data poisoning attack in practice on a real system.
- We discover the trend of convergence in performance between poisoned models and their hypothetical non-poisoned counterparts.
- We create an extensible evaluation framework, RPAL, through which we demonstrate the feasibility of passive recovery from availability data poisoning in the context of Android malware detection. We openly release the code to foster future research (see Section 7).

**Distinction between Passive and Active Recovery.** We define *recovery* as the act of converging the performance of a poisoned model with that of the hypothetical model, which was never poisoned. Specifically, *passive recovery* refers to recovery achieved by an approach which is implemented for another purpose such as uncertainty sampling active learning for concept drift mitigation in the case of this paper. In contrast, the *active recovery* common in the existing literature refers to recovery achieved by an approach specifically designed to reduce or eliminate the effects of data poisoning. We focus on passive recovery because the temporal effects of passive approaches on classifiers are overlooked in existing poisoning research. To our knowledge, we are the first to evaluate the passive recovery of traditional ML and DNN systems from availability data poisoning in the context of malware detection.

## 2. Passive Recovery

The evaluation of passive recovery requires the selection of a classification system that contains the candidate passive recovery mechanism that is being evaluated. Any candidate passive recovery mechanism must have the ability to adapt or modify the model throughout a time-aware evaluation and remain within realistic operational constraints—such as a labeling budget. Subsequently, the poisoning strategy selected is dependent on the threat model of the system using the selected passive recovery strategy.

### 2.1. Parameters and Metrics

In this research, we define recovery as the convergence of the poisoned model performance with that of the hypothetical performance of the same model if it were not poisoned, which we refer to as the *hypothetical model*. In the remainder of this section, we provide the definitions and descriptions of the parameters (tolerance margin) and metrics (intercept and recovery rate) required for the passive recovery evaluation. A visual example can be found in Figure 1. Although this research focuses on the evaluation of a passive recovery strategy, the proposed metrics can also be used to evaluate the speed and stability of recovery over time for other approaches.

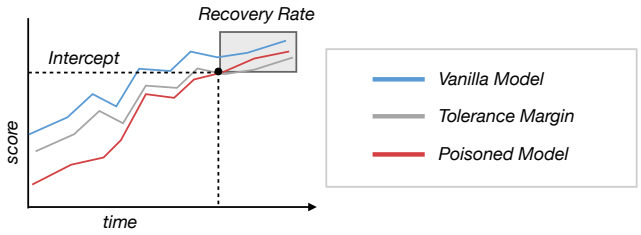



Figure 1: A visual example of the tolerance margin as well as the intercept and recovery rate metrics described.

**Notation.** Model updates occur periodically according to a predefined temporal granularity, with a monthly granularity being used in this research. $\tau$ and $N$ represent the current and total number of updates, respectively. $\mathcal{M}_{\mathcal{P}}$ and $\mathcal{M}$ represent the poisoned model and its respective hypothetical model with $f_{\mathcal{M}_{\mathcal{P}}}$ and $f_{\mathcal{M}}$ as the decision functions of the models. $X_\tau$ and $y_\tau$ represent the feature vectors and ground-truth labels of the samples within the period $\tau$, respectively. $\mathbb{P}$ is a performance metric function that takes the predicted labels from $f_{\mathcal{M}_{\mathcal{P}}}(X_\tau)$ and the ground-truth labels $y_\tau$. We use $F_1$-Score ($F_1 = \frac{2\times TP}{2\times TP+FP+FN}$) as our performance metric in this research therefore, $\mathbb{P}(f_{\mathcal{M}_{\mathcal{P}}}(X_\tau), y_\tau)$ computes the $F_1$-Score for the samples in period $\tau$.

**Definition 2.1** (Tolerance Margin)**.** The tolerance margin $\delta$ ($\delta \in \mathbb{R}, 0 \leq \delta \leq 1$) is a hyperparameter of the recovery evaluation that is subtracted from the performance of the hypothetical model $\mathcal{M}$, such that $\mathbb{P}(f_{\mathcal{M}_{\mathcal{P}}}(X_\tau), y_\tau)$ is compared against $\mathbb{P}(f_{\mathcal{M}}(X_\tau), y_\tau) - \delta$.

The tolerance margin represents how strict the definition of recovery is in the evaluation. A tolerance margin of $0.05$ will report better recovery results, compared to a tolerance margin of $0.01$, but will tolerate more errors. Visually, this means that the performance of the poisoned model must overcome a "lower version" of the performance of the hypothetical model.

**Definition 2.2** (Intercept)**.** Given a tolerance margin $\delta$, the intercept $I$ is a metric defined as the first period $\tau$ where:

$$\mathbb{P}(f_{\mathcal{M}_{\mathcal{P}}}(X_\tau), y_\tau) \geq \mathbb{P}(f_{\mathcal{M}}(X_\tau), y_\tau) - \delta$$

The intercept reports the month in which a model is initially deemed recovered. However, it is insufficient on its own as performance can fluctuate over time due to statistical noise, residual poisoning, and residual drift.

**Definition 2.3** (Recovery Rate)**.** The recovery rate metric $R$ is defined as:

$$\frac{1}{N-I}\sum_{\tau=I}^{N}\begin{cases}1, & \mathbb{P}(f_{\mathcal{M}_{\mathcal{P}}}(X_\tau), y_\tau) \geq \mathbb{P}(f_{\mathcal{M}}(X_\tau), y_\tau) - \delta,\\ 0, & \text{otherwise.}\end{cases}$$

The recovery rate reports the stability of recovery, as it is the percentage of months in which the model maintained recovery after the initial intercept $I$.

**Comparing Recovery.** To compare the recovery of two models for a specific set of settings, both the intercept rate and the recovery rate should be considered. The results of those direct comparisons are then used to deem one of the models superior or that it is a mixed result.

- *Intercept Comparison*: Lower intercepts are better as the model reaches the tolerance margin earlier.
- *Recovery Rate Comparison*: Higher recovery rates are better, as they imply that the model maintains recovered performance more consistently.

To deem one of the models superior or for it to be a mixed result, the aforementioned comparisons are then evaluated together. The distinction between superior recovery performance and a mixed result is decided as follows.

- *Superior Recovery*: a model's recovery is deemed superior to another model's recovery if either both metrics (intercept and recovery rate) are (i) strictly better than the other model's metrics, or (ii) one metric is better and the other metric is equal to the other model's.
- *Mixed Result*: the comparison is deemed mixed if (i) both metrics are equal, or (ii) each model is better in one of the two metrics.

To the best of our knowledge, the metrics defined in this subsection are the first temporally-aware poisoning recovery metrics for malware detection.

## 2.2. Classification System

This subsection presents the time-aware evaluation and classification systems that we used in our passive recovery evaluation. This base system is then augmented with the passive recovery mechanism and poisoning strategy discussed in subsequent subsections.

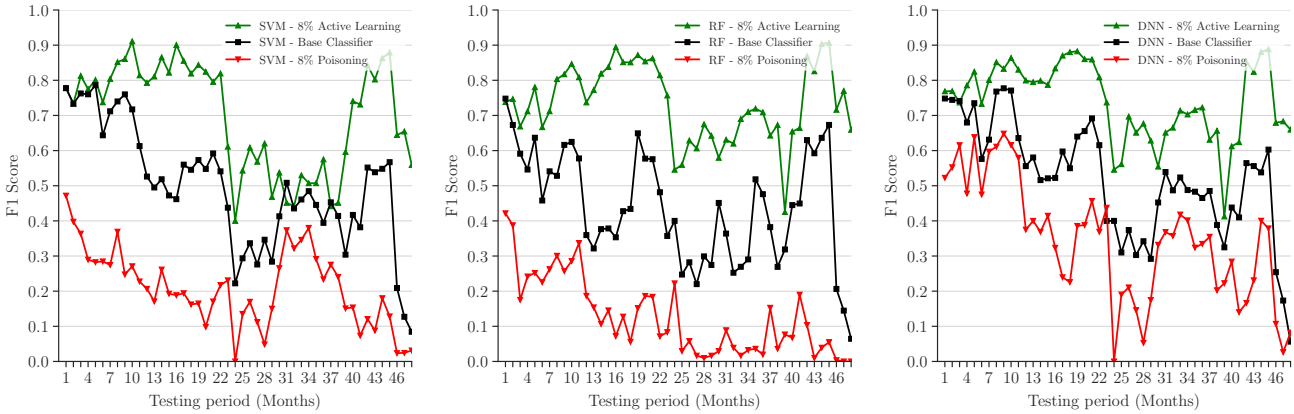
**Time-aware Evaluations.** TESSERACT is a framework proposed by [1] which implements a time-aware evaluation while removing spatial and temporal bias. Spatial bias is the result of an unrealistic test-time class distribution and can be prevented in the Android malware domain by using a realistic $10\%$ malware distribution [1]. Temporal bias is the incorporation of future knowledge into the model and is prevented by the use of time-stamped data to ensure that training data precedes testing data. Finally, concept drift is the divergence of testing data from training data as a result of non-stationarity within the domain, which causes the performance of a model to decay over time [1], [18], [2]. Our time-aware evaluation follows both key recommendations from TESSERACT to ensure that the results presented are not spatially or temporally biased and mitigates concept drift through techniques discussed in Subsection 2.3.

TABLE 1: Breakdown of training and testing splits of the dataset used followed by the rounded number of samples included per year for different active learning rates.

| **Breakdown of D1418** | | | | | |
|---|---|---|---|---|---|
| Years | Training | Testing | | | |
| | 2014 | 2015 | 2016 | 2017 | 2018 |
| Goodware | 52,043 | 28,169 | 36,782 | 60,000 | 55,849 |
| Malware | 5,697 | 3,065 | 3,973 | 7,201 | 6,451 |
| Total | 57,740 | 31,234 | 40,755 | 67,201 | 62,300 |
| Sampling | 2% | 625 | 815 | 1,344 | 1,256 |
| | 4% | 1,249 | 1,630 | 2,688 | 2,492 |
| | 8% | 2,499 | 3,260 | 5,376 | 4,984 |
| | 16% | 4,997 | 6,521 | 10,752 | 9,968 |

**Dataset.** We use the dataset from Barbero et al. [2], which was downloaded from AndroZoo [15] and also used in Kan et al. [14]. The dataset considers the popular DREBIN [16] feature representation. We use a feature selection of the top $10,000$ features as performed in Kan et al. [14]. In the rest of this paper, we refer to this dataset as D1418. The dataset consists of $259,230$ applications covering the five-year period of 2014 to 2018. As in related works [1], [14], we use data from 2014 to train the classifiers. The remaining four years of data are used for our test period to evaluate the long-term stability of passive recovery. Although the samples from the dataset are several years old, this is not an issue for our evaluation for two main reasons. Firstly, the trends being investigated are a result of the non-stationarity at the core of the domain. Therefore, the trends found on this data should continue to hold on newer data. Secondly, labels take at least a year to stabilize [5]; therefore, the use of older samples with new labels enables higher quality ground-truth, which is crucial for our evaluation. The first section of Table 1 reports the details of the train-test splits in the D1418 dataset followed by the number of additional samples included in retraining per year for different active learning rates.

**Classifiers.** The three classifiers used in our evaluation are Linear Support Vector Machines, Deep Neural Networks, and Random Forests. Linear Support Vector Machines, referred to as SVM, was chosen as it was the classifier initially used in DREBIN [16]. Deep Neural Networks, referred to as DNN, are another common approach for malware classification, with the specific implementation of DNN used being based on [19]. Finally, Random Forests, referred to as RF, are an alternative approach used by [20], [17]. We use the same RF settings of $101$ decision trees and a maximum depth of $64$ as [20]. The hyperparameters used for each classifier can be found in Table 6. To show the impact of concept drift on performance and underpin the need for mitigation strategies (such as active learning), Figure 2 displays the baseline performance (without poisoning or active learning) of the three classifiers on the D1418 dataset.




(a) SVM Classifier (b) RF Classifier (c) DNN Classifier

Figure 2: Baseline performance of the three classifiers under the effects of concept drift (black), active learning (green), and poisoning (red).

## 2.3. Passive Recovery Mechanism

A passive recovery mechanism is a method that allows the model to be adapted or modified over time and typically comes with its own operational costs. When deciding on a candidate passive recovery mechanism, it is important to consider a realistic labeling budget, with the cost of labeling being the number of testing objects that must be labeled periodically.

**Active Learning.** One of the most common forms of concept drift mitigation, active learning is a field of machine learning literature that studies the continuous improvement of models through the selection of new samples to be manually labeled and included in retraining. The candidate passive recovery mechanism in this paper is uncertainty sampling, a type of active learning, which selects samples where the classifier is less certain, as these samples are likely the most informative to adjust the decision boundary [21], [22], [23]. To measure uncertainty, we use two different methods based on the classifier. Firstly, for RF and DNN we use probabilistic uncertainty which selects the samples with the lowest probability for its predicted label [21]. Secondly, since SVM does not have probabilities, we use the absolute distance to the hyperplane to measure the uncertainty and select the samples that are closest (most uncertain) [22], [23]. Once samples are selected in a test period, they are then manually labeled and added to the rest of the training set for model retraining. The impact of uncertainty sampling on mitigating concept drift can be seen in the performance improvement over the base classifier in Figure 2.

**Operational Cost.** For the passive recovery evaluation, the uncertainty sampling of 2%, 4%, 8%, and 16% of the monthly samples was used. The granularity of the sampling settings was chosen because it has a higher concentration in lower, more realistic percentages, while maintaining sizable changes to allow variance in the scenarios evaluated. The consequence of using higher percentages is increased manual labeling requirements that may exceed a realistic labeling budget. Miller et al. [5] suggested two possible labeling budgets of 10 and 80 samples per day; assuming an approximate 250 working days per year, the yearly manual labeling budgets would be $2,500$ and $20,000$, respectively. The second section of Table 1 displays the number of new samples included in retraining that must be labeled for a given active learning rate and year of testing. Given the aforementioned yearly labeling budgets and the sampling requirements for our approach, barring one year, 4% active learning fits into the 10 sample a day budget, and all fall well within the 80 sample a day budget.

**Tolerance Margin.** The parameter $\delta$ for the experiments in this research was set to 0.05, 0.01 and 0.00, to show a varying level of strictness—including 0.00 tolerance for absolute recovery.

## 2.4. Threat Model

Adversarial machine learning investigates the possible routes of attack in a machine learning system to discover novel threats and propose ways to mitigate them. The threat in our research is data poisoning, which is the most common adversarial machine learning training-time attack and aims to affect a classifier by adding specially designed data to the training set [24]. Our threat model was defined taking into account existing threat models [25], [24] as well as the guidelines of [26] to avoid inappropriate threat models for the given scenario.

**Availability Data Poisoning.** Specifically, data poisoning attacks can be used to impact the availability of a model through deliberate performance degradation or the integrity of a model via the creation of backdoors. Availability data poisoning further exacerbates concept drift as it causes the divergence of training data from accurate data acting as a pre-test-time drift. Our research therefore focuses on the

TABLE 2: Presents the number of samples, overall features changed as well as the average per-sample features changed needed to perform the clean-label poisoning mimicry of the dirty-label attack.

| Features Flipped by Clean-Label Poisoning | | | |
|---|---|---|---|
| Poisoning Rate | Samples | Overall | Per-Sample |
| 2% | 1,156 | 54,658 | 47 |
| 4% | 2,312 | 106,802 | 46 |
| 8% | 4,626 | 214,254 | 46 |
| 16% | 9,252 | 421,054 | 46 |

threat of availability data poisoning attacks as a result of its close connection with concept drift.

**Attacker's Objective & Capabilities.** There can be two main attacker goals for availability data poisoning: having the classifier missing threats (False Negatives); having the classifier generating false alarms (False Positives). The attacker would design their attack strategy to better suit their primary target. Without loss of generality, we examine both options by evaluating an attack which compromises the general availability of the model. For the capability, we assume that the attacker only has access to a portion of the training data, reflected in the poisoning rate, and no additional information on the classification system utilized. Therefore, there are two potential capabilities for the attacker: either (i) they have influence over the training data labeling process, or (ii) they have the ability to modify the prepared training data (feature vectors). In addition, the poisoning attack strengths of 2%, 4%, 8%, and 16% of the training samples were used in our evaluation. These levels of attack were chosen because they fall within the strength range of similar attacks (such as [27]) and compare fairly with the active learning scenario.

**Poisoning Strategy.** When choosing a specific poisoning strategy for evaluation, it is important that it fits the chosen threat model and classification system. Additionally, the adversarial manipulations performed should be realistic given the proposed attacker's capabilities.

**Dirty-Label Poisoning.** This attack is used for the case in which the attacker has influence over the labeling process as we consider a label-flip poisoning attack [24] for our dirty-label poisoning approach. Label-flip poisoning is chosen as it is model-agnostic, therefore requires no knowledge about the model and does not rely on assumptions or knowledge about the classifier like gradient-based attacks. We implement the attack to flip the labels of training data in an equal ratio of goodware to malware so as to maintain the original class distribution to avoid detection.

**Clean-Label Poisoning.** This attack is used for the case in which the attacker can only modify the prepared training data and not the labels; therefore, we use a feature-flip attack as our clean-label poisoning approach [24]. We designed our clean-label approach to mimic our dirty-label approach to maintain the same model-agnostic property and training data access requirements. Therefore, our clean-label poisoning attack changes the feature values of samples to recreate samples of the opposite class, thereby enabling the creation of the same poisoning set as the dirty-label attack without label modification. We minimize the number of features required to be changed by finding goodware/malware pairs with the least number of different benign and malicious features. The number of features that were changed for each poisoning rate can be found in Table 2.

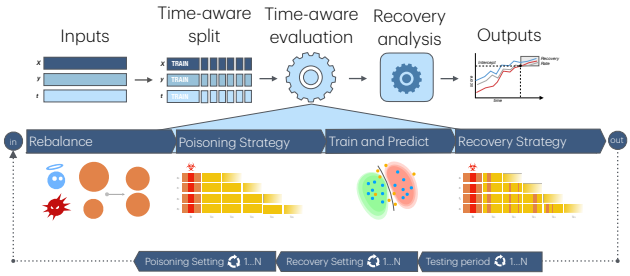



Figure 3: The evaluation of a candidate passive recovery mechanism.

**Poisoning Evaluation.** Since our dirty-label and clean-label poisoning attacks are two different designs for the same attack based on different attacker capabilities, we present only one set of attack results to cover both cases. The impact of data availability poisoning over time on the base classifiers without active learning can be found in Figure 2.

## 2.5. Evaluation Pipeline

The pipeline of a passive recovery evaluation is presented in Figure 3. The initial input is a time-stamped dataset consisting of a feature matrix $X$, labels $y$, and time-stamps $t$. The dataset is temporally split into a training set and testing months using the TESSERACT framework [1]. Then, the time-aware evaluation cycle begins. First, the training dataset is rebalanced to a realistic class distribution (e.g., approximately 10% in the case of Android malware [1]). The dataset is then poisoned based on the current iteration's poisoning rate (starting from 0%, for the hypothetical model baseline) and is then used to train the model. The model predicts on the current test month data. After prediction, the passive recovery mechanism (e.g. active learning [21]) selects new samples to be labeled, and then the model is re-trained. The train, predict, sample, and retrain process is then repeated for all test months. After all test months have been evaluated, the process is repeated for all poisoning and recovery rates in the evaluation settings. Once all setting evaluations have been completed, the intercept and recovery rates are extracted from the results.

# 3. Evaluation

In this section, we conduct experiments following the settings discussed and motivated in Section 2, and aim to answer the following research questions:

TABLE 3: Recovery results table for the different tolerance margins (0.05, 0.01, and 0) and classifiers (SVM, DNN, RF). We report *intercept* (lower is better) and *recover rate* (higher is better) for each scenario, and use background gradients to provide a visual cue. The letter "X" is used if the intercept is never reached within the specified tolerance margin.

| **Recovery Results Table** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Classifiers | | | SVM | | | | DNN | | | | RF | | | | |
| Tolerance Margin | Active Learning Rate | | Poisoning Rate | | | | Poisoning Rate | | | | Poisoning Rate | | | | |
| | | | 2% | 4% | 8% | 16% | 2% | 4% | 8% | 16% | 2% | 4% | 8% | 16% | |
| 0.05 | 0% | Intercept (Month) | 1 | 24 | X | X | 1 | 1 | 7 | X | 2 | 24 | X | X | |
| | | Recovery Rate (%) | 10% | 12% | 0% | 0% | 67% | 56% | 7% | 0% | 62% | 20% | 0% | 0% | |
| | 2% | Intercept (Month) | 1 | 16 | 22 | 22 | 1 | 1 | 3 | 22 | 1 | 5 | 16 | 21 | |
| | | Recovery Rate (%) | 75% | 67% | 67% | 52% | 90% | 85% | 72% | 89% | 94% | 73% | 70% | 56% | |
| | 4% | Intercept (Month) | 1 | 9 | 15 | 22 | 1 | 1 | 5 | 23 | 2 | 9 | 13 | 21 | |
| | | Recovery Rate (%) | 90% | 88% | 79% | 67% | 77% | 75% | 61% | 81% | 87% | 90% | 89% | 75% | |
| | 8% | Intercept (Month) | 1 | 2 | 19 | 21 | 1 | 2 | 7 | 16 | 4 | 4 | 11 | 16 | |
| | | Recovery Rate (%) | 83% | 74% | 83% | 82% | 94% | 87% | 83% | 73% | 100% | 91% | 95% | 91% | |
| | 16% | Intercept (Month) | 1 | 2 | 9 | 21 | 2 | 1 | 4 | 15 | 1 | 4 | 10 | 14 | |
| | | Recovery Rate (%) | 98% | 87% | 82% | 96% | 98% | 90% | 87% | 74% | 90% | 93% | 95% | 86% | |
| 0.01 | 0% | Intercept (Month) | X | X | X | X | 5 | 5 | 23 | X | 8 | 24 | X | X | |
| | | Recovery Rate (%) | 0% | 0% | 0% | 0% | 20% | 23% | 8% | 0% | 27% | 4% | 0% | 0% | |
| | 2% | Intercept (Month) | 9 | 22 | 23 | 22 | 1 | 4 | 3 | 22 | 5 | 8 | 23 | 34 | |
| | | Recovery Rate (%) | 33% | 59% | 46% | 30% | 60% | 57% | 52% | 41% | 70% | 44% | 69% | 53% | |
| | 4% | Intercept (Month) | 2 | 22 | 22 | 22 | 3 | 6 | 6 | 31 | 9 | 10 | 17 | 33 | |
| | | Recovery Rate (%) | 64% | 74% | 70% | 59% | 56% | 37% | 33% | 100% | 62% | 64% | 62% | 88% | |
| | 8% | Intercept (Month) | 2 | 23 | 23 | 23 | 9 | 3 | 15 | 28 | 9 | 12 | 12 | 20 | |
| | | Recovery Rate (%) | 45% | 77% | 65% | 65% | 65% | 52% | 53% | 95% | 70% | 76% | 59% | 52% | |
| | 16% | Intercept (Month) | 8 | 21 | 21 | 23 | 2 | 4 | 14 | 15 | 4 | 10 | 14 | 27 | |
| | | Recovery Rate (%) | 76% | 86% | 93% | 96% | 72% | 67% | 80% | 65% | 78% | 74% | 83% | 95% | |
| 0 | 0% | Intercept (Month) | X | X | X | X | 5 | 5 | 23 | X | 23 | 24 | X | X | |
| | | Recovery Rate (%) | 0% | 0% | 0% | 0% | 16% | 18% | 8% | 0% | 31% | 4% | 0% | 0% | |
| | 2% | Intercept (Month) | 9 | 22 | 23 | 22 | 10 | 15 | 15 | 22 | 5 | 8 | 23 | 34 | |
| | | Recovery Rate (%) | 20% | 48% | 27% | 22% | 51% | 56% | 41% | 30% | 48% | 28% | 54% | 33% | |
| | 4% | Intercept (Month) | 15 | 22 | 22 | 22 | 3 | 6 | 6 | 31 | 9 | 10 | 23 | 33 | |
| | | Recovery Rate (%) | 62% | 63% | 67% | 52% | 46% | 21% | 16% | 89% | 40% | 38% | 42% | 62% | |
| | 8% | Intercept (Month) | 2 | 23 | 23 | 23 | 9 | 9 | 15 | 28 | 9 | 12 | 12 | 23 | |
| | | Recovery Rate (%) | 38% | 57% | 50% | 54% | 56% | 50% | 44% | 95% | 40% | 46% | 32% | 19% | |
| | 16% | Intercept (Month) | 8 | 23 | 23 | 23 | 2 | 4 | 15 | 23 | 8 | 10 | 14 | 30 | |
| | | Recovery Rate (%) | 71% | 88% | 88% | 92% | 66% | 57% | 74% | 81% | 41% | 33% | 40% | 84% | |

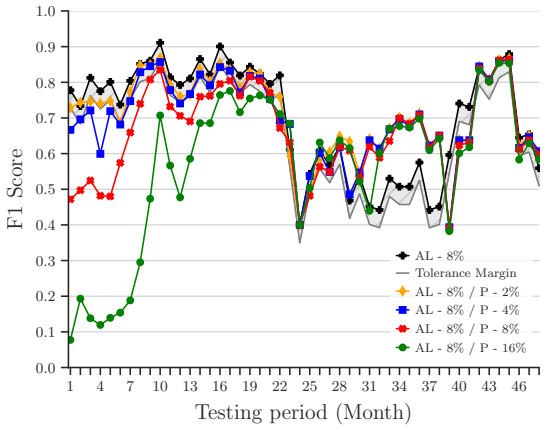
- RQ-INTERCEPT: How do active learning and poisoning rates affect the speed of passive recovery?
- RQ-RECOVERY-RATE: How stable is the passive recovery over time across different settings?
- RQ-CLASSIFIER: Does the type of classifier used affect the passive recovery of the poisoned model?

Figure 4 consists of two sets of experiment results for the performance of SVM, DNN and RF under different settings. Figures 4a, 4c, and 4e show the time-aware evaluation considering a fixed active learning rate of 8%, for varying poisoning rates. The fixed active learning rate plots show how different rates of poisoning impact the passive recovery of the system, with the shaded area representing the tolerance margin. Figures 4b, 4d, and 4f represent time-aware evaluations from a defender's perspective, and assume different active learning rates for a fixed 8% poisoning rate. These plots demonstrate how different active learning rates compare against a constant attack strength. 8% is chosen for both fixed active learning and poisoning rates in the figures, as it examines a moderate strength that lies at the mean of the rates evaluated. In each plot, the dashed black line is a baseline with the variable setting (either active learning or poisoning) set to 0%, the $X$ axis is the test months, the $Y$ axis is the $F_1$-Score, and the remaining colored lines show the variable setting value's performance.

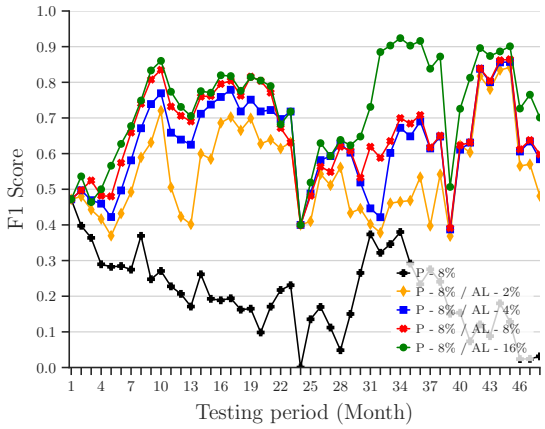
For a general overview on more combinations, Table 3 reports tables with passive recovery results of the complete combination of active learning and poisoning rates for tolerance margins: 0.05 (5%), 0.01 (1%), 0.00 (0%). Each table reports the results of each setting via its intercept and recovery rate. The intercept is as defined in Definition 2.2 and is presented as the test month in which the poisoned model performs within the tolerance margin of the hypothetical model. The recovery rate is defined in Definition 2.3 and is presented as the percentage of months after the intercept within the tolerance margin. The tables provide insight into the numerical results of the settings displayed in the plots along with others not included, allowing for ease of comparison across different settings. The only difference between the three sets of tolerance margin tables is the margin used in the evaluation, allowing comparison between different tolerance margins.

## 3.1. Speed of Passive Recovery

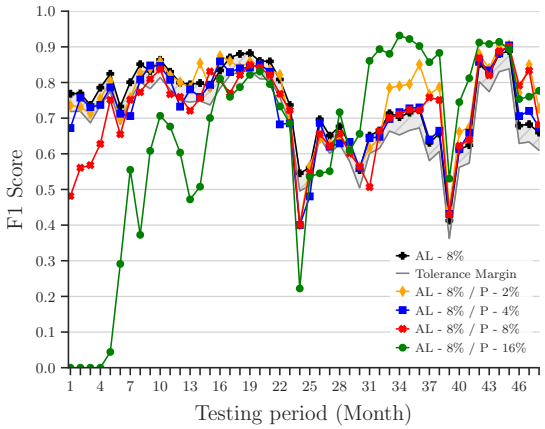
Through comparisons between the tables contained within Table 3 we can evaluate the speed with which the models recover thereby addressing the research question RQ-INTERCEPT. When comparing the intercepts of experiments with the same active learning and poisoning rates in Table 3, we can see that the intercept increases consistently as the rates increase. This shows that an increase in the poisoning rate has a larger impact than the active learning rate on the intercept. Poisoning could force substantially



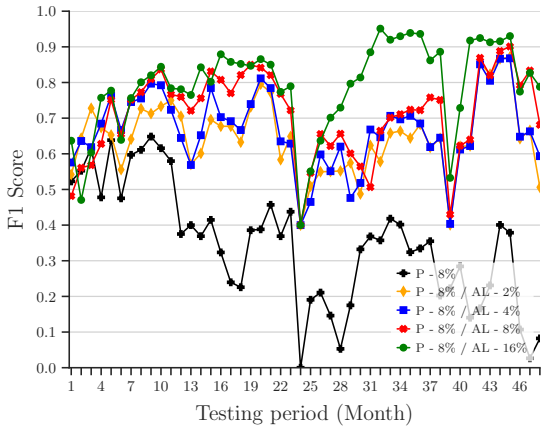

(a) **SVM**: Fixed Active Learning Rate



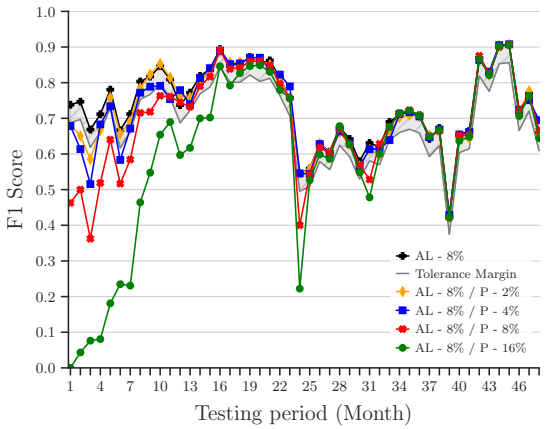

(b) **SVM**: Fixed Poisoning Rate



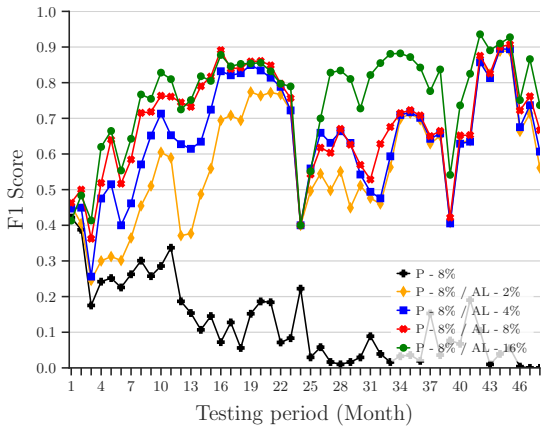

(c) **DNN**: Fixed Active Learning Rate




(d) **DNN**: Fixed Poisoning Rate




(e) **RF**: Fixed Active Learning Rate




(f) **RF**: Fixed Poisoning Rate

Figure 4: Recovery plots showing the impact of varying either poisoning or active learning rate for D1418. The fixed active learning rate plots vary the strength of poisoning, and show the trend of convergence on hypothetical performance despite the increasing impact of poisoning, as can be seen in the starting performance of the different settings. The fixed poisoning rate plots vary the strength of active learning, and show the diminishing returns of performance gains from active learning. The complete results plots can be found in Figure 5

.

## 4. Discussion

This section elaborates on the relevant findings of our evaluation and acknowledges the limitations of our research.

**Relevant Findings.** Our evaluation shows that passive recovery using active learning is capable of achieving an average intercept and recovery rate, across all settings and classifiers, of 9 months and 70%, respectively, for a tolerance margin of 5% given the evaluated threat model. We examined the three key factors of RQ-INTERCEPT, RQ-RECOVERY-RATE, and RQ-CLASSIFIER for passive recovery. In RQ-INTERCEPT, we observe that poisoning does not suffer from the same diminishing returns as active learning, implying that passive recovery is more suitable for cases in which lower attack strengths are used or in conjunction with active recovery approaches to aid in overall recovery. In RQ-RECOVERY-RATE, we evaluated the stability of passive recovery across the different rates of poisoning and active learning. We observe a comparable average recovery rate across the different classifiers with the same tolerance margin. This indicates that the choice of classifier does not have a significant impact on the average recovery rate for a tolerance margin, therefore, all classifiers evaluated are capable of achieving passive recovery. In RQ-CLASSIFIER, we determine the two-fold impact that the choice of classifier has on passive recovery. Firstly, the utilized classifier does impact the overall passive recovery, although the average recovery rate is not largely impacted by the classifier, the intercept denoting the start of passive recovery is impacted. Secondly, the performance ($F_1$ score) is also affected by the choice of classifier, especially under the effects of poisoning and active learning. As a result of the two-fold impact of the choice of classifier, we observe that DNN is the best recovering, when taking into account both intercept and recovery rate, as well as the best performing classifier in our evaluation, followed by RF then SVM. These findings motivate the importance of taking passive recovery into consideration when designing machine learning based malware detection systems, as passive recovery is not only feasible but could be sufficient for long-term stability of classifier performance or a supplement to active recovery approaches depending on the constraints.

**Limitations.** We consider a very specific domain and setting: passive recovery from dirty-label (label-flip) or clean-label (feature-flip) poisoning with uncertainty-sampling active learning, in the context of Android malware detection. Despite the very focused scenario, we were able to derive interesting conclusions and propose a generalizable evaluation that can open up future investigations on more poisoning and passive recovery strategies.

## 5. Open Research Directions

In this section, we cover potential research directions that expand on the concepts explored in this research.

**Problem-space Attacks.** In adversarial machine learning for malware classification, the problem of poisoning is not straightforward as a result of the challenge to make changes to the application while maintaining functionality [28]. For simplicity, we consider only feature-space attacks, by assuming that the attacker is able to poison a dataset by flipping the labels or feature values of existing apps. Future research to explore the impact of problem-space data poisoning attacks on passive recovery would improve the transferability of these findings to practitioners.

**Poisoning-aware Passive Recovery Strategies.** In our recovery strategy, we use the common active learning approach of uncertainty sampling to evaluate passive recovery; however, concept drift mitigations have not been considered in the context of a poisoned dataset. The development of poisoning-aware concept drift mitigation strategies will not only improve the robustness of the system as a whole to an attack, but also improve the passive recovery capability of that system.

**Time-aware Poisoning.** In this paper, we assume the attacker can poison the model only once at training time (cf. Figure 3). However, it would be illuminating for future research to explore whether there are feasible scenarios in which the attacker may want to poison a model in a "low and slow" fashion (similarly to advanced persistent threats), and how that can impact the overall model performance over time.

**Relationship with Poisoning Mitigations.** Existing papers have either proposed defenses against poisoning attacks (e.g. [10]), or investigated with forensics whether a model was indeed poisoned [11]. Here we take an orthogonal direction and investigate whether retraining over time can help in forgetting poisoning, and provide an evaluation framework and metrics for evaluating its impact. A complete classification system should contain passive and active approaches to dealing with poisoning. Therefore, development of passive and active approaches together would enable the most efficient and stable defenses over time.

## 6. Related Work

To the best of our knowledge, this paper proposes the first exploration of passive recovery for availability data poisoning in the context of malware detection. In the remainder of this section, we comment on existing research and provide a link for other future work in terms of feature abstractions (representations), recovery, and poisoning strategies.

**Feature Representations.** This research evaluated the feature representation DREBIN [16] as it is one of the most common feature spaces released so far in the Android malware domain. However, there are other feature extractions yet to be explored in future work. StormDroid is a feature extraction framework from [29] that combined commonly used permissions and sensitive API calls features with two novel features of sequence and dynamic behavior. The sequence features are the number of sensitive API calls requested by malware and goodware, respectively. The dynamic behavior features are obtained by running

the application's APK in DroidBox and then performing static analysis on the log files generated by the run. DroidCat is a feature extraction framework from [30] that uses only dynamic analysis and handles both binary classification and multiclass classification [30]. The DroidCat feature space consists of features for method calls and inter-component communication intents instead of permissions, app resources, or system calls like other dynamic analysis frameworks. DroidCat outperformed both DroidSieve [31] and Afonso [32] with and without obfuscation. Both feature extractions could be candidates for future work to explore the generalization of our results.

**Recovery Strategies.** Active learning with uncertainty sampling was used as a recovery strategy in this research; however, there are other potential approaches to passive recovery. DroidEvolver++ developed by [6] is a malware detection framework which uses pseudo labels to retrain and update the model to prevent concept drift that impacts performance over time. The DroidEvolver++ framework is an extension of the DroidEvolver framework [33]. Despite the improvements, labelless retraining has the limitation that poor predictions can cause the model to self-poison, thereby making it potentially unreliable. CADE [4] is a drift detection and explanation framework that relies on a contrastive autoencoder to map to a lower dimensional feature space which is then used to learn a distance function to measure the similarity of samples in a class and generate a centroid for each class. A sample is deemed to drift if the sample is outside the distribution for all classes. Transcendent is a framework built from Transcend designed to rectify some of the limitations of the original framework [18], [2]. Transcendent uses the same nonconformity measures and p-values as Transcend, but introduces two new conformal evaluators for concept drift detection: an inductive conformal evaluator and a cross-conformal evaluator [2]. CADE or Transcendent could be used as a selection strategy for active learning instead of the uncertainty sampling used in this research.

**Poisoning Strategies.** In this paper, we use dirty-label poisoning (label-flip) and clean-label poisoning (feature-flip) as a first exploratory study, presenting implications and findings on specific classifiers over a long time frame. There are many other perturbation-based poisoning strategies. The paper by [34] investigates the use of three different poisoning attacker models on a plethora of Android malware detection frameworks. The attacked classification systems all use support vector machines, with the evaluated feature spaces being DREBIN [16], DroidAPIMiner [35], MAMADROID [20], and StormDroid [29] with misclassification rates of $80.05\%$, $75.20\%$, $68.95\%$ and $65.35\%$, respectively [34]. The paper by [7] uses explanation techniques to poison the model, creating backdoors in the classifier in a model-agnostic way [7]. A backdoor in malware classification represents a set of features that, if present, will result in a malicious application being labeled goodware. Recovery from backdoor poisoning is an orthogonal variant of this research, as we have focused on recovery from general performance degradation poisoning.

# 7. Conclusion

We demonstrated that active learning, a concept drift mitigation strategy, can indeed facilitate passive recovery from availability data poisoning, and achieve convergence with its hypothetical non-poisoned model's performance. The speed of passive recovery, as well as the model performance, depend on the chosen classifier and passive recovery parameters. Furthermore, the stability of passive recovery is consistent between the classifiers evaluated, demonstrating that they are capable of passive recovery.

The purpose of this research was to investigate the feasibility of recovery from availability data poisoning through the use of standard ML practices, to pave the way for future research on this topic. We hope that the findings in this paper will encourage the development of poisoning-aware concept drift mitigation strategies, as well as broaden the perspective of adversarial machine learning to the non-stationary nature of malware classification.

# Availability

We have implemented our passive recovery evaluation as a Python library. We are releasing the framework to promote future research on the evaluation of passive recovery of other systems. This can also be used to replicate all the experiments in the current paper. The repository can be found at: https://github.com/s2labres/RPAL.

# Acknowledgment

Research partially funded by: the Defence Science and Technology Laboratory (DSTL), an executive agency of the UK Ministry of Defence, supporting the Autonomous Resilient Cyber Defence (ARCD) project within the DSTL Cyber Defence Enhancement programme; Google ASPIRE Awards; UK EPSRC Grant no. EP/X015971/1.

# References

[1] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "TESSERACT: Eliminating Experimental Bias in Malware Classification across Space and Time," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 729–746.

[2] F. Barbero, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Transcending Transcend: Revisiting Malware Classification in the Presence of Concept Drift," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 805–823.

[3] J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodríguez, N. V. Chawla, and F. Herrera, "A Unifying View on Dataset Shift in Classification," *Pattern recognition*, vol. 45, no. 1, pp. 521–530, 2012.

[4] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, "CADE: Detecting and Explaining Concept Drift Samples for Security Applications," in *USENIX security symposium*, 2021, pp. 2327–2344.

[5] B. Miller, A. Kantchelian, M. C. Tschantz, S. Afroz, R. Bachwani, R. Faizullabhoy, L. Huang, V. Shankar, T. Wu, G. Yiu *et al.*, "Reviewer Integration and Performance Measurement for Malware Detection," in *Detection of Intrusions and Malware, and Vulnerability Assessment: 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings 13*. Springer, 2016, pp. 122–141.

[6] Z. Kan, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Investigating Labelless Drift Adaptation for Malware Detection," in *ACM AISec Workshop*, 2021.

[7] G. Severi, J. Meyer, S. E. Coull, and A. Oprea, "Explanation-Guided Backdoor Poisoning Attacks Against Malware Classifiers," in *USENIX Security Symposium*, 2021, pp. 1487–1504.

[8] L. Yang, Z. Chen, J. Cortellazzi, F. Pendlebury, K. Tu, F. Pierazzi, L. Cavallaro, and G. Wang, "Jigsaw Puzzle: Selective Backdoor Attack to Subvert Malware Classifiers," *arXiv preprint arXiv:2202.05470*, 2022.

[9] X. Xu, Q. Wang, H. Li, N. Borisov, C. A. Gunter, and B. Li, "Detecting AI Trojans using Meta Neural Analysis," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 103–120.

[10] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 707–723.

[11] S. Shan, A. N. Bhagoji, H. Zheng, and B. Y. Zhao, "Poison Forensics: Traceback of Data Poisoning Attacks in Neural Networks," 2022.

[12] T. Chow, Z. Kan, L. Linhardt, L. Cavallaro, D. Arp, and F. Pierazzi, "Drift forensics of malware classifiers," in *Proc. of the ACM Workshop on Artificial Intelligence and Security (AISec)*, 2023.

[13] S. McFadden, Z. Kan, L. Cavallaro, and F. Pierazzi, "Poster: Rpal-recovering malware classifiers from data poisoning using active learning," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 3561–3563.

[14] Z. Kan, S. McFadden, D. Arp, F. Pendlebury, R. Jordaney, J. Kinder, F. Pierazzi, and L. Cavallaro, "Tesseract: Eliminating experimental bias in malware classification across space and time (extended version)," *arXiv preprint arXiv:2402.01359*, 2024.

[15] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: ACM, 2016, pp. 468–471. [Online]. Available: http://doi.acm.org/10.1145/2901739.2903508

[16] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and Explainable Detection of Android Malware in your Pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.

[17] X. Zhang, Y. Zhang, M. Zhong, D. Ding, Y. Cao, Y. Zhang, M. Zhang, and M. Yang, "Enhancing state-of-the-art classifiers with api semantics to detect evolved android malware," in *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, 2020, pp. 757–770.

[18] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting Concept Drift in Malware Classification Models," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 625–642.

[19] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial examples for malware detection," in *Computer Security–ESORICS 2017: 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II 22*. Springer, 2017, pp. 62–79.

[20] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro, G. Ross, and G. Stringhini, "MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models (Extended Version)," *ACM Transactions on Privacy and Security (TOPS)*, vol. 22, no. 2, pp. 1–34, 2019.

[21] B. Settles, "Active learning literature survey," 2009.

[22] S. Tong and D. Koller, "Support vector machine active learning with applications to text classification," *Journal of machine learning research*, vol. 2, no. Nov, pp. 45–66, 2001.

[23] J. Kremer, K. Steenstrup Pedersen, and C. Igel, "Active learning with support vector machines," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 4, no. 4, pp. 313–326, 2014.

[24] A. E. Cinà, K. Grosse, A. Demontis, S. Vascon, W. Zellinger, B. A. Moser, A. Oprea, B. Biggio, M. Pelillo, and F. Roli, "Wild Patterns Reloaded: A Survey of Machine Learning Security against Training Data Poisoning," *ACM Computing Surveys*, 2022.

[25] B. Biggio and F. Roli, "Wild Patterns: Ten years After the Rise of Adversarial Machine Learning," 2018.

[26] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and Don'ts of Machine Learning in Computer Security," in *Proc. of the USENIX Security Symposium*, 2022.

[27] A. E. Cinà, S. Vascon, A. Demontis, B. Biggio, F. Roli, and M. Pelillo, "The hammer and the nut: Is bilevel optimization really needed to poison linear classifiers?" in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[28] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," in *2020 IEEE symposium on security and privacy (SP)*. IEEE, 2020, pp. 1332–1349.

[29] S. Chen, M. Xue, Z. Tang, L. Xu, and H. Zhu, "Stormdroid: A Streaminglized Machine Learning-Based System for Detecting Android Malware," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, 2016, pp. 377–388.

[30] H. Cai, N. Meng, B. Ryder, and D. Yao, "Droidcat: Effective Android Malware Detection and Categorization via App-Level Profiling," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 6, pp. 1455–1470, 2018.

[31] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Cavallaro, "Droidsieve: Fast and Accurate Classification of Obfuscated Android Malware," in *Proceedings of the seventh ACM on conference on data and application security and privacy*, 2017, pp. 309–320.

[32] V. M. Afonso, M. F. de Amorim, A. R. A. Grégio, G. B. Junquera, and P. L. de Geus, "Identifying Android Malware using Dynamically Obtained Features," *Journal of Computer Virology and Hacking Techniques*, vol. 11, pp. 9–17, 2015.

[33] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "DroidEvolver: Self-Evolving Android Malware Detection System," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 47–62.

[34] S. Chen, M. Xue, L. Fan, L. Ma, Y. Liu, and L. Xu, "How Can We Craft Large-Scale Android Malware? An Automated Poisoning Attack," in *2019 IEEE 1st international workshop on artificial intelligence for mobile (AI4Mobile)*. IEEE, 2019, pp. 21–24.

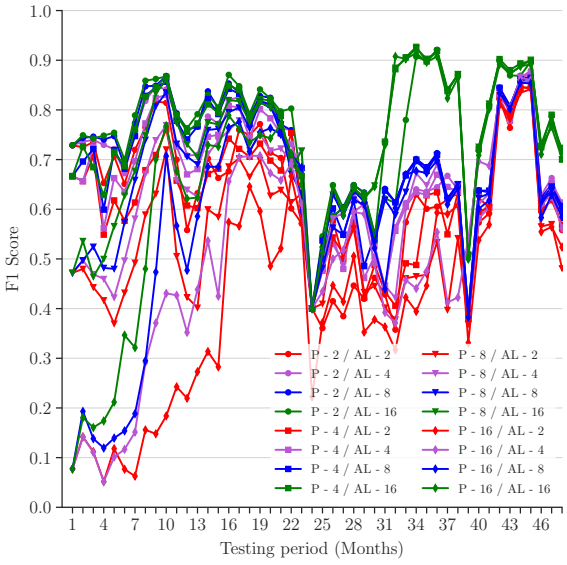
[35] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android," in *Security and Privacy in Communication Networks: 9th International ICST Conference, SecureComm 2013, Sydney, NSW, Australia, September 25-28, 2013, Revised Selected Papers 9*. Springer, 2013, pp. 86–103.

## Appendix

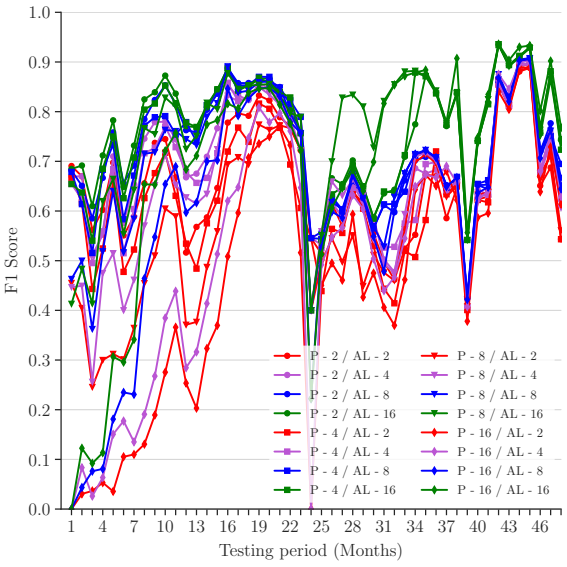
This section presents the table that contains the hyperparameters of the classifiers, as well as the plots that contain the results for every setting given a specific classifier. The plots show the generalization of the trends discussed in Section 3 with respect to the plots in Figure 4.

TABLE 6: Hyperparameters for Different Classifiers

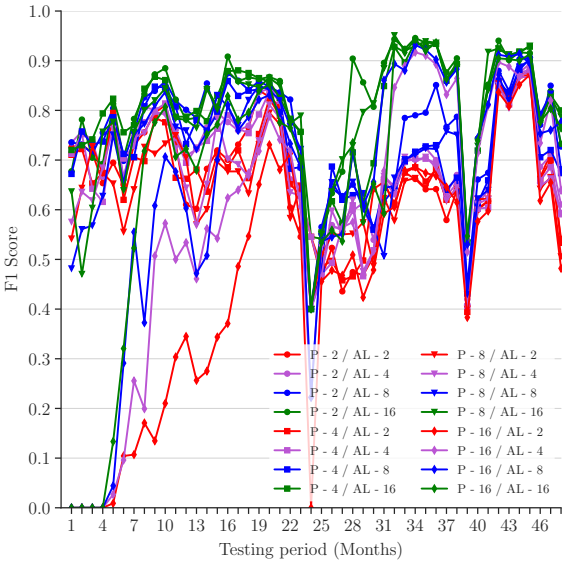
| Classifiers | Hyper Parameters | |
|---|---|---|
| **SVM** | *Max Iterations* | 50000 |
| | *Approach* | LinearSVC (sklearn.svm) |
| **RF** | *Decision Trees* | 101 |
| | *Max Depth* | 64 |
| | *Approach* | RandomForestClassifier (sklearn.ensemble) |
| **DNN** | *Epochs* | 10 |
| | *Batch Size* | 64 |
| | *Learning Rate* | 0.05 |
| | *Training/Validation* | 0.66/0.34 |
| | *Approach* | Input: 10k, ReLU, 0.5 Dropout<br>Hidden: 200, ReLU, 0.5 Dropout<br>Output: 2 |




(a) SVM Results.




(b) RF Results.




(c) DNN Results.

Figure 5: Results plots for all settings given a classifier. The color denotes the active learning rate and the marker denotes the poisoning rate.