Towards more realistic evaluations: The impact of label delays in malware detection pipelines

Marcus Botacin (1) and Heitor Gomes (2)

(1) Texas A&M University, USA - botacin@tamu.edu (corresponding)
(2) Victoria University at Wellington, NZ - heitor.gomes@vuw.ac.nz

Abstract

Developing and evaluating malware classification pipelines to reflect real-world needs is as vital to protect users as it is hard to achieve. In many cases, the experimental conditions when the approach was developed and the deployment settings mismatch, which causes the solutions not to achieve the desired results. In this work, we explore how unrealistic project and evaluation decisions in the literature are. In particular, we shed light on the problem of label delays, i.e., the assumption that ground-truth labels for classifier retraining are always available when in the real world they take significant time to be produced, which also causes a significant attack opportunity window. In our analyses, among diverse aspects, we address: (1) The use of metrics that do not account for the effect of time; (2) The occurrence of concept drift and ideal assumptions about the amount of drift data a system can handle; and (3) Ideal assumptions about the availability of oracle data for drift detection and the need for relying on pseudo-labels for mitigating drift-related delays. We present experiments based on a newly proposed exposure metric to show that delayed labels due to limited analysis queue sizes impose a significant challenge for detection (e.g., up to a 75% greater attack opportunity in the real world than in the experimental setting) and that pseudo-labels are useful in mitigating the delays (reducing the detection loss to only 30% of the original value).

1 Introduction

Malware infections are major threats to modern computer systems and new strategies to fight them emerge at a fast pace. Machine Learning (ML) models are at the core of many state-of-the-art proposals for malware detection. Whereas ML models certainly provide invaluable gains for security, their application is not as straightforward as deploying a single ML model. Rather, a pipeline of solutions is required to enable the long-term operation of the ML model at proper detection rates.

Although the academic literature is rich in proposals for ML-based detectors, most of them [**?**] do not present a complete pipeline to assure the proper operation of the ML model. Most solutions do not account for the different effects ML models are subject to, especially those derived from real-world constraints, such as storage and processing limitations. Neglecting these aspects creates a blurry understanding of the security dynamics of real systems, results in papers that overstate their impact [**?**] and, most importantly, in solution deployments that do not meet the expectations and real-world needs [**?**]. In this scenario, it is key to better understand the pipeline dynamics to allow one to better select the ML-based security architecture that best fits the targeted scenario.

A naive strategy to integrate ML in malware detection is to simply pass incoming samples through a ML model. This pipeline is naive because the performance of the ML model degrades over time due to the evolution of the malware samples (a phenomenon

called concept drift [**?**]). A smarter pipeline includes a concept drift detector to retrain the ML model when its performance is degraded. A major drawback of this solution is that it relies on an oracle to provide the correct labels for concept drift detection. True oracles do not exist in reality. Some works suggest relying on active learning [**?**] as the oracle. However, these works assume that the oracle has unlimited resources and immediate labeling capabilities. In practice, sandboxes (and even more human analysts) used by security companies as oracles have limited availability and a large response time, such that samples spend significant time in queues, causing a delay in the delivery of the ground-truth labels.

A delay in the delivery of the ground-truth labels implies that attackers have an attack opportunity window to act while the sample is not correctly detected by the companies. Thus, it is key to mitigate the delayed label problem to enable faster incident response. Unfortunately, most works in the literature do not model analysis queues as part of their detection pipelines. A really smart architecture to address these real-world constraints should mitigate the problem of the delayed labels via pseudo-labels assigned by an intermediate classifier that bridges the temporal gap between the original ML model and the Oracle solution. Once again, pseudo-labels are not the standard consideration for most pipelines in the literature.

To bridge the gap between real-world constraints and literature works, in this paper, we present malware detection experiments for the multiple architectures presented in the literature while (i) considering and (ii) discarding the real-world limitations and comparing their results. We test all the different architectures with the DDM and EDDM drift detectors with a state-of-the-art classifier [**?**] applied to the DREBIN [**?**] dataset. We simulate conditions where the architecture has full access to retraining (past) data and when the architecture only has access to partial data due to constraints in the malware analysis queue. We propose a new metric to evaluate how exposed to threats users are in each scenario.

Our ultimate goal with this work is that the phenomenon we here describe might be considered by researchers and security engineers in future analyses, designs, and deployments of (realistic) ML-based malware detection solutions.

Our findings for each timeframe in the tested dataset are that:

- Concept drift mitigation reduces the user exposure to new threats on average in $\approx 50\%$;
- Delayed labels increase the user exposure on average in $\approx 50\%$;
- Using pseudo-labels reduces the user exposure on average in $\approx 30\%$.

This paper's contributions are as follows:

- We revisit ML-based malware detection pipelines proposed in the literature and critically analyze their limitations to real-world deployments;
- We demonstrate via experiments that effects derived from real-world constraints, such as label delay, cause a significant impact on the pipeline outcome and should not be neglected;
- We introduce a new exposure metric to evaluate pipelines over time while accounting for the effect of the multiple real-world constraints.

This work is organized as follows: In Section 2, we define the research problem; In Section 3, we critically evaluate the ML pipelines presented in the literature; In Section 4, we detail the proposed experimental methodology; In Section 5, we report experiment results on different architectural designs; In Section 6, we report experiments results on real-world constraints; In Section 7, we discuss the experimental findings; In Section 8, we position our contributions among related works; We draw our conclusions in Section 9.

2 Problem definition

In this section, we formalize the problem addressed in this research work and its associated challenges.

The malware detection problem. Security companies receive multiple suspicious files to scan every

day. The companies have to decide if those files are malicious or not. If the files are malicious, the companies must generate the detection rules. To tackle the malware detection problem, security companies (typically Antivirus ones) use distinct strategies at the backend (their servers) and the frontend (endpoints, also known as user machines) [**?**]. We here investigate what happens in the backend of AV companies, and assume that the outcome of the classification process is a detection rule (e.g. YARA [**?**]) that is efficient at the frontend component,

The process of deciding if an artifact is malicious or not aims to accomplish two goals [**?**]: (I) accuracy and (2) fast response. **Accuracy** means that it should detect malicious and only malicious files, with great confidence. There should be no (or minimum) False Positives (FPs) because blocking legitimate software impedes users from using it, and then they are likely more prone to remove the security software and never buy it again. Currently accepted FP levels are in the magnitude of 1% or less [**?**]. **Fast response** means that it should correctly identify malicious files as soon as possible because the longer it takes to decide (and respond), the greater the chance that attackers will be infecting users with them [**?**].

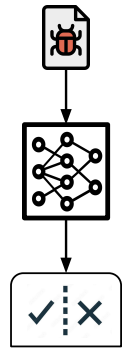
The ML challenge. Combining accurate and fast detection is hard. Companies rely on many techniques for that, including Machine Learning (ML), our focus in this research. To achieve these goals, it is not enough to simply add an ML classification model in production. In fact, it requires a pipeline of ML solutions. This research investigates the best ML pipeline architecture to achieve these two goals. Unlike most previous works, we target a problem setting where the ML model should be capable of evolving to detect new threats.

We model the aforementioned real-world problem as a delayed and partially labeled evolving data stream classification problem [**?**]. The streaming nature of the problem comes from the fact that new malware samples arrive every day, creating an analysis flow. The delayed and partially labeled constraints come from the fact that: (1) new malware might not be immediately correctly labeled, but the wrong labels will be fixed over time, thus the correct labels will be delayed. Also, (2) at any given point in time, only a fraction of the correct labels will be available, as corrections are not immediate, thus the problem presents the partially labeled characteristic. Finally, the evolving aspect comes from the fact that changes are expected as new malware is created to fool the ML pipeline and the ML model is expected to react to it.

3 Pipelines

Modern detection solutions are not composed of a single solution but multiple solutions in an ML pipeline [**?**]. In this section, we revisit detection pipelines proposed in the literature over time and critically analyze their relationship with the real world.

Naive Triage Architecture. The most straightforward detection pipeline uses a single ML classifier for triaging files. Figure 1 shows the execution flow: The unknown suspicious file enters the classifier and it gives its verdict. This classifier should be fast (e.g., static feature extraction) to meet the response time requirement. This pipeline is naive because as the input files change over time, the classifier becomes unable to meet the same accuracy standards as training time [**?**].



1].pdf

Figure 1: **Naive Triage Architecture.** The ML model is never updated and deteriorates over time.

Triage-Update Architecture. A solution for the time-related degeneration is to use adaptive classification models [**?**], updating them either periodically or when concept drift is detected. Thus, in this up-

graded pipeline, the triage classifier is externally updated by an update monitor, such as a drift detector. Figure 2 displays an architecture where the triage classifier is externally updated by an update monitor (e.g., a drift detector).

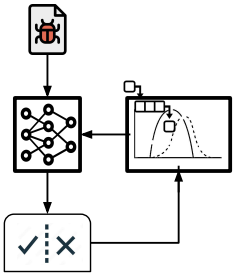

2].pdf

Figure 2: **Triage-Update Architecture.** The triage model is externally updated by a drift detection solution. Most solutions assume that the correct labels are always available for the correction process.

This architecture's drawback is that it requires the correct labels to check if the classification provided by the model matches the real label. Most works in the literature assume that labels are always available by the classification time, which is not realistic [**?**]. In practice, ground-truth labels are often provided by a secondary inspection, such as by human analysts. Analysts take significant time to inspect a sample, thus the ground truth labels often arrive a long time after the samples were first seen by the model. Also, most works assume that correct labels are available in an unconstrained number. Once again, ground-truth labels are usually generated after manual or semi-automated inspection, and human analysts are limited resources, thus the correct labels might spend some time in analysis queues before they are available for the concept drift detector. This dynamic should not be neglected in the statistics otherwise one is evaluating a scenario that does not match reality.

Triage-Oracle-Update Architecture. A more realistic architecture involves not supposing a perfect oracle but adding a second-layer dynamic analysis sandbox to the pipeline, that has a greater discrimination capacity than the static detector and a higher throughput than human analysts. Figure 3 shows an architecture in which the incoming sample is sent both to the static classifier and the dynamic analysis. The rationale is that the static classifier will provide a fast response, while the dynamic analysis solution will provide an accurate response. Therefore, the labels produced by the static classifier and the sandbox are compared by the drift detector to help enhance classification in the future. The hypothesis is that if such predictions diverge significantly, then a concept drift has occurred. This is similar to teacher-student approaches for unsupervised drift detection [**?**] proposed in the stream mining literature.

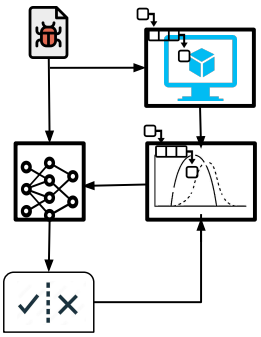
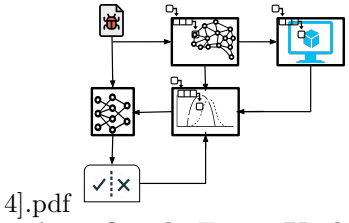

3].pdf

Figure 3: **Triage-Oracle-Update Architecture.** Sandbox execution is treated as an oracle for producing ground truth labels. Dynamic analysis is expensive and it still cannot provide an unlimited number of samples.

Whereas sandboxes are more scalable than human analysts, it is also not an unlimited resource. Tracing samples in sandboxes takes time and although AV companies run large cloud servers, it is not enough to run all the millions of samples that an AV company daily receives. Therefore, some prioritization strategy must be defined: either (1) not all malware samples are sent immediately to the dynamic analysis, but only those whose classifier confidence is low are prioritized, or (2) if a huge number of samples is sent to dynamic analysis, they should wait on a queue. Both strategies once again cause the actual labels to be delayed. This delayed label dynamic cannot be neglected in ML pipeline evaluations, but unfortunately, label delays are hardly ever modeled in the literature papers.

Triage-Oracle-Eager-Update Architecture.

The root cause for the problem of the delayed label is that the Oracle (the sandbox) is much slower in producing labels than the original classifier. This is required because to be more accurate, and thus produce better labels than the original classifier, the Oracle must put in bigger inspection efforts, which costs more time. A solution for mitigating the delay problem is to bridge the timing gap between the two classifiers, i.e., to add an intermediate instance that can produce better labels than the original classifier (but not as good as the oracle) and that is faster than the oracle. The labels produced by this intermediate instance are considered pseudo-labels and they are only valid until the oracle produces the actual ground-truth labels. The idea behind that is to eagerly update the model by speculatively predicting drift occurrence. Figure 4 illustrates the case of a pipeline with an eager retrain configuration.



4].pdf

Figure 4: **Triage-Oracle-Eager-Update Architecture.** A secondary classifier is used to provide pseud-labels to the drift detector to mitigate the effect of delayed ground-truth labels by the oracle.

The incoming sample is classified by the fast triage classifier, but it also enters a second classifier queue. This second classifier will produce pseudo-labels that will be temporarily used by the concept drift detector. Meanwhile, the sample is also added to the sandbox queue. When the sandbox produces the definitive label, it is sent to the concept drift detector again. This allows the correction of any wrong label that might have been speculatively generated by the second-layer classifier.

4 Methodology

To evaluate the best ML pipeline architecture for each scenario, we implemented all discussed architectures and performed experiments to exercise them. In this section, we present the methodological setting developed to perform the evaluation experiments.

Classifiers and Features. We represented the malware samples via textual features derived from their Android packages, based on the implementation provided by [**?**]. All experiments we present are based on the Random Forest (RF) classifier. We adopted the RF classifier because it is the most adopted classifier considering all related works and also the classifier which typically leads to better detection results in most experimental settings [**?**].

Target Detection Rate. All presented experiments have a False Positive Rate (FPR) lower than 1% to reflect the real-world requirements, in which FPs are considered extremely harmful. Classifiers that presented FPRs higher than 1% were discarded, such that we ensure that all presented results meet the target FPR.

Tested Drift Detectors. This work's goal is not to compare drift detectors but to show the role of drift detectors in a pipelined architecture. Thus, we selected for our experiments the DDM and EDDM detectors, since they are the most popular drift detectors and have been used in related works [**?**], thus working as ground truth. We highlight, however, that our proposal is agnostic to the used drift detectors, thus also being able to work with other detectors proposed in the literature [**?**, **?**, **?**]. Whereas the frequency of drift detections might be different for each type of detector, our goal here is to shed light on the overall phenomena of label delays and limited queue sizes, that are present for any number of detections.

Testing Dataset. We considered the DREBIN [**?**] dataset of Android malware in our experiments. This dataset has been widely studied in the literature [**?**] and thus the classification phenomena are well known [**?**], which allows us to claim differences to be due to our proposed architecture and not due to unknown dataset effects. We highlight that we do not claim the prevalence statistics of the DREBIN dataset to generalize to any dataset, but that the drift phenomenon well known to happen in DREBIN is common to any drifting scenario (see App. D).

Streaming. This work models the malware detec-

tion problem as a data stream. We rely on a version of the DREBIN dataset that is temporally ordered based on VirusTotal's first-seem dates. We used the temporal order information to split the dataset into multiple epochs of different numbers of samples each to test the effect of time over the classification while ensuring that new malware characteristics are coherently appearing to the stream. In our experiments, we considered 250 epochs of 500 samples.

Drift detection and epochs. Most ML works detect concept drift via sample-by-sample checks. This is unrealistic since companies cannot have a view of the entire stream of millions of malware samples every day over the years. Thus, we adopted a concept drift detection method based on the elapsed epochs. We check for the occurrence of concept drift for all samples within a given epoch. If concept drift is reported for any sample within a window, the window is considered to be drifting and a new classifier is trained for the next epochs. We consider that the classifier retrain is inexpensive, so it is immediately available. Although realistic retraining might take some epochs, we consider that assuming negligible retrain time was experimentally valid to isolate the effect of label delays from the delay caused by the retraining step.

Full and Partial Data Views. The used drift detectors provide two levels of drift notifications: (1) warnings, when the drift occurrence is potential; and (2) drift occurrence, when concept drift is already happening. We consider that a pipeline has a full data view if all samples between two drift occurrences are considered for retraining. This is unrealistic as it requires the AV company to store a myriad of samples. An alternative is to store the samples only when drift is about to happen. We consider that a pipeline has a partial data view when it collects samples for retraining only after a drift warning.

Class Imbalance Handling. DREBIN presents more goodware than malware, which reminds the real world, where most endpoints have few malware instances. Temporally ordering DREBIN creates an even bigger imbalance, such that models do not effectively learn at this scale. To mitigate that, we adopted random sampling as the undersampling strategy for the majority class (goodware), such that we ensured to always keep the same imbalance between goodware and malware samples over the training set. We identified the target proportion in preliminary tests and kept it over all experiments. We highlight that we kept the temporal relation between goodware and malware during every undersampling procedure, thus not snooping into future data [**?**].

Result Correctneess Assurance. Random sampling causes the experimental results to become probabilistic. To mitigate the randomness effect, all results presented in this paper are an average of 10 different executions of the same experimental setting.

Implementation Framework. All pipelines were implemented in Python via `scikit-learn` and `gensim`. The code is available at: `https://github.com/marcusbotacin/ML.delay.experiments`

5 Exploring Pipeline Designs

In this section, we present a series of experiments aiming to explore the design space of pipeline solutions. Our goal is to identify the challenges affecting the systems operations in the real world and solutions for these challenges. Our findings are presented as different lessons for future malware experiments.

5.1 Adopting a proper evaluation metric

What is the best way to measure user protection? Metrics are key for security [**?**]. However, establishing proper metrics is hard, especially in the AV scenario [**?**]. No single metric tells a whole story (See App. C), thus when one selects a metric, one also selects a way to view the problem. In our view, detection solutions should adopt a historical (temporal) view of the problem, since they aim to tackle the problem in the long term. In our view, the typical metrics used in detection evaluations (accuracy, recall, precision, and so on) suffer from a major drawback: the lack of temporal dimension. Even though one can plot their variation over time, these metrics do not provide a good understanding of how exposed to threats a user has been over time. In our understanding, a good metric should also consider if a user

has been repetitively exposed to a threat and if the detection mechanism reacted to this exposure. Thus, we propose detection strategies to be evaluated using the `exposure` metric. This paper reports all detection results as exposure values.

$$Exp(samples, days) = \sum_{day=1}^{day=d} \sum_{sample=1}^{sample=s} FN(s,d) \quad (1)$$

Equation 1 defines the absolute exposure of a set of samples (1:sample) emerging as a stream problem as the sum of the non-detected samples–the False Negatives (FNs)– in all epochs within a given timeframe (1:day). Each epoch (e.g., day) in which a sample is not detected contributes to increasing the exposure value, as a user is potentially exposed to this threat. If this sample is never detected, it keeps threatening users and thus increasing exposure. If a sample starts being detected on a given date, it stops contributing to the exposure value, but its past effect is not eliminated. Therefore, the coverage values are non-decreasing, as one can only be at most as historically protected in a given day as it was in the past.

The rationale behind this metric is that if a malware sample is misclassified as goodware, it should not count only once to the statistics, but it should count for every epoch it has been undetected by the security solution, as the user has been exposed to this threat during this whole period. Figure 5 exemplifies the use of the exposure measures over time for 5 different scenarios. All of them exhibit exponential characteristics as new exposition to the threats are accumulated over time. Small differences between the detection rates of the two settings that could not be spotted at first glance (beginning of the graph) become clear in the accumulated scenario.

$$RelExp(s,d) = \frac{\sum_{day=1}^{day=d} \sum_{sample=1}^{sample=s} FN(s,d)}{\sum_{day=1}^{day=d} \sum_{sample=1}^{sample=s} 1} \quad (2)$$

A natural extension of the exposure concept is the relative exposure, i.e. when the exposure is normalized with the maximum exposure possible (Equation 2), which is determined by considering the scenario when no detection is effective, i.e., FNs are maximum. In this worst-case scenario, the user would be exposed to all types of threats, as when using no detection solution. Due to the timing nature, the relative exposure is only defined and can only be compared over the same time frames. Figure 6 shows the relative exposure for the same previous scenario. This representation highlights the small differences between the closest curves (6:1 ratio) while still accounting for their overall exposure risk.

5.2 Adopting a realistic imbalance ratio

What is a proper baseline for evaluating detector improvements? Fully-balanced datasets tend to achieve higher detection rates, but they are not realistic. Remind that real machines might see more goodware than malware [**?**]. Thus, it is key to test how detectors perform in unbalanced scenarios. Figure 5 shows absolute exposure over time for multiple scenarios. We varied two parameters: threshold (the model's confidence in the decision) and dataset balance (the ratio of goodware to malware samples). We present exposure for the worst and best cases considering the two-dimensional parameters. We omitted the parameter combinations that either (i) are intermediate values between the worst and best; or (ii) did not reach the target 1% FPR. We present results for splitting the dataset into 500 epochs because it allows observing multiple effects with increased readability. Other scenarios are shown in App. A.

The worst-case scenario for user security is the maximum exposure possible, which is achieved when no detection is performed, i.e., all new malware samples are classified as goodware. As expected, the exposure is accumulated exponentially over time, as their false negatives are counted every epoch. We notice that the scenarios with tighter threshold values (80% and 90%) are closer to the maximum exposure. The tight threshold causes these detectors to have a few to no False Positives (FPs), but they also have little to no detection capabilities, regardless of dataset balance. The classifiers using laxer thresholds achieved greater detection results, but the dataset balance also has a significant influence: Establishing a relation of 6:1 between goodware and malware sam-

ples leads to the best results (lower exposure) for this dataset. The results are even better than the natural distribution of the DREBIN dataset (represented by *). This is explained by the fact that the natural temporal distribution of malware and goodware in the DREBIN dataset significantly changes over the epochs.

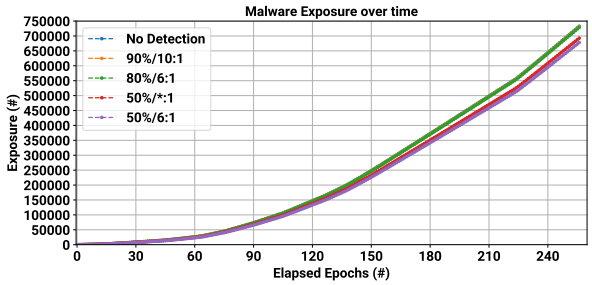


Figure 5: **Absolute Exposure.** Exposure variation for different settings of classifier threshold and dataset balances.

What does a relative exposure show? Figure 6 describes the same scenarios as above but now in terms of their relative exposure to highlight the differences between the curves. In comparison to the previous representation, it highlights that the highest detection curves were always at a significantly lower exposure level than the maximum exposure value. Although the maximum exposure is increasing over the epochs, the relative exposure for these curves is reasonably constant, which shows the number of detected samples increases proportionally to the number of new samples. If the detection rate decreases over time, the relative exposure tends to increase to values closer to the maximum relative exposure possible. This is the scenario observed for the lower curves in the graph.

5.3 Adopting a proper updating mechanism

Problem and goal: Concept drift is a known issue, but how to measure its impact on the users? Attackers update their strategies over time, thus updating the defenses is also key. We here evaluate

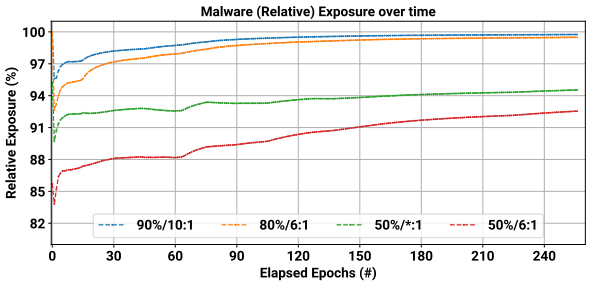


Figure 6: **Relative Exposure.** Exposure variation for different settings of classifier threshold and dataset balances.

how effective retraining strategies are. Figure 6 illustrates a scenario with two sets of curves: (i) the ones on top present a very stable behavior, which is tied to the fact that their detection rate is constantly low, thus the curves are always close to the maximum exposure possible. (ii) the lower curves present greater detection capabilities, thus they are farther from the maximum exposure. We notice that the curve of the greater detection capability (the lower one in the graph) is slowly increasing its value, getting closer to the maximum exposure value. It means that this detector is losing its detection capabilities as new samples are being considered, which is called `concept drift`.

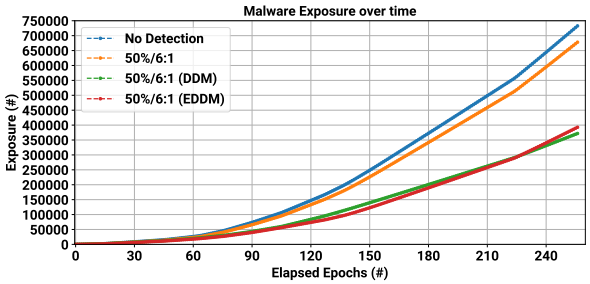


Figure 7: **Absolute Exposure and Concept Drift Mitigation.** The exposure is reduced by half when models are retrained upon the detection of concept drift occurrence.

Figure 7 shows the absolute exposure over time for the (i) worst scenario; (ii) best scenario without

drift detection; and (iii and iv) the best scenarios for DDM and EDDM. Our goal is not to compare drift detectors (see a comparison in [**?**]) but to show a clear way to measure their impact via the exposure metric. The exposure metric makes it clear that retraining at concept drift detection significantly reduces user exposure over time. While the exposure metric is still exponential in both scenarios (with and without drift detection), it grows much less in the scenario with the drift mitigation (reaching ≈50% of the previous exposure value).

What is bigger: the impact of parameter tuning or retraining on drift? The exposure metric highlights that using a drift detection strategy is more effective than any threshold or balance value might achieve for a scenario without drift mitigation since sample evolution is the dominating effect. It also highlights that the difference between DDM and EDDM is smaller than the difference between the worst scenario and the best detector possible without a drift detector, as previously presented. This shows that having a mitigation for drift occurrence is a major effect over the small differences between the different types of detectors.

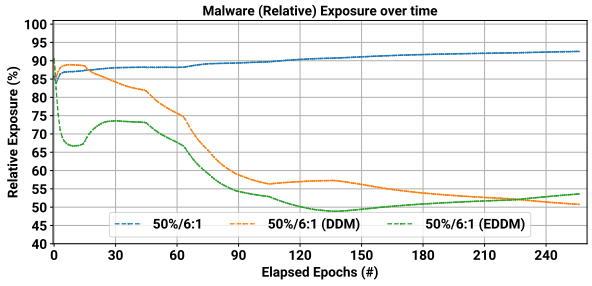


Figure 8: **Relative Exposure and Concept Drift Mitigation.** The relative exposure has been decreasing over time, thus indicating that more samples are detected over time than new rates are added to the stream.

Do the differences between drift detectors matter face to the overall drift effect? Figure 8 shows the relative exposure over time for the (i) best scenario without drift detection; and (ii and iii) the best scenarios for the two tested drift detectors (DDM and EDDM), computed with relation to the (iv) worst scenario, with maximum exposure. It clearly shows that the drift mitigation strategy is effective, as the curves for both drift detectors decreased their relative exposure values over time. It shows that the system has been gaining the ability to detect more samples over time, either by detecting more of the new samples or by detecting more of the previously seen but so far undetected ones. In comparison, the curve for the best scenario without concept drift mitigation presented the usual behavior of stability and small increase, thus showing that it has been losing its detection ability over time. The relative exposure metric allows one to enter the details of the behavior of the drift detectors. We notice that the drift detectors experienced two different scenarios: (i) an initial step with more frequent changes (more frequent drift events) to which the detectors had to adapt; and (ii) a more stable scenario in the long term, where the exposure becomes more constant in comparison to the previous case, as the models have already enough samples to perform detection more stably. This metric also highlights the small differences between the two detectors: while the EDDM's more proactive drift detection policy led to better results in the initial phase, with multiple drifting points, it has been revealed to not be the best strategy for the most stable scenario, where the DDM's conservative policy of only retraining on a drift event detected with high confidence was more effective.

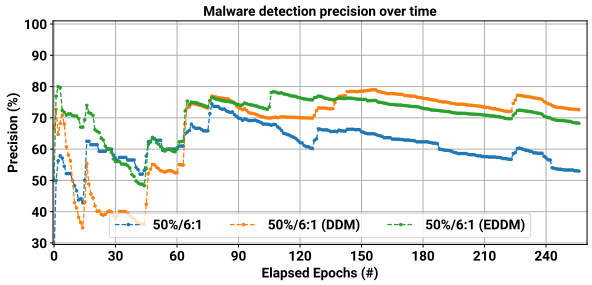


Figure 9: **Classification Precision and Concept Drift (CD) Mitigation.** Whereas one can notice that the precision of the scenario with CD mitigation is greater than the one without it, this metric is not as clear as the exposure metric in highlighting the impact of the mitigation.

Why can't we just measure the drift impact using ordinary metrics? Figure 9 shows the classification precision values over time for the (i) best scenario without drift detection; and (ii and iii) the best scenarios for the two tested drift detectors (DDM and EDDM). Remind that the precision values are affected by the heavy data imbalance in the DREBIN dataset. It shows that the classification process had an initial step with high variation and a second step with more stability, as also shown by Figure 8. Whereas the achievement of a more stable classification step is observed also in the scenario without drift mitigation–which shows it is an intrinsic characteristic of the dataset–the drift mitigation effect can be noticed by the fact that the precision values for the two drift detectors are superior than for the scenario without drift detection. This graph also shows that there are some differences between the two drift detectors. However, this type of representation is not clear in demonstrating the impact of the retraining strategy over the whole process. Although the precision value for the scenarios without drift mitigation is lower, one cannot have a clear notion of how this impacts detection over time. The final difference in the score for the last epoch is 20%, rather than the 50% reported for the accumulated exposure value from Figure 7, which reinforces our claim for metrics that look temporally to the detection process.

5.4 Remind that AVs have partial data views

How do storage constraints affect detection rates? Previous experiments assumed that AVs can see an entire history of samples when retraining models. In practice, due to storage and processing constraints, AV's history size is limited, such that it is key to evaluate its impact. We tested the scenario in which AVs only have a view of the samples collected during drift warnings (partial data view), as described in Section 4.

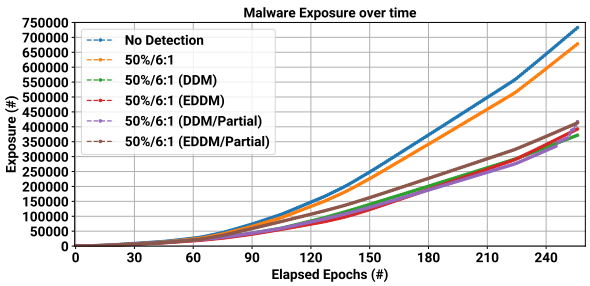


Figure 10: **Absolute Exposure and Concept Drift Mitigation with Partial Data View.** Exposure is less mitigated in the scenario with a partial data view.

Figure 10 shows the absolute exposure over time for the (i) worst scenario; (ii) best scenario without drift detection; (iii and iv) the best scenarios for the two tested drift detectors (DDM and EDDM) with full data view; and (v and vi) the best scenarios for the two tested drift detectors (DDM and EDDM) with limited data view. It shows that having a partial view limits the classifier's retraining capacities, such that they learn less about the malware samples than when having a full view, which leads to greater exposure. This effect is more noticeable for the EDDM classifier than for the DDM one. Although the exposure values are not the same as for the ideal scenario for drift detection, the results are clear in showing that it is still beneficial to use a drift detector with a limited data view than using no drift detector.

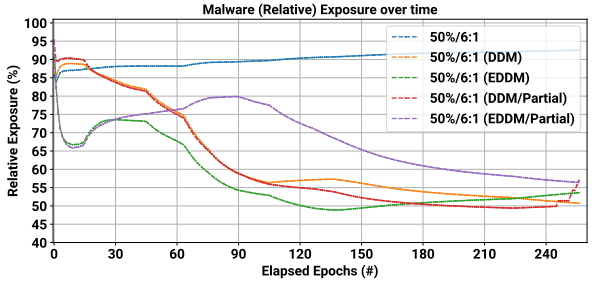


Figure 11: **Relative Exposure and Concept Drift Mitigation with Partial View.** Highlighting the intervals in which the partial view of the drift effect limited the exposure reduction.

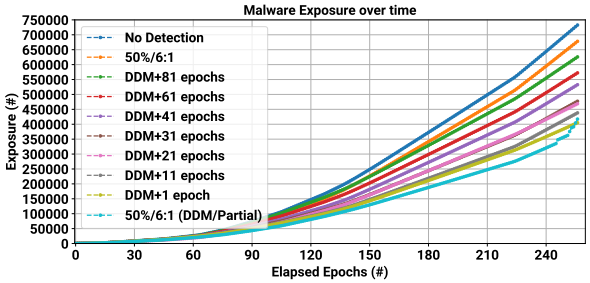


Figure 12: **Delayed ground-truth labels for concept drift detection.** Results for the DDM drift detector.

Are different drift detectors affected differently by partial data view? Figure 11 shows the relative exposure over time for the (i) best scenario without drift detection; (ii and iii) the best scenarios for the two tested drift detectors (DDM and EDDM) with full data view; and (iv and v) the best scenarios for the two tested drift detectors (DDM and EDDM) with limited data view. All relative values are computed in relation to the (vi) worst scenario, without detection. The figure clarifies the behavior of each drift detector and helps to highlight the effect of the partial view. The EDDM model was more affected by having a partial view because of its aggressive posture in predicting drift. Being aggressive with fewer data increases the chance of mistakenly missing drift points. This can be seen clearly via the increase in the relative exposure in the [30-100] interval. After that, the drift detector recovered its capacity, and the relative exposure decreased, achieving the stability rate.

5.5 Remind that labeling takes time

How do delays in labeling affect detection results? Previous experiments assumed that Oracle labels were immediately available. In practice, due to storage and processing constraints in the analysis queues, Oracle labels might arrive only after some time, such that it is key to evaluate its impact.

Figure 12 shows the absolute exposure over time for the scenario where the drift detector for the DDM classifier is exposed to ground-truth labels at different delay rates, according to a different number of epochs. We varied the epochs in chunks of 10 and plotted the biggest variety possible while keeping the didacticism of the figure. Delaying the arrival of ground-truth labels by only one epoch is already enough to cause an observable impact on the absolute exposure value. It happens because the samples whose detection was missed in a given day cumulatively account for the total exposure metric. As expected, the impact of the delay grows proportional to the number of epochs the label is delayed (i.e., for $\forall delay_x/delay_A > delay_B \rightarrow exposure(delayA) > exposure(delayB)$), such that $exposure(delay_{11}) > exposure(delay_1)$ and so on. On average, in the presented scenario, for every 10 epochs by which the labels are delayed, the absolute exposure increases by 50K points. In other words, for every 10 epochs of delay we can mitigate, the total exposure is reduced on average by 6.66%. After 81 epochs of label delays, the exposure is increased by 50% in comparison to the scenario with immediate label availability. In the presented scenario, 81 epochs correspond to $\approx 1/3$ of the entire stream. Delays greater than 81 epochs (e.g., 91, 101, and so on), take the curve to the same value as the scenario without drift, which implies that drift mitigation is not effective at this scale anymore.

Does the situation change with different drift detectors or is it a general phenomenon? Figure 13 shows the absolute exposure over time for the

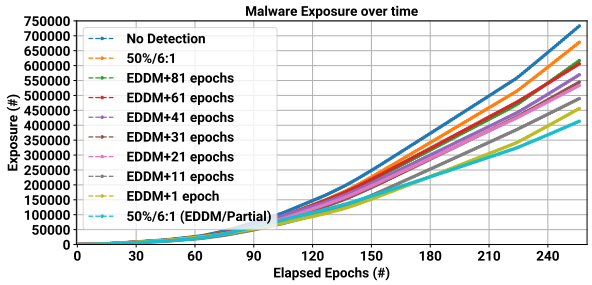


Figure 13: **Delayed ground-truth labels for concept drift detection.** Results for the EDDM drift detector.

scenario where EDDM is exposed to ground-truth labels at different delay rates (in epochs). We varied the epochs in chunks of 10 and plotted the biggest variety possible while keeping the figure readable. A single epoch of delay already causes a noticeable increase in the absolute exposure. Unlike DDM, EDDM's differences manifest more in the long term. It is expected because EDDM tries to detect the drift early, so a single epoch of delay makes no significant difference in the short term, as the model can have another opportunity to detect drift in the following epochs. However, in the long-term, the EDDM detector misses many opportunities to speculative retrain the classifier, which causes the difference in the long term. In this scenario, the exposure also grows according to the delay epochs. On average, a 60K exposure increase is observed every 10 epochs. However, the difference is not well distributed, with a concentration after 21 epochs, thus showing the limits of the delay effect over this detector. The total exposure increased by 50% after 61 epochs of delay. Delays greater than 61 epochs (81, 91, 101, and so on) take the curve to the same value as the scenario without drift, which implies that drift mitigation is not effective at this scale anymore. The results for the two drift detectors show that the delayed label problem has a significant effect on the exposure that cannot be neglected in the evaluations. Also, DDM and EDDM presenting the same effects show that label delays are a major effect, being more important than any small differences between them.

5.6 Remind that AV backends can be diverse

Can pseudo-labels mitigate the effects of label delays? When designing an ML-based classification pipeline, it is key to keep in mind that different restrictions are present in the end-point machines and at the AV company's backend. Therefore, classifiers can be deployed in different settings in these two environments. Following this idea, one can deploy an enhanced version of the main classifier in the company backend and use it as a pseudo-label generator to mitigate the impact of label delays.

To evaluate the effectiveness of this strategy, we repeated the previous experiments–varying delay times in periods of 10 epochs–but now have a second classifier that produces pseudo-labels to mitigate the label delay effect. Although the secondary classifier can be selected with a larger degree of freedom in comparison to the main classifier, we limited our experiment to varying the dataset balance from the main to the secondary classifier. Therefore, in our tests, the secondary classifier is an exact copy of the main one, but without the need to downsampling the training set as in the main model. Our goal was to simulate a real-world scenario where the main model has stronger storage constraints than the AV backend. With this formulation, we isolate variables and claim that any difference in classification power between the models is due to the different dataset balances and not due to model differences, thus reinforcing our claim on the important impact of different architectures in the classification process.

In our experiments, label delay was mitigated in all scenarios, but at different levels. Figure 14 illustrates the exposure values for the scenarios with 21 and 71 epochs of delay, respectively, in comparison to the scenarios with no delay and with no delay mitigation. The scenarios with 21 and 71 epochs of delay were selected to illustrate the extreme scenarios didactically. The exposure values for all delay values are shown in Appendix B. We notice that, in all cases, the exposure for the scenarios with delayed labels is greater than in the scenarios with no delay. However, the curves for the scenarios with delay mitigation have smaller exposure values than their

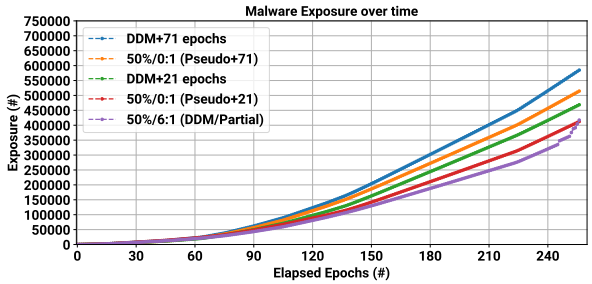


Figure 14: **Delay mitigation via Pseudo-Labels.** Pseudo-labels completely mitigate the delay effect for small delays. The greater the delay, the less effective the pseudo-labels are.

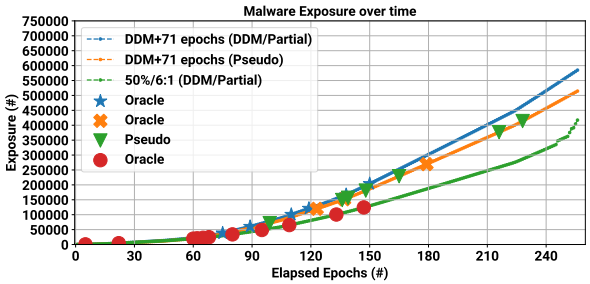


Figure 15: **Drift Points.** New drift detection points are added due to the use of pseudo-labels.

respective curves with delay, thus showing the effectiveness of this strategy. As expected, the impact of the pseudo-labels is different for each one. The scenario with 21 epochs of delay has greater mitigation than the scenario with 71 epochs of delay. This scenario is naturally more challenging because there are fewer opportunities to update the main model and the secondary model.

Do pseudo-labels actually cause retraining? To confirm that the exposure mitigation happens due to the use of pseudo-labels, we compared the drift points of the original curve and the delayed curves, with and without mitigations. Figure 15 shows the curve for the scenario with 71 epochs of delay, which was once again taken as an illustrative example. It shows curves and drift points for the original classification– with no delay due to the use of a perfect oracle, thus no need to use pseudo-labels–and for the scenario with 71 epochs of delay, but mitigated with pseudo-labels. It also shows the curve for the scenario with 71 epochs of delay having only the true Oracle updates.

In comparison to the scenario with no delay, the scenario with 71 epochs of delay has fewer drift points, since there is less information available at each verification time. We also notice that after some point (≈180), only drift points due to pseudo-labels are observed in the curve for the mitigated delay scenario. which justifies the decreased exposure for it. It is important to notice the fact that the last drift point in the base curve is ≈ 140, which should cause a delayed drift point at $a \approx 211$ in the delayed curve, but it did not happen. It shows that delayed and pseudo labels not only anticipate drift points but also change the drift dynamics of the scenarios and eliminate the need for drift retraining in certain points.

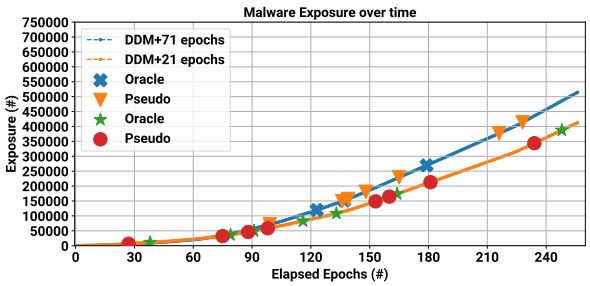


Figure 16: **Drifts over Time.** The smaller the delay, the more opportunities for the oracle to act. Pseudo-labels act when no Oracle data is available.

Although the use of pseudo-labels partially mitigates the effect of label delays, we notice that the problem is still significantly tied to Oracle updates. Figure 16 compares the scenarios with 21 and 71 epochs of delays, both with delay mitigation via pseudo-labels. The exposure for the scenario with 71 epochs of delay is still greater than for the scenario with 21 epochs, although at a smaller scale than the original scenario due to the mitigations. We notice that for the scenario with 21 epochs of delay, there are not only more retrain points due to Oracle updates but also more retrains due to pseudo-label updates.

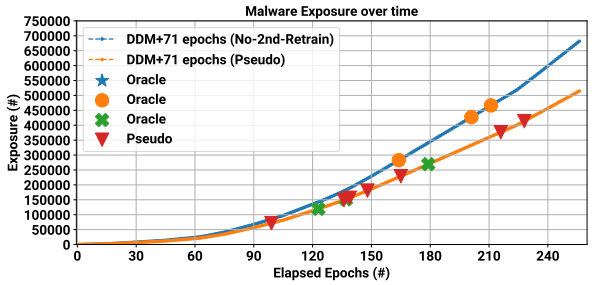


Figure 17: **Classifier update strategies.** The secondary classifier must also be updated at drift points to not degrade the classification performance with poor pseudo-labels.

It happens because although stronger than the main model, the secondary model still has limited classification power in comparison to the Oracle, whose updates are still key to exposure mitigation. The more Oracle updates the secondary model receives, the more the secondary model corrects the main one.

Should the secondary classifier be updated? Previous results highlighted the importance of updating the secondary classifier when the first classifier is updated. This ensures that the differences between the two classifiers are only in the imbalance, not the training set. If the second model is not updated, it starts to suffer concept drift, and its prediction might even cause more harm than good to the main model. Figure 17 compares the scenario for 71 epochs of delay chosen as representative for the cases in which the second model is and is not updated. We notice that at the beginning of the prediction the effects are not so noticeable–the drift effect is even masked by the greater classification capacity of the secondary model. However, over time, the curves tend to differentiate, and the scenario without updates presents a greater exposure than the scenario with updates. Also, the number of times the secondary model causes a retrain due to drift identification is small since the model lost its ability to identify prediction errors.

6 Detection Under Realistic Limits

Problem and Goal: We previously discussed the effects of different architectures on classification results, but there is still a question to be answered: *How do these effects happen in practice?* Or, in other words, *How does it affect the design decisions of a security engineer?*

We identified two assumptions that must be revisited: the queue size and the policy for queueing samples. The first major non-realistic assumption of most architectures is that all samples can be classified by the oracle. In practice, analysis queues are limited and do not support all samples at once. If all samples are sent to the queue, this is the cause of the label delays, as the last samples in the queue will only be analyzed in further epochs. We simulated multiple scenarios to demonstrate and evaluate its impact.

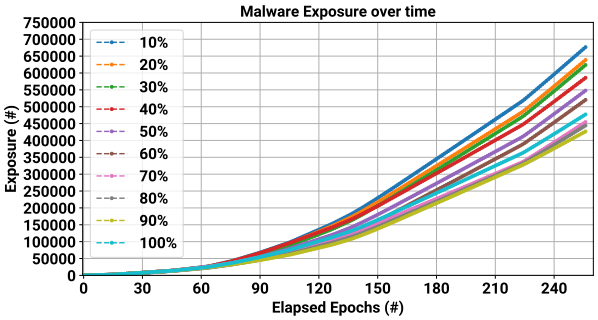


Figure 18: **Exposure vs. Queue Size (Oracle-Only).** Queue size as a proportion of the samples per epoch. The more limited the queue size, the greater the exposure.

How does the exposure vary with the different queue sizes? Figure 18 shows exposure values for different queue capacities as proportions of the input. Since our base scenario considers 500 samples per epoch, a 10% capacity means that the queue handles 50 samples per epoch, and a 100% capacity means that all the 500 samples are handled at once (ideal scenario). In this experiment, queued samples are directly sent to the oracle, with no pseudo-label usage to mitigate delay effects. It implies that sam-

ples queued after the queue reaches its full capacity remain in the queue for the next epochs. We observe that although different exposure values are observed for each configuration, we can draw a general conclusion: the exposure grows inversely proportional to the queue size, i.e., the bigger the queue, the lower the exposure, as one could hypothesize.

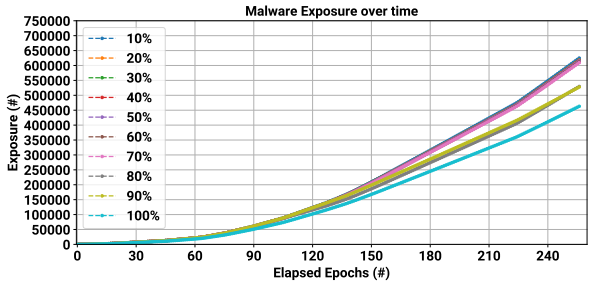


Figure 19: **Exposure vs. Queue Size (Pseudo-Labels).** Queue size as a proportion of the samples per epoch. The smaller the queue size, the greater the exposure. Pseudo-labels mitigate the exposure growth in comparison to Oracle-only.

Do pseudo-labels mitigate the impact of small queue sizes? Figure 19 shows exposure values for the distinct queue capacities in a scenario analogous to the previously presented but now considering pseudo-labels for delay mitigation. In this new scenario, the exposure is still highly correlated with the inverse of the queue capacity. However, we notice that the relation is not linear, but quantized, i.e., the growth happens in blocks. This phenomenon is expected by the fact that pseudo-labels can mitigate the label delay effect to a significant extent within a given range of queue size. Therefore, all exposure curves tend to cluster around the same values. After a given reduction in the queue size, the power of the pseudo-label classifier is not enough anymore to keep exposure on the previous level, however, it can also mitigate delay effects to a given extent within the new range, until the reduction in the queue size is too big for the classifier to keep up with the same exposure level, then a new growth is observed. This phenomenon resulted in the three different clusters of exposure levels shown in the figure.

Table 1: **Queue size to epochs of delay mapping.** Comparing the scenarios with and without pseudo-labels.

Oracle Only					
Size	10%	20%	30%	40%	50%
Epochs	100+	100+	81	71	51
Size	60%	70%	80%	90%	100%
Epochs	41	31	21	11	1
Pseudo-Labels					
Size	10%	20%	30%	40%	50%
Epochs	71	71	71	71	41
Size	60%	70%	80%	90%	100%
Epochs	41	1	1	1	1

How much delay does a limited queue size impose? The reason why reducing queue capacity increases exposure is that more samples have their true labels identified only in future epochs, causing a delay of some epochs in the drift identification and model retraining, leaving users exposed for longer. The delay in the label delivery is exactly the scenario studied in Sec. 5.5. Thus, we can map the equivalent delay each queue capacity causes.

Table 1 presents the mapping between queue capacities and delay epochs for the scenarios with and without label delay mitigation. All epochs are identified within a 10% margin of error on the exposure value. When the queue presents the same size as the number of samples per epoch (100%), the exposure value is within the range of the scenario with no delay (or a single epoch of delay due to the error margin). However, for the scenario without delay mitigation, the growth is almost simultaneous, as every sample not immediately removed from the queue causes a delay. The growth in the delay is directly proportional to the capacity. When the capacity is reduced to 10%, the equivalent delay is more than 100 epochs. The delay is significantly mitigated by pseudo-labels. The queue size could be reduced by 30% (to 70%) and kept presenting the same exposure value. The delay could not be mitigated when the queue was shortened to 40%, where the classifier presented the same exposure as having no pseudo-labels.

Is queuing all samples the best strategy? The second major non-realistic assumption of most archi-

tectures is to assume that all samples will be queued and analyzed by the oracle. If all samples are added to the queue, including those whose main classifier has high confidence in the result, it creates an unnecessary overhead for the oracle. In this case, the queue will be full of samples whose classification by the oracle does not result in additional detection power. The best strategy would be to add to the queue only samples with the potential to cause drift, i.e., the ones with low classification confidence, thus making the best use of limited queue size.

Table 2: **Queue Size vs. Classification Confidence.** Analyzed samples per epoch. Comparing the scenarios with and without exposure mitigation via pseudo-labels.

Oracle Only					
Confidence	60%	70%	80%	90%	100%
Samples	5	41	48	261	500
Pseudo-Labels					
Confidence	60%	70%	80%	90%	100%
Samples	2	13	27	175	500

In this experiment, we verified how long the queue size must be to fit the samples whose confidence level is smaller than a given threshold. Table 2 shows the average number of samples in the queue according to the threshold set for the main classifier's confidence level. When the threshold is 100% (our experimental ground truth), all of the 500 samples are added to the queue. Reducing the target confidence significantly reduces the number of samples that must be added to the queue. Therefore, it is a suitable strategy for limited queue sizes. The scenarios with pseudo-labels result in even fewer samples being added to the queue due to the delay mitigation.

The experiment shows that if we only send to the queue samples with very low confidence, a very small queue is required. for instance, if only samples classified with less than 60% confidence are queued, less than 10 samples per epoch must be analyzed by Oracle, which is suitable for almost any technique. However, as a trade-off, if only a few samples are relabeled, there is only a minimal chance of drift retraining, thus the scenario degrades back to the scenario with no drift mitigation shown in Section 5.3.

Table 3: **Maximum Exposure Vs. Architectural Constraints.** The true oracle decreases more the exposure when there is space available in the queue. The pseudo-labels mitigate more the exposure growth when queue size is a constraint.

True Oracle					
Confidence Size	**60%**	**70%**	**80%**	**90%**	**100%**
10	**175%**	175%	153%	173%	172%
50	**174%**	**158%**	150%	147%	139%
200	**173%**	**143%**	**124%**	125%	**114%**
300	**172%**	**140%**	**131%**	**116%**	**104%**
500	**173%**	**129%**	**125%**	**105%**	**100%**
Pseudo-Labels					
Confidence Size	**60%**	**70%**	**80%**	**90%**	**100%**
10	**175%**	**173%**	**144%**	**144%**	**143%**
50	175%	173%	**147%**	**144%**	**137%**
200	176%	176%	150%	**122%**	122%
300	173%	172%	144%	127%	124%
500	**173%**	170%	134%	123%	111%

Is there an ideal amount of samples to be added to the queue? The previous results show a trade-off between the queue size and the ability to learn about new malware. Thus, a detection engineer might wonder what is the best design decision among keeping short/large and empty/full queues. We elaborate on this reasoning by evaluating all combinations of queue sizes and confidence thresholds in the architectures with and without pseudo-labels.

Table 3 shows exposure values for the scenarios with and without pseudo-labels. The values are normalized by the ground-truth scenario (Oracle, Full-confidence, No queue size constraint). We represent the queue sizes corresponding to the average number of samples queue for each threshold (based on the results from Table 2). The diagonals of the table represent the best scenario for each classifier in terms of constraints, i.e., the queue size is of the same size as the average number of queued samples for that threshold. Thus, the rows above a given cell represent more limited queue sizes, thus causing greater delay. The rows below a given cell represent larger queue sizes, greater than what is needed in the av-

erage case. The columns on the left of a cell represent lighter thresholds, alleviating the pressure on the queue. The columns on the right indicate laxer thresholds, that will queue more samples than the queue size, thus causing delays.

When directly comparing the scenarios with and without pseudo-labels, we notice that the use of oracles is always advantageous, since it presents a greater classification power. Queue size-wise, the effects are one-way. Decreasing the queue size while keeping the same threshold increases the exposure, as it causes label delays. In turn, increasing the queue size beyond the average number of samples does not significantly decrease the exposure (it happens only in the specific epochs that queue more samples than the average). Confidence threshold-wise, in turn, the effects are two-way. On the one hand, increasing the threshold while keeping the same queue size causes more samples' labels to be delayed, In this case, the use of pseudo-labels can at least mitigate the delay effect. On the other hand, decreasing too much the threshold to fit the queue size causes the oracle to have too few data points to retrain, thus also increasing the coverage.

Summary. it is desired to keep the queue closer to its full capacity. It is better to use the Oracle if additional queue space is available. Pseudo-labels must be used if the threshold must be increased and the queue size kept the same. The rationale here is that the extended use of the true oracle ensures that the primary model is not poisoned by incorrect labels generated by the pseudo-classifier [**?**].

7 Discussion

In this section, we discuss the implications of our findings and how they relate to the real world.

Extending the exposure metric concept. The exposure metric is non-decreasing and suppose its use in a controlled environment, which is suitable for testing purposes. For open scenarios, such as observational studies, we propose to use the `actual exposure` metric, which can have decreasing values. Exposure decrease happens in practice when a sample stops working (e.g., because its C&C was sinkhole) or is not distributed anymore (e.g., attackers shifted distribution campaigns). In these scenarios, these samples do not pose a threat anymore and should not be considered for the definition of the maximum exposure possible. The challenge to counting for the actual exposure is to identify if a sample is still active or not, which might be possible only based on other observational studies or AV companies' internal knowledge.

From experimental epochs to real epochs. A challenge faced in developing this research is to find reliable data from the field about how much delay is currently accepted by the detection industry (e.g., AV companies). To avoid making unrealistic assumptions, we adopted a simulation approach, varying the range of the number of epochs (see Appendix A). While this strategy allows us to perform varied observations, it is important to put them in a real-world context. The interpretation of the specified epochs might vary from days to hours, depending on the application scenario. Whereas all results are experimentally valid, their extrapolation to reality might be more plausible in some scenarios than others. For instance, if epochs are mapped to days, delays of 300 days might not be relevant, as attackers tend to quickly change their campaigns [**?**]. On the other hand, if epochs are mapped to processing jobs, multiple epochs might be considered in a day, and scenarios with thousands of epochs become realistic.

Future Works. We believe that our approach can be generalized to operate in any scenario that presents concept drift. Therefore, in future works, we will deploy it in other domains, such as IoT networks [**?**].

8 Related Work

In this section, we present related literature works to better position our contributions and to support our claim that many assumptions made by many literature works are too ideal for application in real-world scenarios.

Realistic Pipelines. All research works have to make assumptions about the study's non-core aspects. However, it is hard to draw a line on to which

extent assumptions are realistic. The security literature only recently started to question with a greater emphasis its assumptions [?, ?], a research effort this paper is part of. In this sense, the closest related work to ours is the proposal for a more realistic dataset [?], that questions many common assumptions (e.g., no presence of obfuscated samples). Our work extends over it by shifting the focus from the dataset to the security pipeline, emphasizing more previously uncovered aspects, such as the delay inherent to limited analysis queue sizes.

Global vs. Local views. Our work differentiates from previous solutions by not supposing that the AVs can have an unlimited, centralized view of the malware operation [?]. Instead, we assume that the AV front-end and back-end are different and that each one has its own constraints.

Security metrics. Multiple metrics have been proposed over time to evaluate systems' detection capabilities [?]. Their major drawback is not (i) considering the effect of time and (ii) cumulative counting misdetection effects, thus resulting only in a partial view of the risk a user has been facing. We mitigated this problem by proposing the exposure metric. We are the first to propose a single-score metric to account for the entire detection risk a user faces.

Dataset balances. Despite being a known issue, a minority of research works consider that datasets will be imbalanced [?]. When they do so, they often do not consider drift occurrences. Concept drift is mostly considered on balanced datasets. The few works that consider drift effects on imbalanced datasets [?] typically consider a single classifier with the same imbalance over the whole pipeline. However, modern AVs are multi-stage [?] and their models might have different imbalances at the userland and the cloud. We here associate different imbalances with different pipeline stages to show that there is no one-size-fits-all balance ratio.

Concept drift. Although many works in the literature propose to handle concept drift and evolution [?, ?], it is still possible to find many proposals of newer pipelines without drift detection capabilities [?]. Drift detection usually comes along with active learning capabilities [?]. A major drawback of these proposals is to suppose that an unconstrained number of samples will be available for drift detection. In reality, AVs can keep a limited queue of previously seen samples in their drift detection pipelines.

Delayed labels. Drift detection requires to have actual labels to compare with the predicted ones. Whereas most works assume these labels are immediately available, these labels must arrive at a much later point than the drift point. Label delays might occur due to: (1) Sandbox queues [?]. Sandboxes are of limited scalability in comparison to the number of samples daily received by AV companies, such that it is common for samples to wait a large time for an execution slot; and (2) Human analysis queues [?, ?]. Human analysts are even more scarce than sandbox time. If a malware sample depends on human analysts to be correctly labeled, it might take a long time until a drift is noticed. Most works in the literature do not account for the label delay problem and just assume that labels are immediately available, which does not correspond to reality. This unrealistic assumption can be seen even in modern active learning-based pipelines [?, ?].

Whereas some works try to argue that detection latency is not a problem [?], others acknowledge it is [?], but do not provide a concrete measure to handle it. Information from AV companies state that human analysts can only handle around 80 queries a day [?], which is a significant limitation. Despite that, the information about how much delay it caused in the actual AV pipeline is not available and the same study assumes that ground-truth labels are immediately available. A complementary study suggests that a 1-week delay is an acceptable trade-off by AV companies [?]. Longitudinal studies of AV detection behaviors state that for 50% of the AV, the response time is around 19 19 days [?]. In this work, we present simulation results to understand the impact of different amounts of delay over a detection pipeline, including the scenarios compatible with the ones reported in these previous works.

Pseudo-labels. The use of pseudo-labels in the detection context has been proposed before, but it is mostly focused on accuracy aspects, not on label delay mitigation. For instance, previous work leveraging pseudo-labels was concerned about assigning malware to unlabelled families [?]. Using pseudo-label

along with drift detection is often more seen from the poisoning effect perspective [**?**] than for label delay mitigation, as here proposed.

9 Conclusion

In this work, we investigated how current research assumptions of most ML-based malware detection pipelines do not resemble real-world constraints. More specifically, we shed light on the label delays problem, i.e., on the fact that the ground truth labels used by online retraining solutions are not immediately available, which is often neglected by these approaches. We here demonstrated and measured how label delays affect all the multiple pipelines proposed in the literature (up to 75% detection decrease in our experiments). In particular, we highlight the problems of: (1) Using metrics that do not account for the effect of time–which causes results to display smaller discrepancies in experiments than they cause in the real world (20% rather than 50%, in our experiments); (2) Ideal assumptions about the amount of drift data a system can handle–which is much smaller in the real world than in experimental settings; and (3) Ideal assumptions about the availability of Oracle data for drift detection–which are not immediate in the real world. To mitigate these phenomenons, we claim the need for relying on pseudo-labels for mitigating drift-related delays, which allows recovering the detection rates in the scenario with delayed labels to values closer to the ones proposed in the original pipelines (30% loss rather than 70% loss, in our experiments). To foster more realistic evaluations, we proposed a new exposure metric that accounts by the effect of time–which can be used in future works–, and we made the source code of our experimental framework available to foster more experiments in this research direction.

Reproducibility. All developed codes for this research is available at: `https://github.com/marcusbotacin/ML.delay.experiments`

Acknowledgments. Marcus Botacin thanks NSF for the support via the CNS 2327427 grant. Heitor Gomes thanks the Marsden Fund for the award number VUW2213.

A Results for multiple timespans

To present a more comprehensive evaluation of the exposure metric and the effects of time when the classification is drifting and delaying, we repeated the experiments from Section 5 now varying the number of samples per epoch (and thus the number of total epochs).

Figures 20 to 29 show the absolute exposure values for the curves for the multiple detection strategies presented in this paper. For the delayed label evaluation, we present (i) an intermediate result to provide a sense of the average case and (ii) the last delay value possible before the delay occurrence completely mitigates the drift retraining effect. We show absolute values and limit results to the DDM method to simplify reading the presented results.

In all scenarios, we still consider the samples in the dataset ordered in a temporal manner based on their labels to ensure that new features are seen by the models in the same order as in the real world. However, we now vary the temporal distancing between these features by distributing all samples in the dataset inside different epochs. We opted to vary the number of samples per epoch from 100 (Figure 20) to 1000 (Figure 29) to cover effects observed for different magnitude orders, given the size of the tested dataset.

By varying the number of epochs we can vary the distance between two features appear. With fewer epochs to fit the entire dataset, a greater number of samples is seen per epoch, making new features appear closer, which tends to cause more drift points, but alleviates the effect of wrong classifications in the long term. With more epochs, fewer samples are seen per batch, reducing drift detection ability, and thus causing a higher impact on the final exposure result.

We notice that for all numbers of epochs, the graphs presented very similar curves, keeping the exponential characteristic of the exposure metric. In all scenarios, retraining on drift occurrence produced a significant gain, usually decreasing the exposure by half (50%). We state relative terms for comparison here because the exposure values are absolute and

depend on the number of epochs. With more epochs, the exposure grows more because a misdetected sample is evaluated over a longer period.

In all cases, detection delay causes a significant impact on the detection, but it is a less significant effect than concept drift. In all scenarios, the advantage of retraining on drift occurrence is only nullified by the delay effect when a significant portion of the dataset is considered (e.g., more than 300 epochs for the scenario with more than 1000 epochs, as shown by Figure 20). In reality, it is unlikely that a sample will wait for so long in the analysis queue. A more probable and significant effect is the limited data view for concept drift detection. This phenomenon is more noticeable the more epochs the scenario presents, as it highlights the minor differences caused in the long term by missing a few samples in a specific epoch. For the scenario with more than 1000 epochs (Figure 20), the limited view of the drift detector causes a relative exposure increase of the same proportion (6%) as delaying the delivery of the labels by 200 epochs.

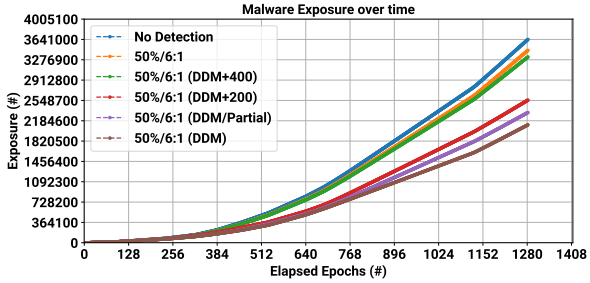


Figure 20: 100 samples per epoch (≈1400 epochs).

B Results for Pseudo-Labels

To present a more comprehensive evaluation of the effects of the delay mitigation via the pseudo-labels, we repeated the experiments from Section 5 now varying the number of epochs of label delays.

Figures 30 to 37 show the absolute exposure value curves for the multiple epochs of delay. We notice that for the scenario with 11 epochs of delay, the impact of the delay is fully mitigated by the use of

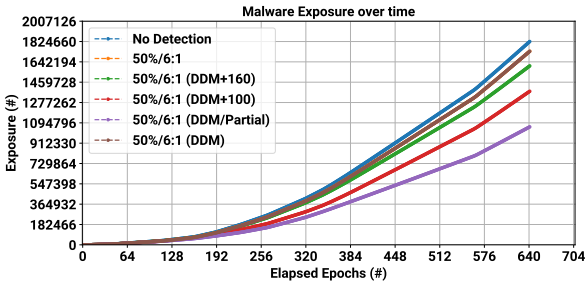


Figure 21: 200 samples per epoch (≈640 epochs).

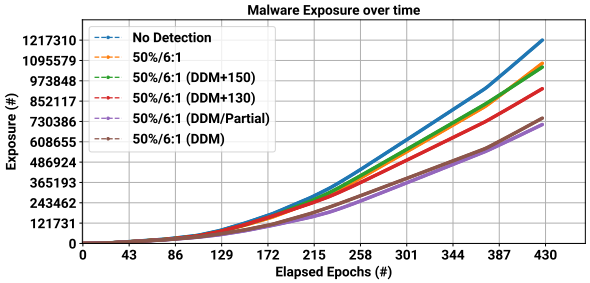


Figure 22: 300 samples per epoch (≈430 epochs).

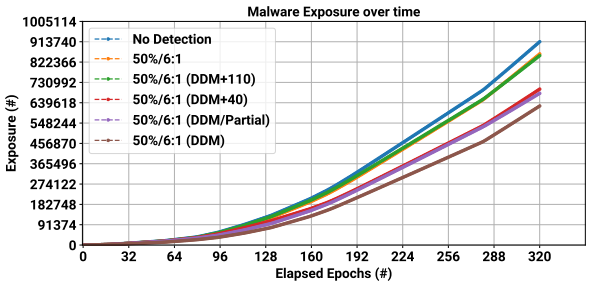


Figure 23: 400 samples per epoch (≈320 epochs).

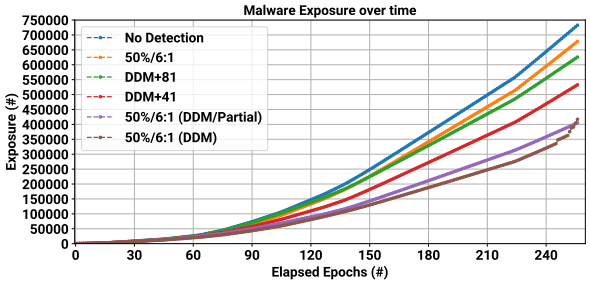


Figure 24: 500 samples per epoch (≈250 epochs).

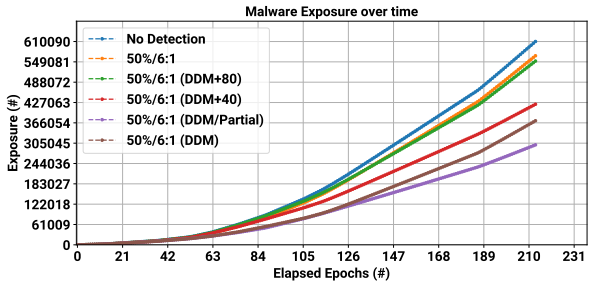


Figure 25: 600 samples per epoch (≈210 epochs).

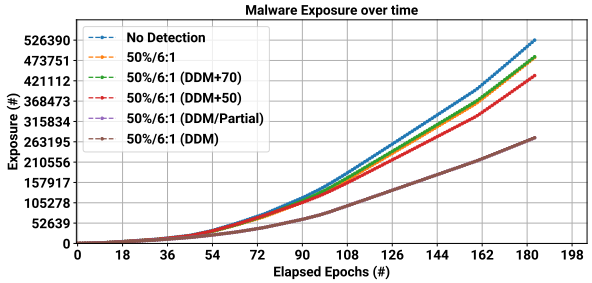


Figure 26: 700 samples per epoch (≈180 epochs).

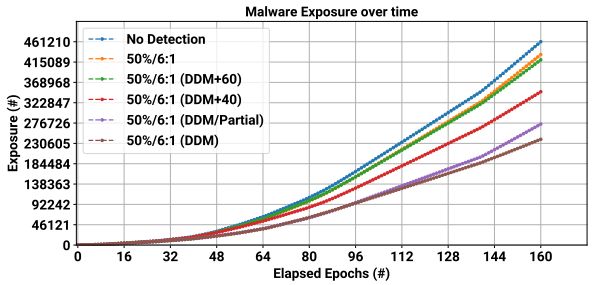


Figure 27: 800 samples per epoch (≈160 epochs).

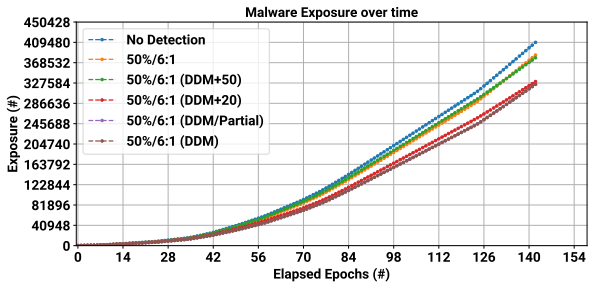


Figure 28: 900 samples per epoch (≈140 epochs).

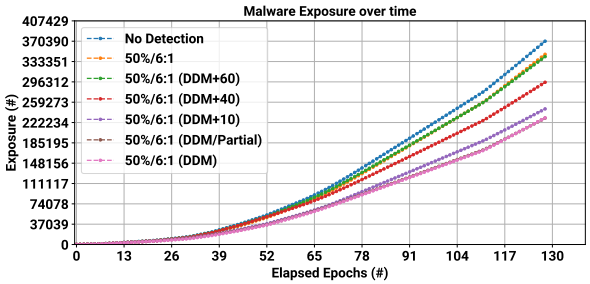


Figure 29: 1000 samples per epoch (≈130 epochs).

pseudo-labels. The reason for that is that the oracle quickly updates the secondary model, which also fast discovers deviations in the main model; prediction errors for being too aggressive are also quickly mitigated.

The delay starts to be each time less mitigated over time until the mitigation effect was null in our tests for the scenario with 45 epochs of delay. In this case, using or not pseudo-labels caused the same exposure. This is explained by the fact that the oracle was delaying the labels for a significant time and that the pseudo-labels were not powerful enough to cause retrains.

However, delay mitigation starts to be effective again for greater delay values, because in these scenarios the oracle takes too long to provide the correct labels to the point that the classification power of the secondary classifier becomes relevant once again. This result holds for all delay values.

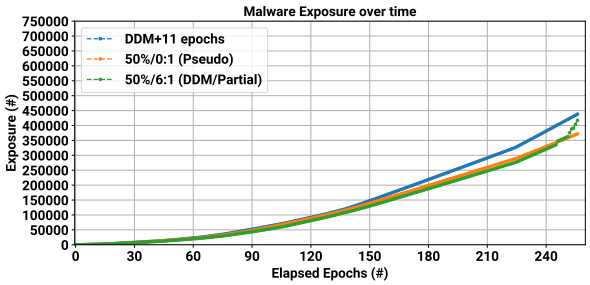


Figure 30: 256 epochs. 11 epochs of oracle delay.

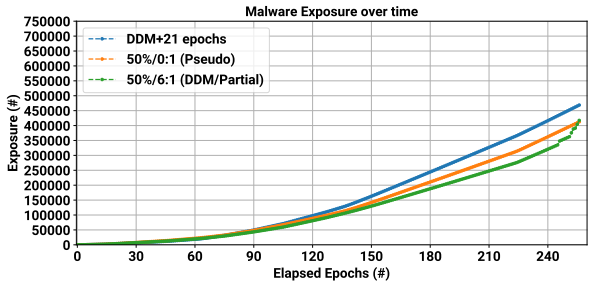


Figure 31: 256 epochs. 21 epochs of oracle delay.

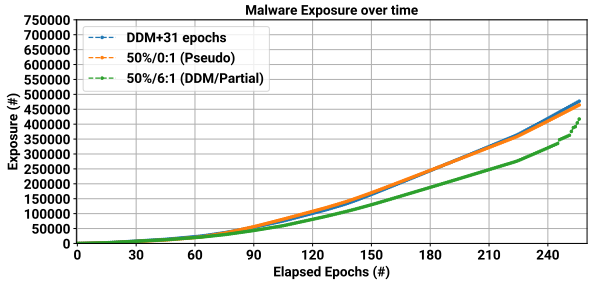


Figure 32: 256 epochs. 31 epochs of oracle delay.

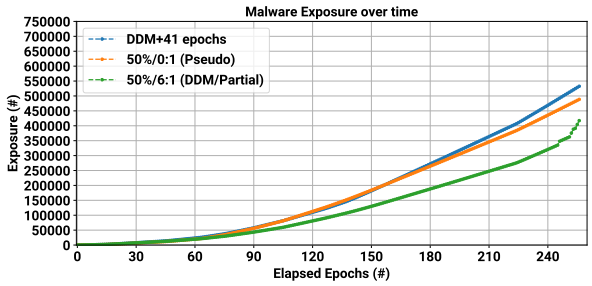


Figure 33: 256 epochs. 41 epochs of oracle delay.

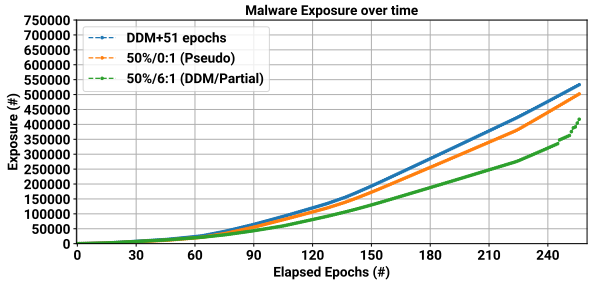


Figure 34: 256 epochs. 51 epochs of oracle delay.

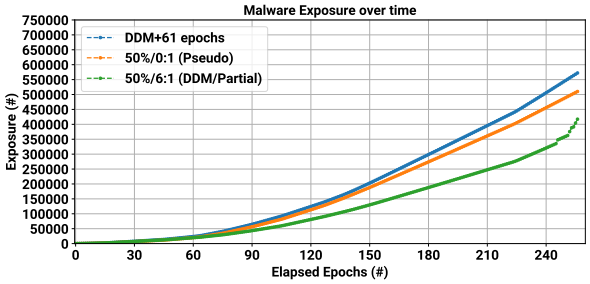


Figure 35: 256 epochs. 61 epochs of oracle delay.

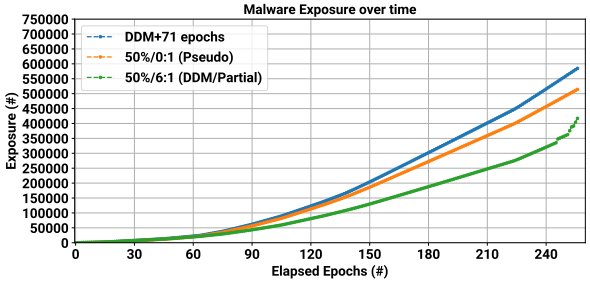


Figure 36: 256 epochs. 71 epochs of oracle delay.

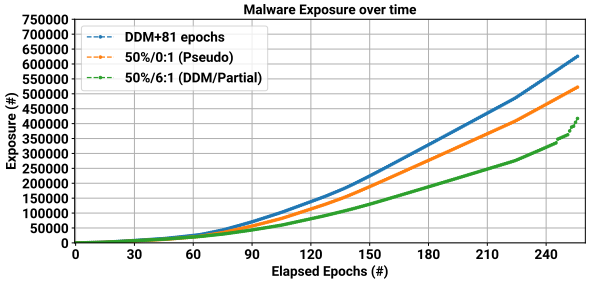


Figure 37: 256 epochs. 81 epochs of oracle delay.

C Comparing Temporal Metrics

A key contribution we propose is the use of a temporal metric (exposure) to account for the cumulative results of malware misdetection over time. In this sense, the closest proposition to ours is the Area Under Time (AUT) [**?**] metric, which proposes the trapezoidal sum of typical metrics, such as the F1-score, over time. For comparison, exposure is computed as the sum of the False Positives in a given period. Whereas operating with FPs rather than F1, precision, or recall, exposure might be understood as an extension of the AUT metric, even though it is a conceptionally non-trivial expansion.

A significant difference between the metrics is their focus. Whereas AUT focuses on the classifier, exposure focuses on the user. It leads to interpretation differences. Whereas AUT measures how good a classifier is, exposure measures how exposed to risk the user is. Exposure is a metric with a natural interpretation, as it refers to the FP, whereas the AUT might not have a proper meaning when applied to some metrics (e.g., recall). In practice, AUT and exposure tend to go in different directions. Whereas a classifier's AUT is better as closer to 1, the user is more protected as the exposure is closer to zero.

Table 4: **Metrics Comparison.** Comparing AUT and Exposure.

Delay	AUT_{Prec}	AUT_{Rec}	AUT_{F1}	**Exposure**	**1-Exp**
0	0.71	0.42	0.52	0.49	0.51
10	0.68	0.39	0.49	0.55	0.45
20	0.66	0.32	0.43	0.56	0.44
30	0.65	0.35	0.45	0.65	0.35
40	0.65	0.24	0.35	0.69	0.31
50	0.63	0.22	0.32	0.72	0.28
60	0.62	0.20	0.30	0.73	0.27
70	0.62	0.16	0.25	0.79	0.21
80	0.58	0.11	0.18	0.87	0.13
90	0.57	0.01	0.01	0.90	0.10

Table 4 illustrates the difference in the use of AUT and exposure metrics for the case of classification under delay. The scenario with 500 samples/epoch is taken as the reference. The AUT is capable of demonstrating the classification power degradation over time, as the metric decreases over time. This finding is true regardless if AUT is computed for precision, accuracy, or F1-score. However, the results are not coherent for the metrics, because precision scores degrade less than recall ones. Exposure is also capable of showing degradation over time, as exposure increases. Since each metric goes in the opposite direction, we defined the inverse of exposure (1-Exp) to be able to compare them in the same ground. In the provided example, the inverse of exposure presents values compatible with the AUT F1 scores, but the easiness of interpretation is more favorable to the first. For instance, for 80 epochs of delay, it is hard to understand how good is an AUT(F1) = 0.18 classifier, but it is possible to understand that one is protected against 13% of the historical threats.

D Extending to other datasets

This paper's key point is to discuss how the ML pipeline architecture affects malware detection results. More specifically, we evaluate the impact of label delays in the classification process. To this end, any malware dataset that presents the drift effect would suffice, as we focus more on the drift occurrence as a phenomenon rather than on its prevalence in actual samples. Thus, our claims that label delays affect drift mitigation retrain should be agnostic to the dataset. Despite the theoretical support for this claim, we stepped further and evaluated the generalization of these claims and findings for a different dataset.

We repeated all the previous experiments for the Androzoo dataset, which contains 213,928 goodware and 70,340 malware samples collected in 2016 and 2017. The samples were labeled using Virustotal and temporally ordered, the same way as the DREBIN dataset. Figure 38 shows exposure curves for representative scenarios (selected for didactic purposes). We considered the scenario closer to 250 epochs as a reference, as used in most of the paper. The dataset required, in fact, ≈ 270 epochs to fit, given its imbalance.

We notice that the curves exhibit a characteristic very similar to the DREBIN scenario, but the absolute values are significantly higher, as the dataset is

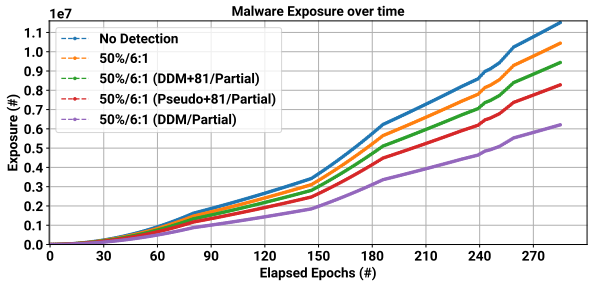


Figure 38: **Androzoo experiments.** The effects observed in the Androzoo dataset are analogous to the ones observed for the DREBIN dataset, but at a different scale, as the Androzoo dataset is much larger and complex.

bigger. There is a $\approx 10\%$ exposure reduction if no drift detection is used, and $\approx 50\%$ if drift is used. Intermediate cases appear when label delays are considered. The maximum delay allowed before complete performance degradation is achieved with 81 epochs, the same as for DREBIN.