Temporal Analysis of Distribution Shifts in Malware Classification for Digital Forensics

Francesco Zola
Vicomtech, Basque Research and Technology Alliance (BRTA)
Donostia/San Sebastian, Spain
fzola@vicomtech

Jan Lukas Bruse
Vicomtech, Basque Research and Technology Alliance (BRTA)
Donostia/San Sebastian, Spain
jbruse@vicomtech

Mikel Galar
Institute of Smart Cities, Public University of Navarre
Pamplona, Spain
mikel.galar@unavarra.es

***Abstract*—In recent years, malware diversity and complexity have increased substantially, so the detection and classification of malware families have become one of the key objectives of information security. Machine learning (ML)-based approaches have been proposed to tackle this problem. However, most of these approaches focus on achieving high classification performance scores in static scenarios, without taking into account a key feature of malware: it is constantly evolving. This leads to ML models being outdated and performing poorly after only a few months, leaving stakeholders exposed to potential security risks. With this work, our aim is to highlight the issues that may arise when applying ML-based classification to malware data. We propose a three-step approach to carry out a forensics exploration of model failures. In particular, in the first step, we evaluate and compare the concept drift generated by models trained using a rolling windows approach for selecting the training dataset. In the second step, we evaluate model drift based on the amount of temporal information used in the training dataset. Finally, we perform an in-depth misclassification and feature analysis to emphasize the interpretation of the results and to highlight drift causes. We conclude that caution is warranted when training ML models for malware analysis, as concept drift and clear performance drops were observed even for models trained on larger datasets. Based on our results, it may be more beneficial to train models on fewer but recent data and re-train them after a few months in order to maintain performance.**

***Index Terms*—Concept drift, temporal analysis, malware classification, forensic exploration, explainability**

I. Introduction

Malware represents a particular piece of code whose aim is to alter normal program behaviour. Today, malicious software is more disruptive than ever with more than 450,000 new malicious programs (malware) and potentially unwanted applications (PUA) being registered every day [1]. These malware programs are frequently used for cyberattacks aimed at stealing information, deploying ransomware, or carrying out other illegal operations or cybercrimes. Furthermore, their versatility allows one to adapt and use them in a wide range of devices such as the Internet of Things (IoT) and mobile domains [38], [48].

Modernization of security software and detection techniques have led hackers and malicious operators to develop new methods for bypassing firewalls, spreading malware through a system, and injecting backdoors [46]. This trend of creating new malware variants based on techniques that are able to evade "traditional" detection systems such as anti-virus scanners [13], has generated an increasing amount and diversity of malware applications. Due to the increasing complexity and range of malware functionalities, it is not only important to detect malware but also to distinguish the (growing number of) different types of malware *families* in order to gain a better understanding of current and future malware capabilities, and their possible impacts [19]. Given this scenario, malware detection and classification has become a key objective within the field of information security [32].

Machine learning (ML) algorithms are commonly used to detect malware elements [36]. These models are trained to extract, analyze, and classify source programs as benign or malware. However, the majority of studies in recent literature [34], [36] are purely focused on the classification task - typically racing towards the best classification performance while neglecting the interpretation of the results. More and more complex ML solutions are implemented to squeeze and alter the initially given information while common pitfalls are overlooked so that often over-optimistic conclusions are drawn that do not reflect real-world cybersecurity scenarios [5]. In fact, program behaviour and malware families can change rapidly through the continuous use of new malicious code, which ultimately leads to trained ML models becoming outdated after only a few months without being noticed [26]. Moreover, these learning models may also be subject to cyber-attacks aimed at deliberately "confusing" the model and altering prediction results, which in turn may cause financial loss, infrastructure damages, etc. [4]. This situation results in a loss of control over the data, especially knowing that these models are usually implemented by ML experts that are neither the actual cybersecurity operators nor malware analysts [27].

Prior studies have tried to address these concerns [26], [45], for example proposing an approach to evaluate temporal decay on Android malware. However, the authors addressed the problem as a binary classification and they did not analyze the causes that generated the drift. Concept drift is also analyzed in [23] and [30]. In both cases, again, the authors only applied methods to detect concept drift, without analyzing the causes behind this performance decay and its relation with the malware behaviour.

Digital Forensics is a field within forensic science that

is concerned with the identification, acquisition, processing, analysis, and reporting of electronically stored data [8]. Its primary objective is to extract useful information from digital evidence, transform it into actionable intelligence, and present the findings. This strategy is used in a variety of contexts, such as criminal investigations, incident response, cyberattacks investigation, data breaches, etc. [6] In the same way, in this paper, we aimed to use a digital forensic approach to gain more knowledge about the ML models, shedding light on input data evolution and model behaviors. In particular, we propose a methodology for forensics exploration based on temporal analyses and concept drift detection, with the main goal of extracting a deeper understanding of the challenges faced by ML-based classification and exploring possible reasons for model failure. Thanks to this, it will be possible to have a broader vision of the model characteristics without limiting the conclusions to its performance, helping cybersecurity analysts and practitioners. To validate the effectiveness of this approach, we apply it to a specific case study involving a multi-class task with portable executables (PE).

Our methodology is based on three main steps. In Step 1, we analyze how the data temporality in the training dataset affects model performance and leads to concept drift and points of model failure. This step applies a *temporal dissection* approach dividing the given data into short temporal chunks and using a rolling window strategy to train multiple models in order to evaluate how the quality of the information affects model performance over time. Then, in Step 2, we apply a *temporal aggregation* operation for analyzing the temporal consistency and reliability of the implemented models to unveil concept drift and points of model failure. This operation increases the amount of information (the number of temporal chunks) used for training the models. Finally, in Step 3, we carry out an in-depth misclassification and feature analysis combining feature importance score, prediction results, and performance evolution to understand trends and patterns that may have generated the concept drift and model failure.

By applying digital forensic concepts and recently published guidelines [5], it is possible to extract new domain knowledge that could be used to enhance the analyst's expertise and to improve the detection of potential future attacks [12]. We believe that the introduced methodology represents an initial step to highlight behavioral trends and causes of model failure of both models and input data. This information can help analysts create a more valid and transparent ML model during the research phase and then translate their lessons learned to a production environment. In fact, identifying the specific input information that is causing concept drift can allow users and practitioners to acquire additional domain knowledge that, in turn, can be used to take preventive actions, such as modifying the data pre-processing pipeline, adjusting the model's parameters, determining when retrain or update their model, for finally avoiding future model failures.

Inspired by previous approaches in which embeddings were computed from malware Control Flow Graphs (CFGs) [41], [43] in this work, we evaluate our methodology on multi-class classifiers that directly use graph structural properties of CFGs as embeddings for the malware classification. In particular, we use three different classifier families, each one implemented in two configurations. To the best of our knowledge, this is the first study that is not limited to merely analyzing PE malware multi-class classification performance but also proposes to use detailed temporal and misclassification analyses to identify and comprehend concept drift and potential points of failure for digital forensics.

The main contributions of this work can be summarized as follows:

- We compare the performance of six ML models able to classify six malware families (multi-class problem) using structural graph properties obtained from malware CFGs;
- Two temporal analyses are proposed: *temporal dissection* to evaluate the quality of the training data and how it affects model performance over time, and *temporal aggregation* to detect significant concept drift points, even when a larger dataset is used for training the model;
- An extensive misclassification and feature analysis is performed to extract insightful information and explain the causes that may have led to the observed concept drifts and model failures.

The rest of the paper is organized as follows. In Section II, concepts regarding malware classification and concept drift are introduced. In Section III, we present the proposed methodology, whereas in Section IV, the dataset and the evaluation metrics considered, as well as the experiment configurations, are introduced. Then, in Section V, results are presented and discussions are reported in Section VI. Finally, Section VII provides conclusions and guidelines for future work.

II. Background

In this Section, concepts about malware classification and concept drift are introduced. More specifically, Section II-A describes the malware classification task and its related works, whereas in Section II-B the concept drift problem is presented.

A. Malware Classification

Malware classification tasks are usually based on *static*, *dynamic*, or *hybrid* analyses. In the static case, malware binaries are analyzed without actually running the code. The code is used to extract information such as signatures, and hashes, or for creating graph structures on top of which the classification is performed [40]. In dynamic analyses, the malware code is executed in an isolated environment, and its behaviour is studied, as shown in [14]. However, this approach can be more complex and time-consuming due to the required malware execution in a secure environment. Finally, in the hybrid analysis, both static and dynamic information are combined for the final prediction [3]. One of the most promising approaches is based on static analysis, extracting CFGs in order to represent the flow of control between the basic blocks in the program, i.e., a maximal-length sequence of a branch-free code [9]. In these CFGs, vertices represent

sequential code without branches or jump targets, whereas edges represent the jumps in the control flow of the program.

In particular, Xu et al. [42] used CFGs and data flow graphs extracted from Android applications, whereas Bruschi et al. [7] introduced a detection method for detecting malicious code exploiting their representation via CFGs. In [40], graph information extracted from CFGs was combined with an instruction sequence for a malware family classification through graph neural networks. In [43] and [41], authors proposed to use embeddings directly extracted from Malware CFGs using deep learning models. However, in this way, embeddings were computed directly by the learning models, so they worked like "black boxes", which do not allow linking classification performance to the embedding values. In this sense, it was not possible to extract additional insightful information for improving CTI.

Inspired by the approaches presented in [41] and [43], in this work, we directly use structural properties of the CFGs as *graph embeddings* and use them for the final multi-class classification. In this way, our hypothesis is that we can directly relate classification performance to the structural graph embeddings and use the analysis of input trends to improve malware CTI. Furthermore, while [41] focused on firmware images or vulnerable functions for binary classification, we extract CFGs directly from a large PE malware dataset to perform a multi-class classification.

B. Concept Drift

In machine learning problems, results strongly depend on the data used during the training process. Hence, uncontrolled changes in the input features can generate inconsistent and misleading results. This problem becomes even more relevant when temporal data are analyzed and novel information is used as input for models trained on old data [35]. This problem is usually known as *concept drift* or *dataset shift* [39]. In particular, concept drift can appear following different temporal patterns, which can be distinguished into four categories [24]: *abrupt, incremental, gradual* or *reoccuring* drift. It is to be noted that although outliers and noise can be seen as points that instantaneously change the data distribution, as happen in *abrupt* drift, formally, they are not considered as concept drift [24].

Concept drift is a highly relevant problem in cybersecurity applications, especially when machine learning models are implemented. It has been predominantly studied in applications such as anomaly detection [17], fraud [10] and spam [31] detection. However, in recent years, concept drift has been recognized as a key issue for malware detection as malware developers constantly try to create new variants to avoid detection, leading to continuously evolving malware behaviour that may render certain machine learning models outdated after a few months only. More specifically, in [11], authors try to improve classifier performance by detecting concept drift and creating adaptive models, i.e., systems able to detect drift while being in operation ("on-running") and address it by retraining themselves with more recent data. Other studies, such as [23] and [30], propose novel methods for detecting concept drift, however, they do not deeply analyze the relationship between these performance decays and the actual input of the malware classifier.

For these reasons, in this work, we propose to analyze concept drift in depth using digital forensic analysis. Our focus is not on directly mitigating concept drift, but rather on developing a methodology to study it and gain a better understanding of its causes and its impact on model decisions over time.

III. Towards practical malware threat analysis

The methodology proposed in this study aims to conduct a digital forensic analysis that involves temporal and feature analysis. This approach is then validated using a dataset of PE malware binaries. The idea of this strategy is to extract insightful information about the limitations of both the model and the data. More specifically, the methodology is based on three main steps: *Temporal dissection, Temporal aggregation* and *Misclassification and feature analysis*.

1) **Temporal dissection**: this step allows us to evaluate how the *quality* of the data changes over time and how it affects the classification performance. In this sense, we propose to split the dataset into M temporal chunks of fixed size, and then train several ML models using a x-chunks rolling window data and finally evaluate them with data from the following chunks, respectively (Figure 1).
2) **Temporal aggregation**: this step changes the size of the training dataset to incorporate more chunks, as shown in Figure 2. This strategy helps us to detect more consistent drift points in the model performance, i.e., performance drops found when larger datasets are used during the model training. These drift points are further analyzed in the next step.
3) **Misclassification and feature analysis**: in this step, we first analyze confusion matrices to highlight which classes are misclassified. Then, a feature importance analysis is carried out. This operation can be performed using different approaches, such as the mean decrease impurity (MDI) [28], permutation method (PI) [2], SHapley Additive exPlanation (SHAP) [16], etc. This operation helped us in ranking the features and focusing the analysis just on the most relevant ones used by the classifier. Hence, the temporal trends of these features, i.e., the feature values per class in all chunks, can be analyzed. In this way, it is possible to detect rare behaviours in the data that may help to explain what the model learned and why it misclassified certain classes. However, in cases where ML decisions are complex and difficult to interpret, methods such as SHapley Additive exPlanation (SHAP) [16] or Local Interpretable Model-Agnostic Explanations (LIME) [47] can be employed to study the predictions of individual instances or cases (local interpretability) and to attempt to generalize the model's behavior.

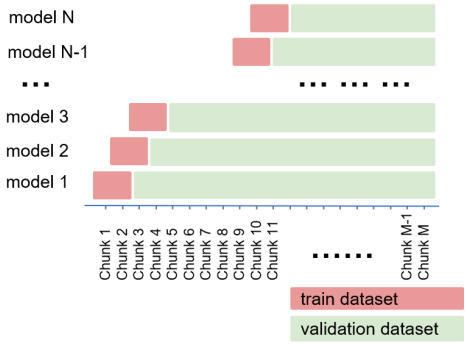


Fig. 1: Temporal dissection with 2-chunks rolling window data.

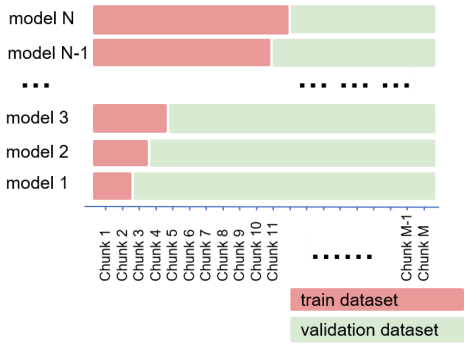


Fig. 2: Temporal aggregation models trained with different data amount.

IV. Experimental Framework

In this Section, we describe the dataset, the used metrics and the experiment configurations used in this work. More specifically, in Section IV-A an overview of the used dataset is reported, whereas in Section IV-B the metrics are described. Finally, in Section IV-C, we present the classifier and the parameters used during the experiments.

A. Dataset

In this work, as a case study for validating the introduced approach, a PE Malware dataset called SOREL-20M [20] is used. This dataset contains information about 20 million binaries, of which 10 million are benign and 10 million malicious, collected from January 1, 2017 to April 10, 2019. This dataset publishes the executable files of the malware only, which are opportunely disarmed. For each of these malware programs, a first-seen timestamp is provided as well as one or more "tags". These tags represent the number of times in which a certain sample has been detected as belonging to a concrete malware family. If malware belongs to more than one family, it has multiple tags, creating a multi-label dataset. The dataset contains 11 malware families: *Adware, Dropper, Spyware, File Infector, Worm, Downloader, Flooder, Ransomware, Packed, Cryptominer* and *Installer*.

For the aim of this work, we focus only on the most represented classes which are Adware, Dropper, Spyware, Packed, File Infector, Worm, and Downloader. Furthermore, from this list, we excluded the Packed family as well, since the compression of the malicious code works as an obfuscation technique that obstacles their detection [44]. Hence, their identification is out of the scope of this work. This leaves us with 6 families to be distinguished. As mentioned, malware samples in the SOREL-20M dataset can have more than one tag (multi-label problem), however, for the sake of simplicity, we focus on the ones that are characterized by one tag only, addressing the problem as a multi-class task. In order to decrease the computational cost mainly related to CFG extraction and avoid imbalance problems, $9,000$ samples are chosen among the obtained multi-class data to create the dataset for the experiments. The final dataset was composed of 54,000 single-labelled malware samples (9,000 per class) as shown in Table I, whereas Figure 3 shows the monthly distribution of the malware families in the 28 months (from 01/2017 until 04/2019).

	Short	SOREL multi-labelled (M)	Single-labelled class reduced	Final dataset extracted
Months from		01/2017	01/2017	01/2017
Months to		04/2019	04/2019	04/2019
Adware	Adw	$\sim$ 2.4	306,972 (27%)	9,000 ($\sim$ 16.67%)
Dropper	Dro	$\sim$ 3.5	210,833 (18%)	9,000 ($\sim$ 16.67%)
Spyware	Spy	$\sim$ 4.5	93,305 (8%)	9,000 ($\sim$ 16.67%)
File Infector	Fil	$\sim$ 3.3	193,932 (17%)	9,000 ($\sim$ 16.67%)
Worm	Wor	$\sim$ 3.4	288,417 (25%)	9,000 ($\sim$ 16.67%)
Downloader	Dow	$\sim$ 2.5	51,037 (5%)	9,000 ($\sim$ 16.67%)
Flooder	-	$\sim$ 0.1	-	-
Ransomware	-	$\sim$ 1.1	-	-
Packed	-	$\sim$ 3.7	-	-
Cryptominer	-	$\sim$ 0.3	-	-
Installer	-	$\sim$ 1	-	-
Total		-	1,144,496 (100%)	54,000 (100%)

TABLE I: Malware families distribution in the dataset.

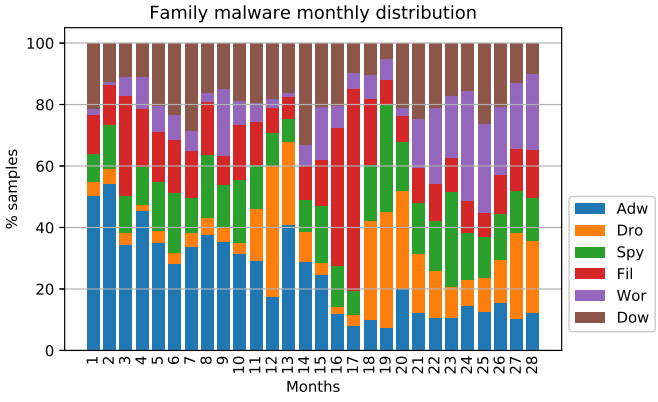


Fig. 3: Malware families in each month (in percentage).

B. Metrics.

F1-score and *Area Under Time (AUT)* were used to compare the performance of the different models. Hereby, *F1-score* represents the relation between actual positive labels and those given by the classifier, whereas *AUT* indicates the area under the performance curve over time [26]. In our experiments, the AUT is computed using the F1-score monthly curve. Furthermore, two AUT variants are considered: AUT_{next6} and AUT_{last6}.

Model family	Name	Parameters
Random Forest	RF1	estimators = 100, max depth = 10
	RF2	estimators = 100, max depth = 100
Adaptive Boosting	ADA1	estimators = 100, max depth = 10
	ADA2	estimators = 100, max depth = 100
Support Vector Machine	SVM1	C = 1, gamma = 0.5
	SVM2	C = 1, gamma = 0.1

TABLE II: Classifiers families and configuration used in the experiments.

The first one is computed considering only the 6 months following the ones used for training the model, whereas the latter is computed using only the last 6 months of the dataset. The AUT_{last6} can be used for evaluating the performance of the models over a common dataset. All considered metrics were applied in multi-class classification using micro-average operations, which do not take into consideration possible differences between classes [18]. All metrics take values in a range between 0 and 1.

C. Experiment Configurations

In this work, we apply our three-step methodology to classifiers able to analyze CFG properties extracted from PE malware binaries. In particular, we extract CFGs from SOREL dataset by using the Angr tool[1] [29], [33]. Then, from each malware CFG, we extracted 9 common graph properties to describe them: *the number of nodes in the CFG; the number of edges; the number of strongly connected components [25]; the number of weakly connected components [37]; the number of isolated nodes; transitivity of the CFG [21]; maximum centrality degree, minimum centrality degree, centrality degree on average.* These CFG properties allow us to transform the graph information into feature vectors (embedding) that are finally used for training ML classifiers.

Three model families are used to learn and classify malware behaviours extracted from CFGs. More specifically, Random Forest (RF), Adaptive Boosting (ADA) [15] and, Support Vector Machine (SVM) [22] are implemented, each one in two different configurations, as reported in Table II.

For validating the introduced approach, the malware dataset is split into chunks of 1 month, generating 28 chunks (M). In particular, during the temporal dissection (Step 1), a rolling window is used to keep selecting 3 consecutive months (x) to be used for training the classifiers, which are then evaluated with data from the following months - while moving forward in time, as described in Section III. The process is repeated by moving the rolling window until reaching month 12 (12-2017), hence, generating 10 different models (N). On the other hand, during the temporal aggregation (Step 2), 3 temporal sizes are used, i.e., 3 models for each family-configuration are trained. In particular, train datasets from the first month until months 3, 6 and 12 are used. This approach simulates training with as much data as one has at hand in an equivalent real-world scenario, and it helps to detect consistent performance drifts.

[1]https://docs.angr.io/built-in-analyses/cfg

Finally, due to the ML models trained in this case study, the Permutation strategy is used for computing the feature importance (*Misclassification and feature analysis* step). This strategy measures the decrease in model performance after the feature's values are shuffled [2]. Then, just the top-4 features are further analyzed in terms of temporal (monthly) trends to detect rare behaviours in the data. Lastly, to improve the explainability of the results of unclear decay points, the local interpretability of the model is studied using the SHAP strategy [16]. The local interpretability allows us to detect how individual features of a concrete input sample contribute to the final prediction.

V. Experimental Results

In this section, the results obtained by applying our methodology to the introduced malware dataset, are reported. In particular, the temporal dissection (Step 1) and temporal aggregation (Step 2) results are described in Section V-A and Section V-B, respectively. Finally, in Section V-C, misclassification and feature analysis results (Step 3) are presented.

A. Temporal dissection.

Figure 4 reports the monthly F1-scores of models trained with 3-month rolling window data. The figures show that all the models (of the three families) perform well in the initial months after training. However, their performance dramatically deteriorates over time. More specifically, models trained with the early months (from months 1 to 5) exhibit poor results in terms of F1-score in almost every test month. This behaviour is more evident in SVM models (Figure 4c and Figure 4f). When month 5 is included in the training dataset (3-5 and 4-6), SVM models still yield poor results. In contrast, RF and ADA models perform well in the 3-4 following months. They show a strong decay in months 11, 12, and 13 and then again, an improvement until month 17 (their peak value). However, all these models show their lowest performance in month 19. ADA models trained in months 5-7 and 6-8 (Figure 4b and Figure 4e) exhibit promising results with high performance in the following 9-10 months. Yet, with a strong decay (lowest point) on month 19. Finally, all the models (of the three families) trained from month 7-9 show high results just for the first 3-4 months.

B. Temporal aggregation

AUT, AUT_{next6} and AUT_{last6} of the models trained with temporal aggregation strategy are reported in Table III. The 3-month models achieve very low values, especially in terms of overall AUT (values ≤ 0.43). However, RF2, ADA1 and ADA2 models show better performance in the 6 months following the training dataset, reaching $AUT_{next6} \geq 0.60$. Nonetheless, their performance decreases dramatically when "recent" data are used (AUT_{last6}), consistent with the results obtained in the temporal dissection analysis. The same trend is shown for 6-month and 12-month models. However, in the latter, RF2 and ADA2 models reach overall AUT ≥ 0.62 and $AUT_{next6} \geq 0.77$.

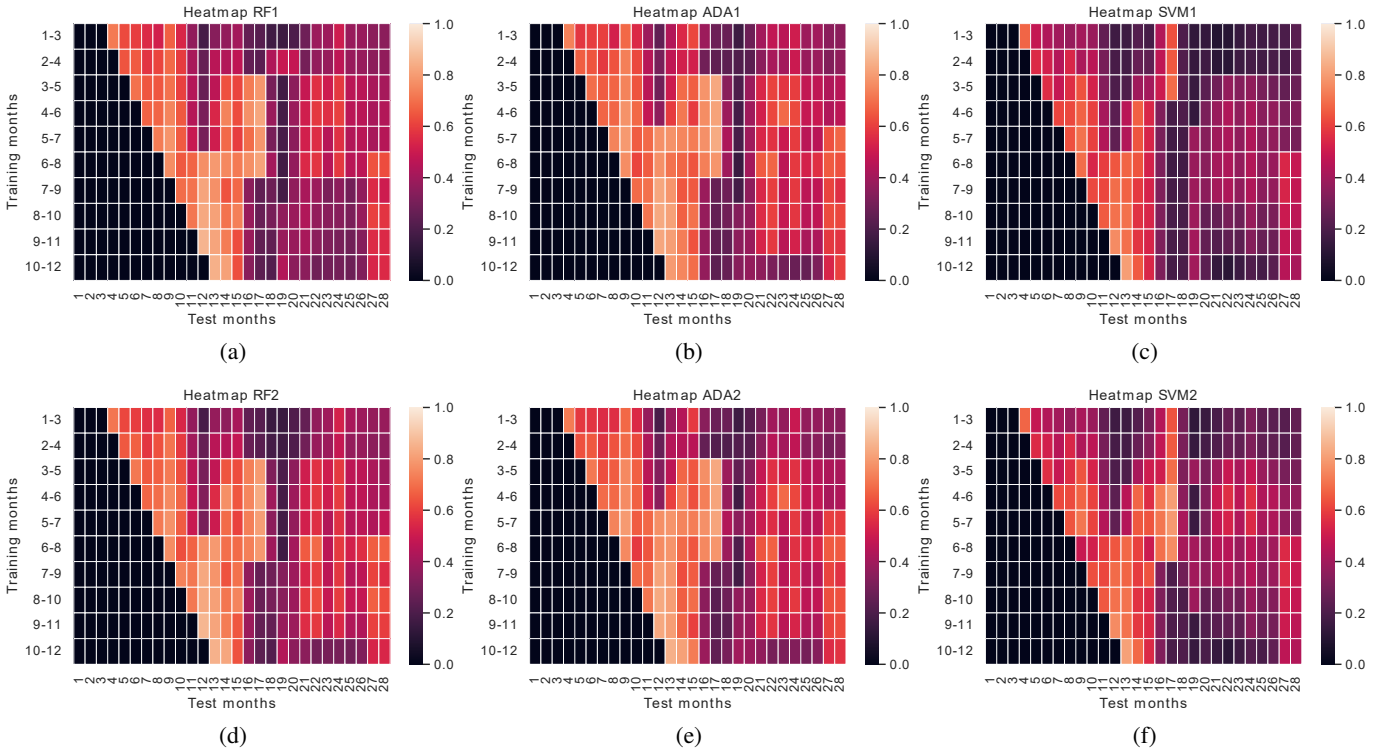


Fig. 4: Monthly F1-score values for each model trained with 3-month rolling window data (temporal dissection).

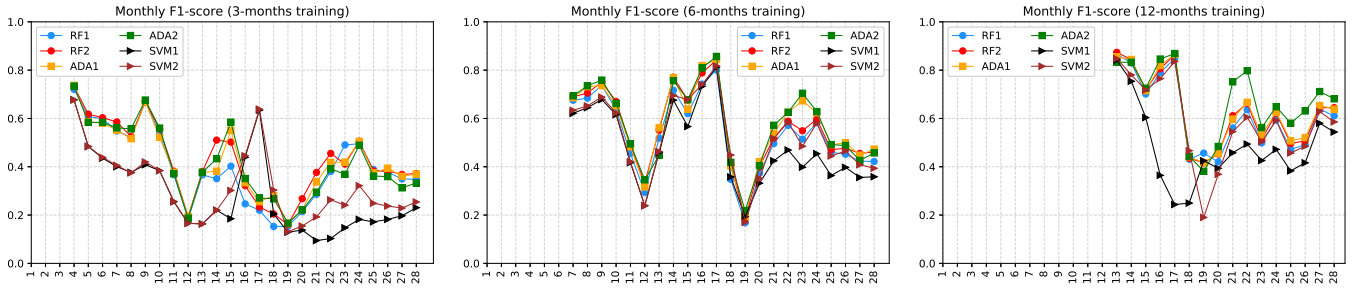
Model name	Training months	AUT	AUT_{next6}	AUT_{last10}
RF1	1 - 3	0.39	0.59	0.40
RF2	1 - 3	0.43	0.61	0.41
ADA1	1 - 3	0.41	0.60	0.38
ADA2	1 - 3	0.41	0.60	0.40
SVM1	1 - 3	0.27	0.45	0.18
SVM2	1 - 3	0.31	0.45	0.26
RF1	1 - 6	0.53	0.60	0.48
RF2	1 - 6	0.56	0.63	0.50
ADA1	1 - 6	0.57	0.62	0.53
ADA2	1 - 6	0.58	0.63	0.52
SVM1	1 - 6	0.48	0.56	0.39
SVM2	1 - 6	0.52	0.56	0.47
RF1	1 - 12	0.61	0.76	0.55
RF2	1 - 12	0.62	0.77	0.58
ADA1	1 - 12	0.61	0.75	0.57
ADA2	1 - 12	0.63	0.80	0.57
SVM1	1 - 12	0.46	0.50	0.47
SVM2	1 - 12	0.58	0.75	0.54

TABLE III: Overall AUT, AUT_{next6} and AUT_{last6} values for models trained on differently sized temporal datasets (temporal aggregation).

Figure 5 shows the monthly F1-scores for each trained model, revealing a performance variance according to the training dataset size. In particular, the trend showed by all 3-month models (Figure 5a) can be seen as a *incremental* concept drift, with a slow decline and two significant drops at months 12 and 19. SVM1 model has two more decay points in months 21 and 22. Models trained with 6-month (Figure 5b) do not show a clear concept drift pattern, but they disply fragmented values that in some cases can be considered both an *abrupt* concept drift or *outlier*. All 6-month models showed similar performance until month 20, when their monthly performance started to diverge. Yet, all of them exhibit a performance drop in months 12 and 19, with their best value in month 17. Finally, analyzing Figure 5c, again all the models (excluding the SVM1) showed high performance in month 17 and a decay in month 19.

The results obtained from the temporal dissection and the temporal aggregation analysis, have highlighted the presence of three interesting points, two related to performance decay (months 12 and 19) and one related to highest performance (month 17). Our hypotheses, for each point, are the following:

1) *Hypothesis A - month 12:* one possible explanation of this decay is that there was a change in the distribution of the Dropper family (Figure 3), as evidenced by the higher presence of Dropper samples in this month compared to the previous ones that were used for training the models.
2) *Hypothesis B - month 17:* the performance peak in this month could be attributed to the fact that models learn to perfectly distinguish all the classes, especially the File Infector which is the most represented one in this test month (Figure 3).
3) *Hypothesis C - month 19:* this second decay could be due to problems in distinguishing Dropper and Spyware samples which are the most represented classes in this test month (Figure 3), despite the models are trained



(a) 3-month training dataset. (b) 6-month training dataset. (c) 12-month training dataset.

Fig. 5: Monthly F1-score computed for models trained on differently sized temporal datasets (Step 2 - temporal aggregation)

with larger datasets that included both families.

In order to validate our hypotheses, we propose to use the misclassification and feature analysis (Step 3) to investigate these points in depth. For the sake of simplicity, the characteristics of the best model (ADA2) are further analyzed.

C. Misclassification and feature analysis

Figure 6 shows the confusion matrices of the 3-month and 6-month ADA2 models when test data on month 12 are used. In particular, the 3-month model confuses 95% (755/793) of Dropper for Adware samples, whereas the 6-month model confuses 94% (747/793) of Dropper for Downloader ones. Furthermore, from this analysis, it is possible to highlight that the 3-month model (Figure 6a) has a problem also in detecting 65% (216/336) of Downloader, which are classified as Adware. This insight seems to confirm our *Hypothesis A*. In fact, in both cases, the models have problems in detecting Dropper samples which are the most represented in this test month, affecting the final results.

Analyzing Figure 7, it is possible to see that all the models that showed their best performance on month 17 (6-month and 12-month ADA2 models), can correctly classify Adware, Spyware, Worm, Downloader and File Infector. More specifically, 98% of File Infector samples are correctly classified, affecting the final performance. On the other hand, the model 3-month ADA2 is not able to distinguish File Infector samples as the other models, and in fact, 88% of them are classified as Dropper. This analysis seems to verify the correctness of our *Hypothesis B*.

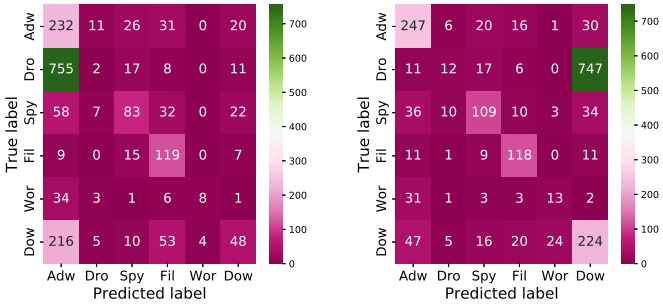
Finally, Figure 8 reports confusion matrices of the models affected by performance decay on month 19. Figure 8a shows that the 3-month ADA2 model is able to distinguish Adware, File Infector, and Downloader samples. However, its main problem is in the detection of Dropper and Spyware families. In fact, about 94% (1,986/2,121) of Dropper samples are classified as Spyware, whereas about 66% (1,293/1,973) of Spyware are classified as Dropper. These two families are also misclassified by the 6-month model (Figure 8b), however in this case, although the Dropper family is again confused with the Spyware with a 94%, the model mixes Spyware with Worm (60%). Finally, analyzing the confusion matrix of the 12-month model (Figure 8c), it is possible to see that this model learns to classify the Spyware family correctly. Yet, it still shows problems with Dropper samples, where 94% of them are classified as Spyware. This analysis seems to verify also our *Hypothesis C*. However, we are interested in extracting information about the possible causes of this misclassification. For this reason, feature importance and feature trend analyses are reported.

	Permutation Importance on ADA2 models			
Features	**3-month**	**6-month**	**12-month**	**AVG.**
# nodes	0.09	0.10	0.17	0.12
# edges	0.18	0.24	**0.25**	0.22
# weakly conn.	**0.34**	**0.26**	**0.25**	**0.28**
# strongly conn.	**0.31**	**0.25**	**0.27**	**0.28**
# isolated nodes	0.23	0.20	0.02	0.15
transitivity	0.00	0.01	0.01	0.01
# max. centrality degree	**0.31**	**0.35**	0.14	**0.27**
# min. centrality degree	**0.24**	**0.34**	**0.34**	**0.31**
# avg. centrality degree	0.18	0.14	0.08	0.13

TABLE IV: Feature importance scores based on Permutation Importance. The top-4 features for each model are highlighted.

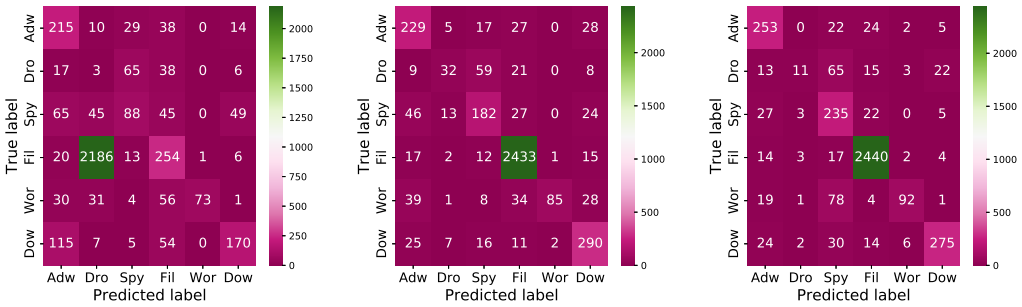
In Section IV-C, Performance Importance (PI) values were calculated for each ADA2 model, and the results have been presented in Table IV. The analysis shows that for all models, *number of weakly connected components, number of strongly connected components* and *minimum centrality degree* are essential features for the final classification. However, 3-month and 6-month models also consider *maximun centrality degree* in their top-4 important features, whereas the 12-month model looks for *the number of edges*. Following the averaged PI values over all the models (Table IV), the temporal trends of *number of weakly connected components, number of strongly connected components, minimum* and *minimum centrality degree* are further analyzed in this study. The trend of a feature is computed by averaging its values for each malware family on a monthly basis, as shown in Figure 9. Then, the top 4 feature trends are compared in order to detect rare behaviours in the data that may help to explain what the models learnt and why they misclassified certain classes. In particular, just the trends of Dropper, Spyware and Worm samples are analyzed since these classes are the ones misclassified, as shown in the previous analysis (Figure 8).

Figure 9 shows that the feature trends of Dropper, Spyware



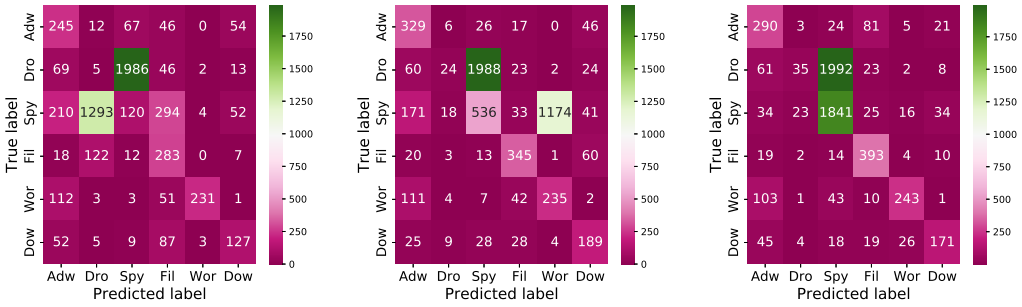
(a) 3-month ADA2 (b) 6-month ADA2

Fig. 6: Confusion matrices of the ADA2 models computed using test data on month 12 (first drop).



(a) 3-month ADA2 (b) 6-month ADA2 (c) 12-month ADA2

Fig. 7: Confusion matrices of the ADA2 models computed using test data on month 17 (first hype).



(a) 3-month ADA2 (b) 6-month ADA2 (c) 12-month ADA2

Fig. 8: Confusion matrices of the ADA2 models computed using test data on month 19 (second drop).

and Worm change constantly. In particular, when a model is trained in just the first 3 months, it learns behaviours that are very different from the ones that occur in month 19. In fact, in terms of weakly connected components (Figure 9a), Dropper samples showed very low values in the first 3 months (from 10 to 38), whereas the Spyware reached high values (from 40 to 85). However, in month 19, it is the Dropper family that shows high values rather than Spyware. This trend occurs also remarked in the strongly connected components feature (Figure 9b), while in minimum and maximum centrality degrees (Figure 9c and Figure 9d) is the Spyware that started with low values for then overwhelming Dropper ones in month 19.

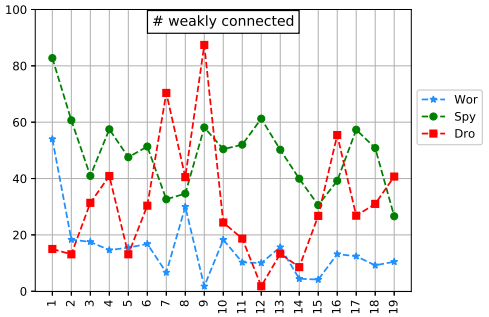
Taking into account 6 months for training the model, still does not address the misclassification of Dropper and Spyware. However, in this case, the new 3 months integrate more information that misleads the model in classifying Spyware as a Worm family. In fact, in these new 3 months of the datasets, the values of the strongly connected components (Figure 9b) of Worms and Spyware are aligned, with the first always below the latter. However, in month 19, again the Spyware samples show low values never reached before. A similar behaviour can be observed in minimum and maximum

centrality degrees (Figure 9c and Figure 9d), where in month 19, Spyware reached high values usually reached by Worm samples (at least in the 3 new months added in the training dataset).

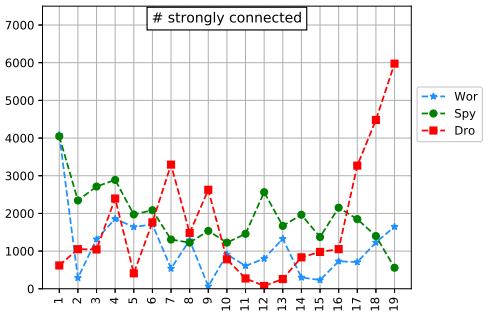
Finally, it is not possible to extract a clear explanation about causes that generate misclassification when 12 months are used as training dataset. In fact, in this case, limiting the trend study only to the top-4 features is not enough for a comprehensive analysis. For this reason, in order to extract information related to this case, in Figure 10, the local explainability results computed using the SHAP strategy are reported. In particular, the figures show how each feature contributes to the classification of a specific input sample. More specifically, two samples of the misclassified class (X_1 and X_2 Dropper) are randomly chosen from the month 19 test dataset. Figure 10a shows that, although X_1 is a Dropper sample, the values of almost all the features lead the model to classify it as Spyware with a score of 98%. In particular, in this case, the value 0 of the minimum degree centrality, a single isolated component and average degree centrality equal to 0 are the most important features. On the other hand, as shown in Figure 10b, X_1 is classified as Dropper just with a score of 1%. In fact, only the huge amount of nodes, and the number of strongly and weakly connected components seem favoring the Dropper label. A similar trend is also shown in Figure 10c and Figure 10d, when X_2 is used as input to the model. In this case, again, the sample is classified as Spyware with a score of 42%, thanks to the 0 in the average centrality degree, the low number of weakly connected components and the maximum centrality. This latter also positively impacts the Dropper classification (Figure 10d). However, the number of edges, the weakly connected, and the number of isolated components decrease the probability of the Dropper label (12%). With this analysis, it is possible to see, that values of weakly connected components around 40 favouring the Spyware classification over the Dropper one, and this is exactly the average value that the Dropper family shows in month 19 (Figure 9a). Furthermore, Figure 10 shows that also other features like the number of edges and average centrality degree are determinants for penalizing Dropper classification.

VI. Discussion

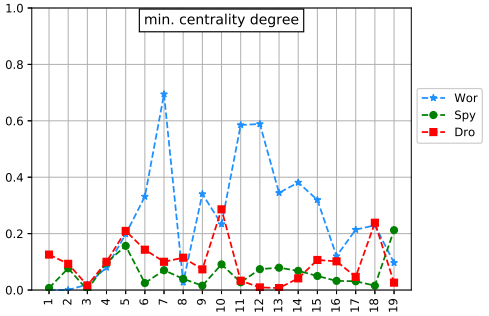
The temporality of the data and hence the lifetime of malware strongly affected model performance. Analyzing the performance of models trained with the presented rolling window technique, one can appreciate that all models worked better (and surprisingly well) in months closer to the months used for training them, whereas their performance decayed dramatically when evaluating with far away temporal data, i.e., malware developed about 4 months later. This is an important finding as it warrants caution when applying once-trained models in real-world applications: especially when dealing with malware, models do have a "lifetime" and should be re-trained after defined intervals, i.e., 4 months in our case study. On the other hand, *temporal aggregation* strategy showed that the more data were used in the training dataset, the better



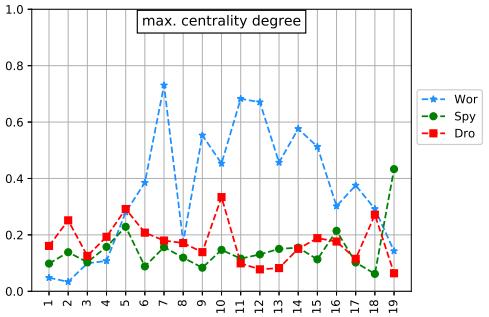
(a) Weakly connected components



(b) Strongly connected components

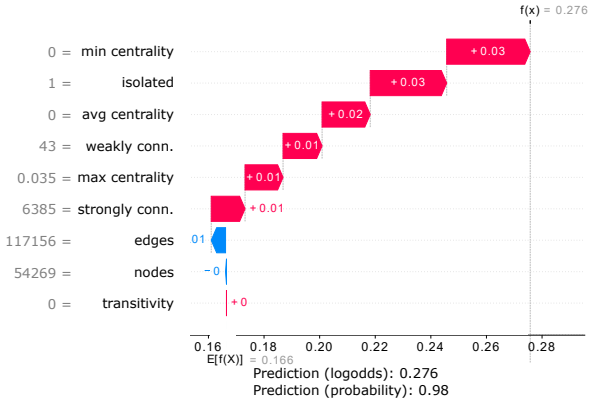


(c) Minimum centrality degree

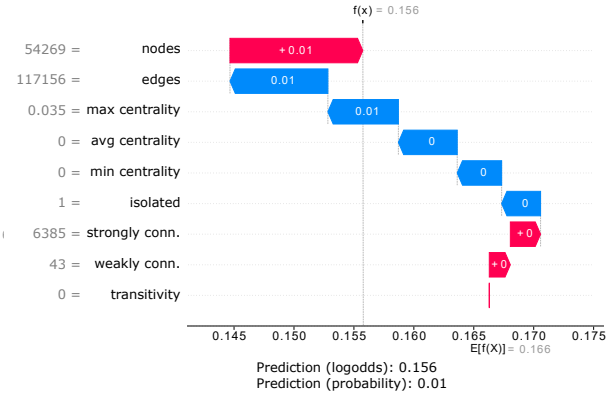


(d) Maximum centrality degree

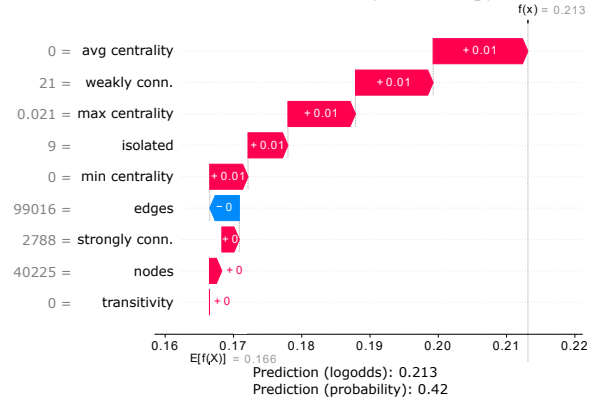
Fig. 9: Trends of the top-4 features for the Dropper, Worm and Spyware classes until month 19.



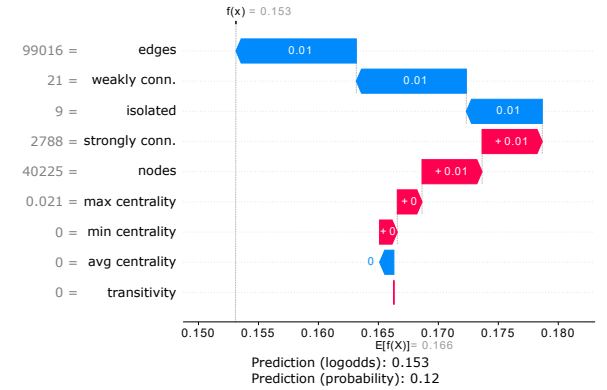
(a) Feature contribution to classify X_1 as Spyware



(b) Feature contribution to classify X_1 as Dropper



(c) Feature contribution to classify X_2 as Spyware



(d) Feature contribution to classify X_2 as Dropper

Fig. 10: Local explainability with SHAP: how features contribute to the classification of X_1 and X_2 (Dropper samples classified as Spyware) in the 12-month model.

the overall classifier performance (Table III). Furthermore, temporal aggregation analysis showed that even using a larger dataset in the training process, all the considered models were affected by concept drift - with the last models, trained with 12-month (Figure 5c), showing the least concept drift as one would expect. In particular, 3-month models (Figure 5a) showed several clear points of decline after which their monthly performance tended to decrease. Finally, the 6-month models (Figure 5b) did not show a clear concept drift pattern but more fragmented values which can be considered both abrupt concept drift or outliers.

In the final misclassification and feature analysis step, we analyzed the confusion matrices extracted for the dates of critical model failure (months 12 and 19) and for the date where the models showed their best values (month 17). We performed a feature importance analysis in order to explore features and their trends in more detail. Table IV shows that all structural graph properties extracted from the CFG were useful for the final classification, except for "transitivity", which can be removed in future applications to improve generalization and computational efficiency. When analyzing the trends of the 4 most important features (Figure 9), we observed that at least one feature usually changed substantially in the months affected by performance decay. At the same time, other features seemed to align with features from the classes that the initial class was confused with. This combination of divergence and alignment of such features, together with the analysis of the composition of the training dataset, allows us to shed light on possible causes of the misclassifications.

Limitations and common pitfalls. The proposed methodology has been applied to a subset of the SOREL-20M dataset for multi-class PE classification. However, a significant portion of the dataset was filtered out to remove (a) less-represented classes and (b) samples with multiple labels. For this reason, the obtained results and conclusions represent just a preliminary analysis and should be verified also when both conditions, (a) and (b) would be relaxed, as happen in the real environment. Furthermore, although we relate CFG properties with performance drifts over time, these features do not really allow us to determine what is changed in the malware executable. However, these tasks are out of the scope of this work, which aim was to validate the whole methodology for discovering insights and improving explainability on model failure points.

To evaluate the quality of the proposed methodology, we discuss possible pitfalls that can generate misinterpretation of preliminary results. These pitfalls are analyzed following the same notation as introduced in [5]. *P1. Samples Bias* and *P2.*

Label Inaccuracy represent the hardest pitfalls to mitigate. In fact, although a single dataset (SOREL-20M) was used for our analysis, it was composed of information from multiple sources (P1), and it was not possible to verify the correctness of all the labels (P2). Therefore, the limitations of the dataset were acknowledged (P1), and the analysis focused only on samples with a single label (P2).

P3. Data Snooping, *P4. Spurious Correlations* and *P5. Biased Parameter Selection* represents three pitfalls related to the system design. P3 and P5 were mitigated by splitting the dataset into train and temporal test sets for validating the performance. On the other hand, regarding P4, we provided an analysis of feature importance to highlight which features were more relevant for the classification, becoming more prone to be used in an attack.

P6. Inappropriate Baseline, *P7. Inappropriate Performance Measures* and *P8. Base Rate Fallacy* are common pitfalls related to the evaluation of performance. In our approach, the models themselves evaluated over time were used as the baseline models (P6). However, following suggestions given for P7, appropriate performance measures were used, such as AUT, and F1-score. Regarding P8, the overall dataset result was balanced, since we selected the same amount of samples for each class. However, classes were not balanced temporally due to their distribution in the original dataset.

Finally, the last two pitfalls - *P9. Lab-Only Evaluation* and *P10. Inappropriate Threat Model.* – are related to the deployment in real scenarios. In this work, we dealt with these two pitfalls in trying to replicate real-world conditions, such as considering data availability in time, the evolution of behaviors, the amount of training information (which can affect the model usability), etc.

VII. Conclusions and Future work

In this work, we aimed to highlight the problems that may arise when applying ML-based multi-class classification to real-world malware data. For this reason, a three-step approach was introduced to carry out a forensics exploration of model failures in malware classification. Firstly, we performed a detailed temporal analysis of concept drifts, in which models were trained a) using data from 3-month rolling windows (temporal dissection) and b) by gradually increasing the amount of data available for training (temporal aggregation). Temporal analyses revealed that all models were affected by concept drift (performance breakdown) at the same temporal points, even when trained with a large amount of data. In particular, Dropper and Spyware samples were observed to be among the most misclassified classes. The outcomes of this study can be summarized as follows:

- The temporality of data and the potential lifetime of malware are critical factors when evaluating the performance of classification models. In fact, concept drift and several distinct points of model failure could be observed even using models trained on a relatively large amount of data;
- Our *temporal dissection* approach highlighted that several models trained with only a few months of recent data (3 months) performed surprisingly well on immediately following data;
- Analyzing trends of the most important features over time for confused classes, we observed that in the lead-up to critical points of model failure, it was usually due to more than one feature changing substantially and rather abruptly. Furthermore, in unclear points, local explainability techniques are the key to gaining insights from model failure.

Our final conclusion is that malware features as captured by structural graph properties from CFGs are constantly changing and evolving over time, leading to significant concept drifts and points of classifier failure (in our particular study case for Dropper and Spyware classes). Therefore, training a model and leaving it in operation without re-training presents a potentially serious security risk for practitioners and stakeholders. Following our results, it may be beneficial to train models on fewer but recent data, apply them for a few months only and then re-train using the newest data again (rolling window approach) - even though further research is necessary to confirm this hypothesis and to define re-train intervals better.

In further future work, it may be interesting to validate our methodology in a more complex scenario exploring additional feature sets that could directly reflect malware behaviour changes (not only related to graph properties), problem settings (e.g., introducing other types of malware, consider execution/dynamic traces, etc.), and larger datasets. All models implemented here showed concept drift, and most showed distinct points of model failure. This hints that the problem of malware multi-class classification indeed is not resolved yet and that other strategies, such as adaptive models, need to be explored to mitigate concept drift promptly.

Acknowledgement

This work has been partially supported by the European Union's Horizon 2020 Research and Innovation Programme under the project STARLIGHT (Grant Agreement No. 101021797). Furthermore, the authors would like to thank Prof. Dr. Lorenzo Cavallaro from the University College of London (UCL) for his guidance throughout the research process and his help in designing this study.

References

[1] Av-test. 2022. malware statistics and trends report by av-test. retrieved january 16, 2022 from.

[2] André Altmann, Laura Toloşi, Oliver Sander, and Thomas Lengauer. Permutation importance: a corrected feature importance measure. *Bioinformatics*, 26(10):1340–1347, 2010.

[3] Blake Anderson, Curtis Storlie, and Terran Lane. Improving malware classification: bridging the static/dynamic gap. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, pages 3–14, 2012.

[4] Eirini Anthi, Lowri Williams, Matilda Rhode, Pete Burnap, and Adam Wedgbury. Adversarial attacks on machine learning cybersecurity defences in industrial control systems. *Journal of Information Security and Applications*, 58:102717, 2021.

[5] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. Dos and don'ts of machine learning in computer security. In *Proc. of the USENIX Security Symposium*, 2022.

[6] Humaira Arshad, Aman Bin Jantan, and Oludare Isaac Abiodun. Digital forensics: review of issues in scientific validation of digital evidence. *Journal of Information Processing Systems*, 14(2):346–376, 2018.

[7] Danilo Bruschi, Lorenzo Martignoni, and Mattia Monga. Detecting self-mutating malware using control-flow graph matching. In *International conference on detection of intrusions and malware, and vulnerability assessment*, pages 129–143. Springer, 2006.

[8] Fran Casino, Thomas K Dasaklis, Georgios P Spathoulas, Marios Anagnostopoulos, Amrita Ghosal, Istvan Borocz, Agusti Solanas, Mauro Conti, and Constantinos Patsakis. Research trends, challenges, and emerging topics in digital forensics: A review of reviews. *IEEE Access*, 10:25464–25493, 2022.

[9] Keith Cooper and Linda Torczon. *Engineering a compiler*. Elsevier, 2011.

[10] Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. Credit card fraud detection and concept-drift adaptation with delayed supervised information. In *2015 international joint conference on Neural networks (IJCNN)*, pages 1–8. IEEE, 2015.

[11] Abdulbasit A Darem, Fuad A Ghaleb, Asma A Al-Hashmi, Jemal H Abawajy, Sultan M Alanazi, and Afrah Y Al-Rezami. An adaptive behavioral-based incremental batch learning malware variants detection model using concept drift detection and sequential deep learning. *IEEE Access*, 9:97180–97196, 2021.

[12] Ali Dehghantanha, Mauro Conti, Tooska Dargahi, et al. *Cyber threat intelligence*. Springer, 2018.

[13] Jérémy Donadio, Guillaume Guerard, and Soufian Ben Amor. Collection of the main anti-virus detection and bypass techniques. In *International Conference on Network and System Security*, pages 222–237. Springer, 2021.

[14] Ying Fang, Bo Yu, Yong Tang, Liu Liu, Zexin Lu, Yi Wang, and Qiang Yang. A new malware classification approach based on malware dynamic analysis. In *Australasian Conference on Information Security and Privacy*, pages 173–189. Springer, 2017.

[15] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[16] Daniel Fryer, Inga Strümke, and Hien Nguyen. Shapley values for feature selection: The good, the bad, and the axioms. *IEEE Access*, 9:144352–144360, 2021.

[17] Heitor M Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9):1469–1495, 2017.

[18] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for multi-class classification: an overview. *arXiv preprint arXiv:2008.05756*, 2020.

[19] Steven Strandlund Hansen, Thor Mark Tampus Larsen, Matija Stevanovic, and Jens Myrup Pedersen. An approach for detection and family classification of malware based on behavioral analysis. In *2016 International conference on computing, networking and communications (ICNC)*, pages 1–5. IEEE, 2016.

[20] Richard Harang and Ethan M. Rudd. Sorel-20m: A large scale benchmark dataset for malicious pe detection, 2020.

[21] Paul W Holland and Samuel Leinhardt. Transitivity in structural models of small groups. *Comparative group studies*, 2(2):107–124, 1971.

[22] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification, 2003.

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

[24] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, Joao Gama, and Guangquan Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2018.

[25] Esko Nuutila and Eljas Soisalon-Soininen. On finding the strongly connected components in a directed graph. *Information processing letters*, 49(1):9–14, 1994.

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

[27] Sherif Saad, William Briguglio, and Haytham Elmiligi. The curious case of machine learning in malware detection. *arXiv preprint arXiv:1905.07573*, 2019.

[28] Erwan Scornet. Trees, forests, and impurity-based variable importance. *arXiv preprint arXiv:2001.04295*, 2020.

[29] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Audrey Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. Sok: (state of) the art of war: Offensive techniques in binary analysis. 2016.

[30] Anshuman Singh, Andrew Walenstein, and Arun Lakhotia. Tracking concept drift in malware families. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*, pages 81–92, 2012.

[31] Ge Song, Yunming Ye, Haijun Zhang, Xiaofei Xu, Raymond YK Lau, and Feng Liu. Dynamic clustering forest: an ensemble framework to efficiently classify textual data stream with concept drift. *Information Sciences*, 357:125–143, 2016.

[32] Alireza Souri and Rahil Hosseini. A state-of-the-art survey of malware detection approaches using data mining techniques. *Human-centric Computing and Information Sciences*, 8(1):1–22, 2018.

[33] Nick Stephens, John Grosen, Christopher Salls, Audrey Dutcher, Ruoyu Wang, Jacopo Corbetta, Yan Shoshitaishvili, Christopher Kruegel, and Giovanni Vigna. Driller: Augmenting fuzzing through selective symbolic execution. 2016.

[34] Adem Tekerek and Muhammed Mutlu Yapici. A novel malware classification and augmentation model based on convolutional neural network. *Computers & Security*, 112:102515, 2022.

[35] Alexey Tsymbal. The problem of concept drift: definitions and related work. *Computer Science Department, Trinity College Dublin*, 106(2):58, 2004.

[36] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. Survey of machine learning techniques for malware analysis. *Computers & Security*, 81:123–147, 2019.

[37] JJP Veerman and Ewan Kummel. Diffusion and consensus on weakly connected directed graphs. *Linear Algebra and its Applications*, 578:184–206, 2019.

[38] Aohui Wang, Ruigang Liang, Xiaokang Liu, Yingjun Zhang, Kai Chen, and Jin Li. An inside look at iot malware. In *International Conference on Industrial IoT Technologies and Applications*, pages 176–186. Springer, 2017.

[39] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, 1996.

[40] Rongze Xia and Baojiang Cui. Malware classification based on graph neural network using control flow graph. In *International Conference on Broadband and Wireless Computing, Communication and Applications*, pages 129–138. Springer, 2021.

[41] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 363–376, 2017.

[42] Zhiwu Xu, Kerong Ren, Shengchao Qin, and Florin Craciun. Cdgdroid: Android malware detection based on deep learning using cfg and dfg. In *International Conference on Formal Engineering Methods*, pages 177–193. Springer, 2018.

[43] Jiaqi Yan, Guanhua Yan, and Dong Jin. Classifying malware represented as control flow graphs using deep graph convolutional neural network. In *2019 49th annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, pages 52–63. IEEE, 2019.

[44] Wei Yan, Zheng Zhang, and Nirwan Ansari. Revealing packed malware. *ieee seCurity & PrivaCy*, 6(5):65–69, 2008.

[45] Gaofeng Zhang, Yu Li, Xudan Bao, Chinmay Chakarborty, Joel JPC Rodrigues, Liping Zheng, Xuyun Zhang, Lianyong Qi, and Mohammad R Khosravi. Tsdroid: A novel android malware detection framework based on temporal & spatial metrics in iomt. *ACM Transactions on Sensor Networks (TOSN)*, 2022.

[46] Qinghua Zhang and Douglas S Reeves. Metaaware: Identifying metamorphic malware. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pages 411–420. IEEE, 2007.

[47] Yujia Zhang, Kuangyan Song, Yiming Sun, Sarah Tan, and Madeleine Udell. " why should you trust my explanation?" understanding uncertainty in lime explanations. *arXiv preprint arXiv:1904.12991*, 2019.

[48] Saman Zonouz, Julian Rrushi, and Stephen McLaughlin. Detecting industrial control malware using automated plc code analytics. *IEEE Security & Privacy*, 12(6):40–47, 2014.