Navigating Concept Drift and Packing Complexity in Malware Family Classification

Numan Halit Guldemir[1], Oluwafemi Olukoya[1] and Jesús Martínez-del-Rincón[1]

[1]*Centre for Secure Information Technologies (CSIT), Queen's University Belfast, United Kingdom*

Abstract

In the rapidly evolving cybersecurity landscape, the classification of malware families presents significant challenges due to the dynamic nature of malware, a phenomenon known as concept drift. This research classifies Windows PE malware families using static analysis of raw opcode sequences. By leveraging Convolutional Neural Networks (CNNs) to extract unique features from these sequences, our approach achieves high classification accuracy rates of 97.93% and 89.55% on the Microsoft Malware Classification Challenge and BODMAS datasets, respectively. We also conducted a temporal analysis on BODMAS over 13 months to observe the evolution of malware families and identify periods where our model's accuracy decreases. We implemented a retraining strategy, allowing us to observe how retraining the model with new data helps it adapt to new malware patterns. The study also examined the impact of packed malware and different types of packers on the model's performance. Our findings indicate that packed malware significantly affects the model's accuracy, with some packers having a more pronounced impact than others. These results underscore the importance of regular model updates and specialized handling of packed malware to maintain robust detection capabilities.

Keywords

malware family classification, concept drift, Windows PE malware, temporal analysis, packed malware, packer

1. Introduction

Detecting malware cannot be understated, and a great focus lies in identifying family-related threats. As technology rapidly grows, so does our reliance on various software, making us vulnerable to cyberattacks. This has led to an increase in harmful software created by cyberattackers. A malware statistics and trends report by AV-Test [1] indicates that approximately 300,000 new malware emerge daily. This rise is due to the accessibility of malware construction toolkits, which empower even individuals without the expertise to create malware samples [2]. These toolkits facilitate the implementation of sophisticated methods like polymorphism and metamorphism. Consequently, the need to prevent the spread of these harmful files and effectively detect them remains of utmost importance.

One critical challenge in the field of malware detection is the continuous evolution of malware. Building reliable detection models becomes increasingly challenging as malware develops and changes over time. This is because when these models come across new or evolved types of malware, they may have difficulty accurately identifying them. This phenomenon is referred to as concept drift [3], model aging [4] or time decay [5] in the literature. Drifted samples are prone to being misclassified by detection models. Therefore, it is essential to identify and monitor concept drift in these models, as it enables the detection of when the model is becoming outdated and incapable of effectively identifying new and emerging malware threats.

In this study, the primary focus of the malware detection system is on Windows malicious applications, commonly referred to as Windows PE malware. This focus is based on the observation that, despite the increasing prevalence of malware targeting other platforms such as Android, Windows remains the primary target for malware creators [1]. Furthermore, Windows PE malware exhibits distinct characteristics that set it apart from other types of malware. These differences in behavior and structure

CAMLIS'24: Conference on Applied Machine Learning for Information Security, October 24–25, 2024, Arlington, VA
nguldemir01@qub.ac.uk (N. H. Guldemir); o.olukoya@qub.ac.uk (O. Olukoya); j.martinez-del-rincon@qub.ac.uk (J. Martínez-del-Rincón)
0000-0003-1202-6841 (N. H. Guldemir); 0000-0003-2771-2553 (O. Olukoya); 0000-0002-9574-4138 (J. Martínez-del-Rincón)

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073

make it necessary to develop specialized detection models and strategies tailored to the unique challenges posed by Windows PE malware.

Various research on malware detection showcases a spectrum of approaches, from signature-based to behavior-based methodologies [6]. Conventional approaches heavily relied on signatures and hashes to identify binaries. Nevertheless, these techniques are susceptible to exploitation, as malware creators can effortlessly manipulate them; for instance, they can alter hashes using methods like polymorphism or metamorphism [7]. It is important to note that, by adopting these methods, alterations can be made to the malware's binary structure, including its hash, without affecting the underlying malicious behavior. This ability to evade signature-based detection emphasizes the need for more sophisticated approaches to malware detection.

Static analysis presents a rapid means to extract characteristics of PE malware. What distinguishes this approach is its independence from execution, thus rendering it a secure option. Through static analysis, a range of attributes can be derived from PE malware, encompassing API calls, byte sequences, string patterns, opcodes, and the inherent characteristics of the PE malware itself [8]. On the other hand, dynamic malware analysis involves executing malware in controlled environments like sandboxes to examine its behavior. This method requires more time and resources than static analysis and faces scalability challenges. Some malware can even detect when running in controlled environments and adjust its behavior to avoid detection, making dynamic analysis less effective [9]. Another method known as memory analysis has gained popularity in recent years. This technique has piqued the interest of malware analysts due to its ability to provide a comprehensive assessment of malware [10, 11, 12, 13]. However, this method can be time-consuming, with generating a single memory dump taking up to 2 minutes, and the resulting large dump files exceeding 1 gigabyte. These challenges intensify resource requirements and hinder scalability and efficiency in malware detection and analysis [10].

This research leans towards utilizing static features, mainly because of the limitations of the alternative methodologies discussed. Within these static attributes, the emphasis on using opcode sequences stems from their potential to offer valuable insights into the operational characteristics of binary files, all without necessitating their execution [14]. Opcodes are the fundamental building blocks of executables, comprising a structured assembly of directives [15]. By examining opcode sequences, important patterns of behavior can be extracted. This approach eliminates the necessity for real-time execution, making the analysis process more efficient and secure.

The contributions of this paper are highlighted below:

- We perform an initial temporal analysis to observe how the performance of our model changes over time, providing insights into the presence of potential concept drift.
- We conduct preliminary retraining of the model with random samples from specific months to observe how its performance may improve as it encounters new data patterns over time.
- We examine the impact of packed malware and different packers on model performance.

This paper is organized as follows: Section 2 reviews relevant research on malware detection, malware family classification and concept drift. Section 3 describes our methodology, including data processing and model design. Section 4 outlines the datasets and experimental setup. Section 5 discusses our results and key findings, and Section 6 examines the effects of packed malware. Finally, Section 7 concludes with insights and future directions.

2. RELATED WORK

2.1. Opcode-based Malware Family Detection

Opcodes, which represent the fundamental instructions executed by a binary, provide valuable insights into its behavior and characteristics. One notable study is Bilar's work, which made significant contributions to malware detection through opcode analysis [16]. In this study, the author investigated an approach for identifying malicious code through the statistical analysis of opcode distributions.

The results showed statistical distinction in opcode distributions between malware and non-malicious software, with infrequent opcodes demonstrating a higher predictive capability. Parildi et al. [17] utilized opcodes that are obtained at runtime to make a binary classification of PE samples. They used Word2Vec to generate opcode word embeddings, followed by CNN, RNN and Transformers. McLaughlin et al. [18] proposed an Android malware detection system using a deep convolutional neural network. The system performs static analysis of raw opcode sequences from disassembled programs to learn features indicative of malware. Lu [19] employed opcodes of PE malware to determine malware family statistically. They propose a two-stage LSTM model for malware detection, which utilizes two LSTM layers and one mean-pooling layer to obtain the feature representations of opcode sequences of malware. The experiments were conducted on a 969 malicious and 123 benign files dataset. Kakisim et al. [2] introduced a malware detection method called SOEMD. This approach uses static analysis to detect malware by creating a new feature space of sequential opcode embeddings through Random Walk and Skip-Gram techniques. Kale et al. [20] investigated the use of word embedding techniques (HMM2Vec, Word2Vec, BERT, and ELMo) for malware family classification. They employed four classification algorithms (k-nearest neighbors, support vector machines, random forests, and convolutional neural networks) on each word embedding technique. The experiments are conducted on opcode sequences from seven malware families.

In summary, all these methods demonstrated how the use of opcode sequences and deep learning techniques is a powerful and effective combination in the detection of malware as well as the malware family. However, since they rely on a training set collected at a given time, they are subject to performance degradation over time. Additionally, these methods did not consider packed samples and packer techniques, which can significantly impact the performance and effectiveness of malware detection systems [8].

2.2. Concept Drift

Although our research on concept drift is preliminary, it is essential to highlight the significance of this phenomenon and recognize the valuable contributions of recent studies in this area. Jordaney et al. [3] designed a framework called TRANSCEND to identify when a classification model was consistently misclassifying new data, indicating that concept drift had occurred. The TRANSCEND framework uses statistical metrics to detect concept drift. It employs a non-conformity measure to assess how well test samples fit, using a p-value. Building on this, Barbero et al. developed TRANSCENDENT, which has enhanced the efficiency and reduced the computational costs of the original Transcend framework. Yang et al. [22] developed framework called CADE. The primary objective of the framework is to identify drifting samples that deviate from the existing class and provide explanations for the occurrence of drift. Ceschin et al. [23] investigated the impact of concept drift on Android malware detection, utilizing datasets from DREBIN and AndroZoo. They examined how evolving malware characteristics affect machine learning classifiers and proposed a data stream learning pipeline that updates both classifiers and feature extractors. The authors [24] proposed a three-step forensic exploration method to evaluate concept drift and model performance over time. This approach involves comparing models trained with different temporal windows, evaluating the influence of temporal information on model drift, and conducting a detailed misclassification and feature analysis to identify the causes of drift and model failure.

To sum up, the objectives of our experiments revolve around addressing the following research questions:

RQ1: *Given the use of a recent dataset, can opcode analysis still effectively provide insights into the behavior and functionality of various malware families?*

RQ2: *How does the performance of our opcode-based classification model change over time when applied to Windows PE malware, and what patterns can we observe?*

RQ3: *What initial steps or observations can be made towards understanding and addressing concept drift in an opcode-based analysis of Windows PE malware?*

RQ4: *How do packed samples and different packers influence the performance of malware family classification models?*

3. METHOD

We employ a deep learning approach inspired by natural language processing techniques to process the opcodes. First, we transform these opcodes into word embedding arrays. We then utilize a CNN model to classify the malware using these transformed representations.

3.1. Disassembly of PE Malware

In this study, we employed the distorm3 disassembler to extract instructions from PE malware samples. Distorm3 [25] is a disassembler tool that converts binary code into human-readable assembly language. To streamline our analysis and focus on high-level behavioral patterns, we discarded operands similar to [18, 2, 20, 17], which specify the data to be manipulated by the instructions, from the extracted instructions.

3.2. Packer Detection

To detect packers in the BODMAS dataset, we employed the Detect It Easy [26]. Detect It Easy is known for its comprehensive capability to identify various types of packers embedded in PE files. This tool provides detailed information regarding the presence of packers within the PE binaries. By utilizing Detect it Easy tool, we were able to identify whether a sample was packed as well as packer information, which enabled us to analyze the impact of packed samples and packers on our model's performance. This detection step is crucial for understanding how different packing techniques may affect the performance of our malware classification model and for addressing the specific challenges posed by these techniques. Additionally, as we do not have binaries in the BIG2015 dataset, we were unable to apply this method as it requires actual binaries.

3.3. Model Architecture

Leveraging deep learning techniques, we automated the feature extraction process for analyzing PE malware through opcode sequences. We integrated an opcode embedding layer into our model architecture. Unlike integer-encoding and one-hot vector representations, which are limited in capturing complex patterns, the embedding layer enables the model to learn patterns and semantic relationships within the opcode data. Initially, the weights of the embedding layer are set randomly and updated during training. This allows the embedding layer to represent opcodes in a continuous vector space, where similar opcodes are positioned closer together. This approach reduces the dimensionality of the input data, alleviating computational complexity and improving generalization across various opcode sequences.

Building upon the embedding layer, our model includes several key layers to enhance the analysis. The embedding layer has an embedding dimension of 256. We applied a dropout layer with a rate of 0.25 after the embedding layer to prevent overfitting. The first convolutional layer uses 1024 filters with a kernel size of 7 and ReLU activation to capture local patterns of consecutive instructions within the opcode data. Following the convolutional layer, a global max pooling layer reduces dimensionality while retaining significant features. A dense layer with 1024 units further interprets these features, learning non-linear combinations. Another dropout layer with a rate of 0.25 is added before the final softmax classification layer, which translates the learned features into probability distributions across different malware categories. The model is trained using the Adam optimizer with a learning rate of $1e-4$. This comprehensive structure allows the model to provide insights into the most likely classification for a given opcode sequence. The model structure is given in Figure 1.

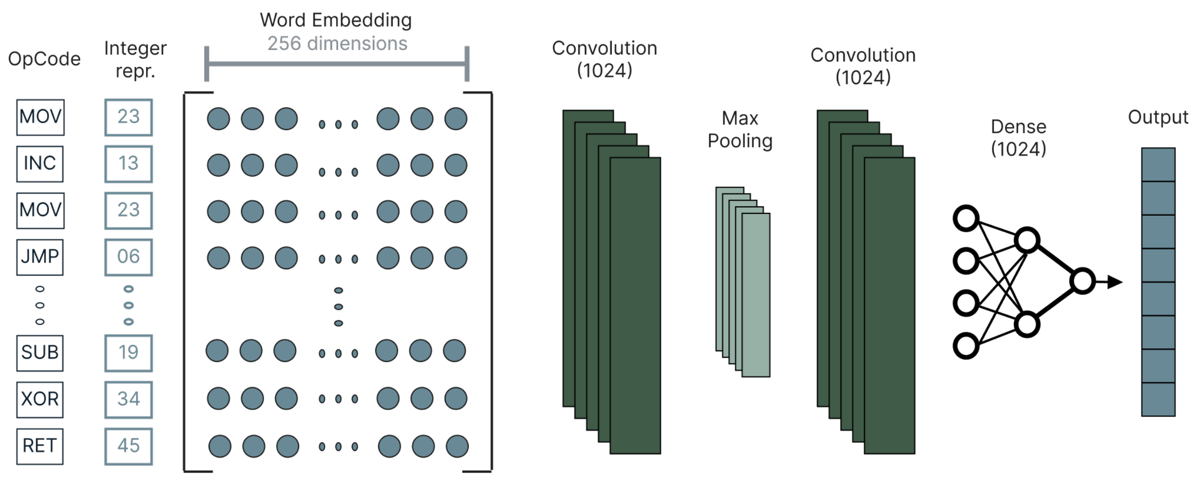


Figure 1: Architecture of the opcode-based convolutional neural network model for malware family classification.

4. Dataset and Experimental Setup

This section describes the dataset utilized and the preprocessing procedures applied before training our malware family classification model.

4.1. Dataset 1: BIG2015

Microsoft Malware Classification Challenge (BIG 2015) dataset is a common benchmark in the field [27]. The dataset comprises approximately 10,868 training and 10,873 testing PE malware which were labelled into 9 different families. Each malware sample provided by Microsoft is represented through two files: an assembly `asm` file containing low-level language code, as well as a corresponding compiled `bytes` file containing CPU machine code. Table 1 provides an overview of the distribution of samples among the different malware families.

Table 1
Sample Distribution Across Malware Families in the BIG2015 Dataset.

Name	Samples	Type
`Ramnit`	1541	Worm
`Lollipop`	2478	Adware
`Kelihos_ver3`	2942	Backdoor
`Vundo`	475	Trojan
`Simda`	42	Backdoor
`Tracur`	751	Trojan Downloader
`Kelihos_ver1`	398	Backdoor
`Obfuscator.ACY`	1228	Obfuscated Malware
`Gatak`	1013	Backdoor

4.2. Dataset 2: BODMAS

BODMAS dataset is an open dataset comprising a collection of 57,293 malware samples and 77,142 benign samples (a total of 134,435) [28]. In this dataset, malicious files were gathered between August 29, 2019, and September 30, 2020, and benign samples were collected from January 1, 2007, to September 30, 2020, aiming to represent the real-world distribution of PE binaries. The dataset also includes family information for malicious samples, encompassing a total of 581 families. Additionally, the authors have provided timestamp information indicating the first-seen (also referred to as first-submission) date, which is based on VirusTotal reports.

4.3. Experimental Setup

We implemented several preprocessing steps before the training and testing process. In the BIG2015 dataset, opcodes were extracted from the .asm files using a regex pattern to isolate only the opcodes. For the BODMAS dataset, which consists of actual PE binaries, distorm3 disassembler [25] was employed to perform the opcode extraction. Operands were removed from the instructions in both datasets to simplify analysis and capture the essential behavior of the malware. Additionally, the input was limited to the first 200 opcodes for each sample. This number was determined based on the results of our preliminary experiments, as shown in Figure 2. This approach helped reduce complexity and provided a consistent input size for our deep learning model.

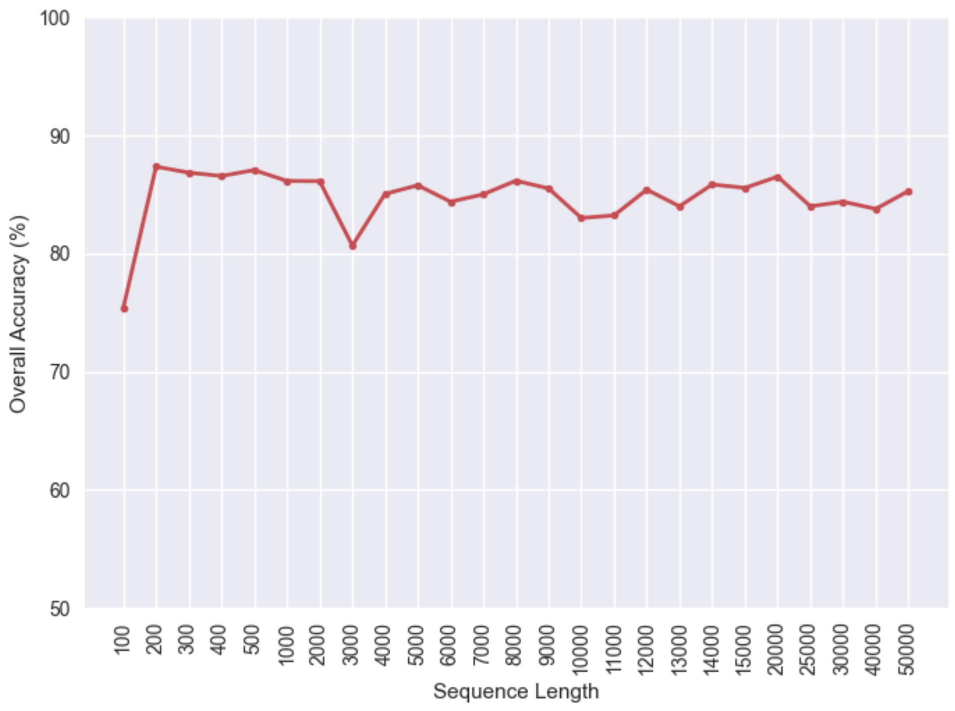


Figure 2: Model accuracy for varying opcode sequence lengths (BODMAS dataset).

5. EVALUATION

In the context of our study on PE malware, we conducted a series of four distinct experiments to comprehensively evaluate our proposed approach and perform temporal analysis and concept drift analysis and mitigation. These experiments were designed to delve into our methodology's performance. The foundational datasets for our analysis were sourced from *Microsoft Malware Classification Challenge (BIG 2015)* [27] and *BODMAS* [28]. We adopted our deep learning-based approach using opcode word embeddings (see Subsection 3.1) to analyze the opcodes within the same Microsoft dataset. These initial investigations aimed to establish a comparative benchmark for our baseline model's performance in a domain already replete with existing works. Shifting our focus to concept drift, we first applied our word embedding model to the entirety of the BODMAS dataset. Furthermore, to measure the temporal evolution of our models' efficacy, we conducted our final experiments using the BODMAS dataset separated into individual months.

5.1. Experiment 1: BIG2015

In this experiment, we evaluated the effectiveness of our model using the Microsoft Malware Classification Challenge dataset. Despite being relatively older, the BIG2015 dataset was chosen to provide an overall idea of our model's performance. We extracted opcodes from the .asm files using regular expressions to focus solely on opcode analysis. We enhanced the dataset quality by filtering out files containing only repetitive instructions, excluding 54 samples. Due to the absence of testing labels, the training dataset was used for both training and testing, employing a 5-fold cross-validation method. This involved splitting the dataset into five subsets, with each subset serving as a testing set once, while the remaining portions were used for training. The training dataset was split into an 80:20 ratio in each fold, ensuring that every family was represented in both the training and test sets.

The model described in Subsection 3.3 was used for our experiments, with results shown in the normalized confusion matrix in Figure 3. In the confusion matrix, rows represent the true class labels, and columns indicate the predicted labels. Each cell shows the normalized proportion of predictions, with each row summing to 1, allowing for a straightforward comparison of the model's performance across classes. Additionally, Table 2 includes a comparative analysis with other state-of-the-art studies as cited in those papers. Although there might be minor variations in experimental setups, this test confirms the effectiveness of our model as one of the reliable baselines for assessing concept drift.

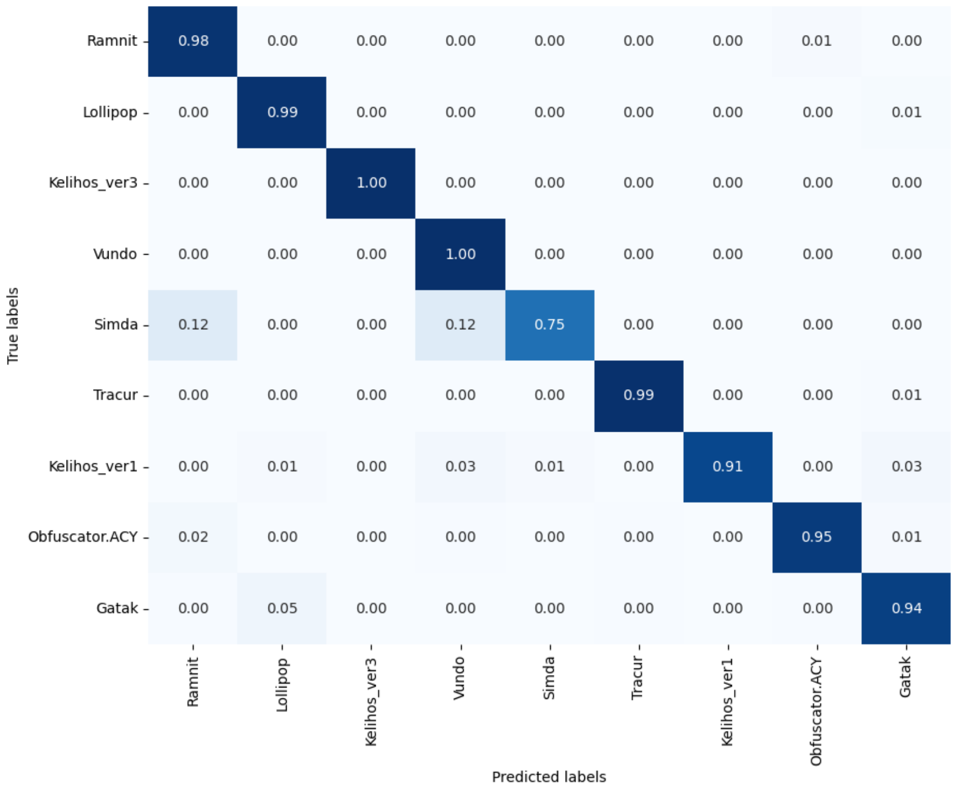


Figure 3: Confusion matrix for malware family classification using BIG2015 dataset.

5.2. Experiment 2: BODMAS

In this experiment, the *BODMAS* dataset is utilized [28]. While the dataset offers the actual binary files for the malicious samples, benign files are not provided by authors due to copyright concerns. Due to limitations in obtaining benign binaries, our investigation focused solely on malicious samples for our experiments.

We employed the most prominent 20 malware families from the dataset. This choice ensured precise training by establishing a minimum sample size. In our experimental phase, we constrained opcodes to

Table 2
Performance Comparison of Various Models on the BIG2015 Dataset.

Algorithm	Features	Accuracy
Alaeiyan et al. [29]	Opcode frequencies	97.52%
Lu et al. [30]	Byte sequences	96.96%
Zhu et al. [31]	Byte & opcode sequences	99.05%
Ahmadi et al. [32]	A set of static features	99.76%
Our model	Opcode sequences	97.93%

200, guided by the experiments given in Figure 2, as the highest overall accuracy score achieved using 200 opcode length.

Full training on BODMAS

In this experiment, we evaluated our model on the BODMAS dataset to understand its performance on a more recent dataset. Given the availability of timestamps for each sample, we opted not to use k-fold cross-validation. Instead, we followed the guidelines specified in [5] by splitting the dataset into an 80:20 ratio based on the first-seen timestamp. This approach involved using older samples to train the model and newer samples to test it, which helps eliminate temporal bias and simulate a realistic setting. Specifically, the training set includes samples from August 2019 to June 2020, while the testing set includes samples from June 2020 to September 2020. Additionally, 10% of the training data was reserved as a validation set to ensure the model's resilience. We used the same word embedding model from our initial experiment with the BIG2015 dataset (see Subsection 5.1).

The resulting testing dataset comprises 6,375 malware samples distributed across 20 different families. In the training dataset, there were a total of 1013 distinct vocabulary items, representing unique opcodes. As mentioned in Subsection 3.3, a word embedding size of 256 was employed, reducing the embedding size below 256 led to a slight drop in performance while increasing it beyond that value did not impact performance, despite increased resources and computational demands. The achieved accuracy was 89.55% across the 20 malware families, and a detailed confusion matrix is given in Figure 4. Additionally, Table 3 compares our model's performance with other studies. To facilitate a straightforward comparison with our monthly separated temporal analysis experiment, as described in Section 5.2, we used the most prevalent 20 families according to our training month, September 2019.

Temporal analysis on BODMAS

In this experiment, our primary goal was to conduct a temporal analysis of our model's performance. We categorized the malware samples based on their occurrence over 14 months, from August 2019 to September 2020. To ensure sufficient representation of each malware family for comprehensive training, we combined the data from August and September 2019. This decision was necessary because using August's data alone would not provide enough samples. We then evaluated the model's performance over the subsequent 12 months, from October 2019 to September 2020, allowing us to assess the model's adaptability and effectiveness over time thoroughly.

Our investigation focused on the top 20 malware classifications. While the model initially showed a relatively stronger accuracy of 83.14%, this accuracy varied significantly throughout the year. The performance fluctuated, as illustrated in Figure 5, with notable dips. Specifically, the accuracy ranged from a low of 65.49% to a high of 91.45%. This indicates variability in the model's ability to correctly classify malware samples over time, reflecting the possible impact of concept drift.

When analyzing only the non-packed malware, the model's performance demonstrated higher consistency and generally better accuracy throughout the year. The accuracy for non-packed malware remained above 85% for most months, suggesting that packed malware contributes significantly to the

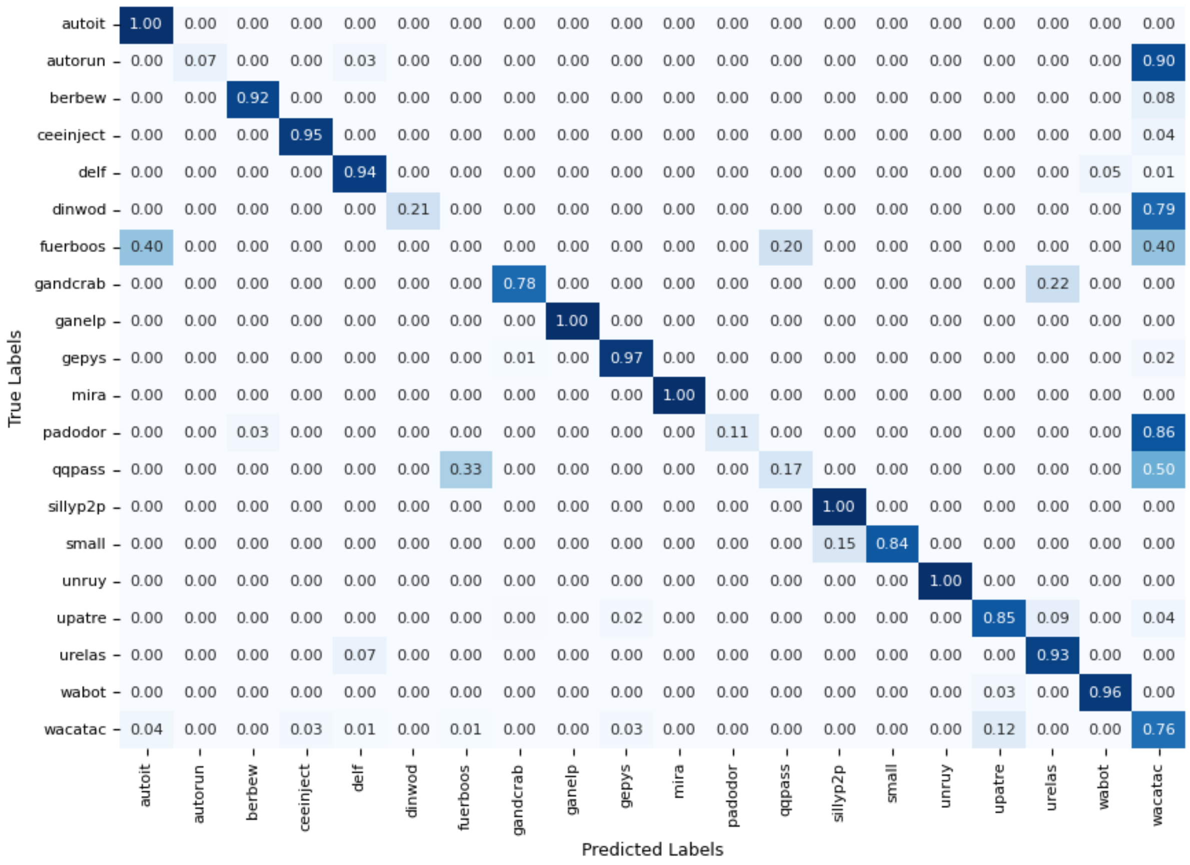


Figure 4: Confusion matrix for malware family classification using BODMAS dataset.

observed variability and overall reduction in model performance. This highlights the importance of effectively handling packed malware in malware classification tasks to maintain robust and reliable performance.

Concept drift mitigation

To explore and address the concept drift phenomenon, we conducted a preliminary experiment to understand how our dataset evolves. From October 2019 to September 2020, we selected 10 samples from each malware family (totaling 200 samples) each month. These samples were used to fine-tune our classification model, originally trained on August and September 2019 data. We then evaluated the model's performance on the entire dataset for each subsequent month.

The result, shown in Figure 6, illustrates the effectiveness of the fine-tuned models. The blue line represents the base model, while the red line depicts the fine-tuned model. The captions indicate the months during which additional samples were collected for retraining. The minimal difference between the blue and red lines indicated that retraining the model with samples from October 2019 to January 2020 did not significantly improve performance. However, a significant improvement is observed when the model is fine-tuned with data from February 2020. Specifically, the model's accuracy on February data increased; importantly, the accuracy on subsequent months' data also showed improvement. For example, the accuracy in March, which initially had the lowest accuracy of 71.5%, increased by approximately 20% up to 91.7% with the fine-tuned model, despite not being trained on any March data. This suggests that the model adapted well to new malware variants in February, which also benefited the performance in later months. This pattern is also observed for models retrained with samples from March, April, May, July, August, and September 2020, except for June, where the improvement was not as pronounced. This exception may indicate that June did not introduce significant new variants. These

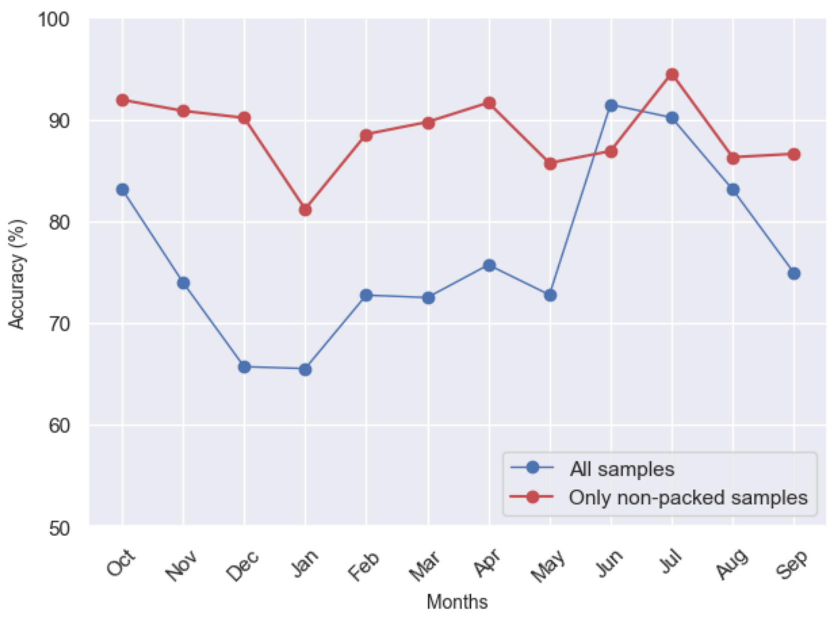


Figure 5: Temporal accuracy trends for malware classification on BODMAS dataset, comparing packed vs. non-packed malware.

Table 3
Performance Comparison of Various Models on the BODMAS Dataset.

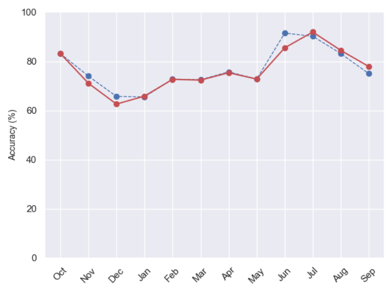
Research	Accuracy	Type	Number of classes	Temporal analysis
Lu et al. [30]	96.96%	Multi	11	No
Lu et al. [30]	93.42%	Multi	49	No
Hao et al. [33]	99.29%	Multi	14	No
Louk and Tama [34]	99.97%	Binary	-	No
Our work	89.55%	Multi	20	Yes

findings highlight the evolving nature of malware and underscore the importance of regularly updating the model with new data to maintain its accuracy. This preliminary experiment informs future research aimed at developing robust strategies for detecting and mitigating concept drift.

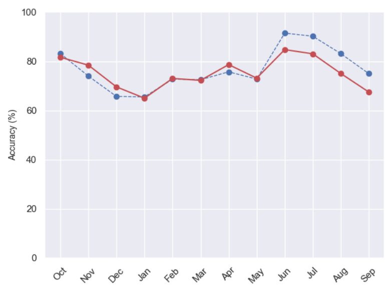
6. Impact of Packed Malware and Packer

Packed malware and packers play a critical role in the field of malware analysis [8]. Packers are tools used to compress or encrypt executable files, often to obfuscate the code and evade detection by security software. When a file is packed, its original code is transformed into a more compact and unreadable format, which is then decompressed or decrypted back into its original form during execution. Malware authors commonly use this technique to hide malicious payloads and hinder reverse engineering efforts. Analyzing packed malware, therefore, requires specialized techniques to unpack and examine the underlying code, making it a challenging but essential aspect of modern cybersecurity practices.

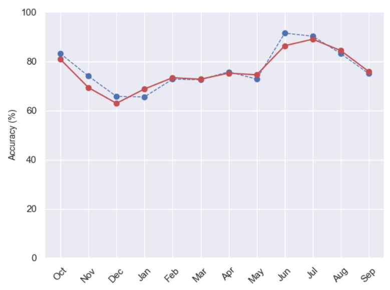
Our deeper analysis of incorrectly classified samples found that about 67% of these samples were packed. This led us to investigate the role of packers and their effect on our model's performance. We wanted to see if the packing process caused misclassification or if other factors were at play. We also noticed that not all packed malware were incorrectly classified. This suggested that the type of packer used might significantly influence classification accuracy, which is also indicated in [8]. To understand this better, we analyzed how different packers affected our model's ability to identify malware families correctly. Table 5 indicates the number of samples each packer has, how many were misclassified, and



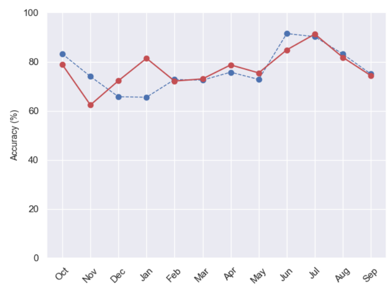
(a) October 2019



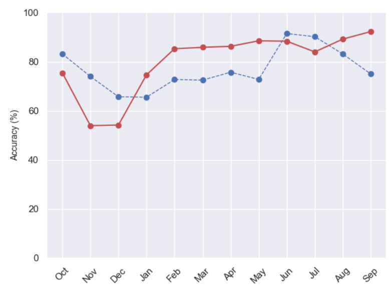
(b) November 2019



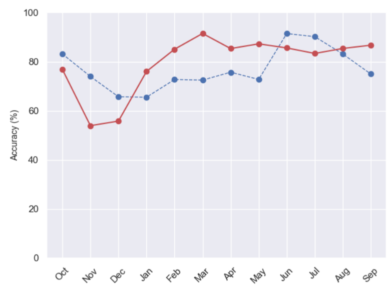
(c) December 2019



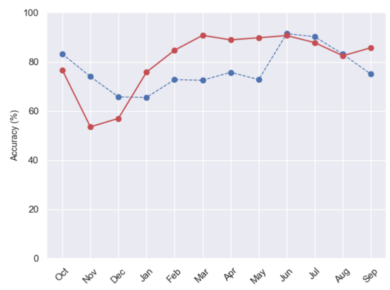
(d) January 2020



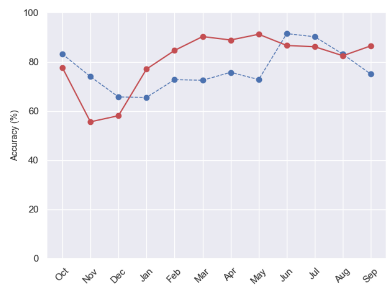
(e) February 2020



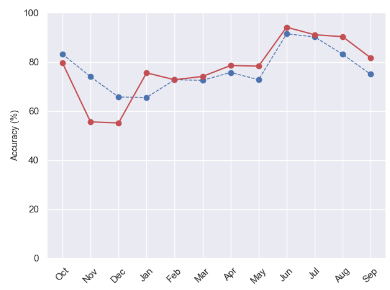
(f) March 2020



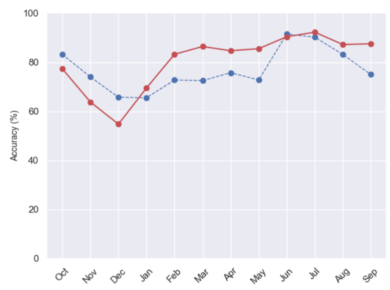
(g) April 2020



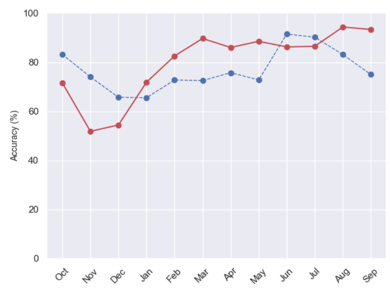
(h) May 2020



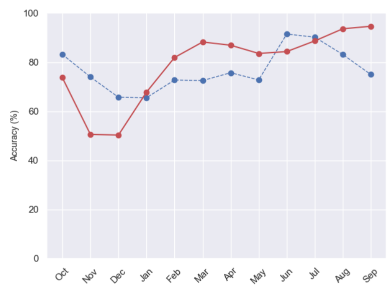
(i) June 2020



(j) July 2020



(k) August 2020



(l) September 2020

Figure 6: Fine-tuned model performance over time (October 2019 to September 2020). Blue line: Base model trained in August/September 2019. Red line: fine-tuned model with selected samples from the month indicated in the subtitle.

the percentage of misclassified samples for each packer.

In our analysis of various malware families, we observed significant variability in detection accuracy, sample size, and the prevalence of packed samples. Table 4 provides a detailed overview of this variability across different malware families. Notably, families such as *Dinwod*, *Fuerboos*, *Ganelp*, and *Wacatac* exhibited low average accuracy rates of 27.57%, 27.78%, 23.37%, and 59.90%, respectively. Many samples from these families were packed, likely contributing to the reduced classification accuracy. This high

Table 4
Summary of Detection Accuracy, Sample Size, and Packed Malware Proportion for Each Malware Family from October 2019 to September 2020, Highlighting Primary Packer Type.

Family	Average Accuracy	Total Sample	Percentage of Packed Sample	Majority Packer (Percentage)
autoit	98.32%	834	37.53%	UPX (100.00%)
autorun	79.33%	571	34.50%	PECompact (44.67%)
berbew	99.10%	1674	0.48%	DxPack (100.00%)
ceeinject	96.63%	1099	0.45%	UPX (100.00%)
delf	96.93%	749	1.20%	AsPack (55.56%)
dinwod	27.57%	1788	85.57%	DxPack (47.58%)
fuerboos	27.78%	198	90.40%	UPX (84.36%)
gandcrab	99.22%	769	2.21%	UPX (94.12%)
ganelp	23.37%	2148	76.63%	Petite (100.00%)
gepys	84.28%	935	13.69%	MPRESS (51.56%)
mira	98.15%	1889	0.95%	PECompact (94.44%)
padodor	9.91%	646	7.89%	DxPack (100.00%)
qqpass	95.50%	200	79.00%	Petite (43.67%)
sillyp2p	99.87%	1510	4.44%	UPX (100.00%)
small	77.90%	3059	3.04%	UPX (98.92%)
unruy	99.18%	734	0.41%	UPX (100.00%)
upatre	95.07%	3447	21.26%	MPRESS (54.16%)
urelas	79.25%	583	13.21%	PECompact (49.35%)
wabot	97.96%	3521	1.90%	MPRESS (100.00%)
wacatac	59.90%	4242	79.00%	UPX (51.12%)

prevalence of packed samples suggests that the packing process significantly impacts the ability to classify these malware families accurately.

To better understand the impact of different packers on misclassification, we analyzed the misclassified samples based on the packer used. We identified the total number of misclassified samples for each packer and the classes into which these samples were incorrectly classified. For instance, among the samples packed with the *Petite* packer, we found that out of 2045 misclassified samples, 1850 were misclassified as *Wacatac*, accounting for 90.45% of the misclassifications. Similarly, for the *MPRESS* packer, 124 misclassified samples were classified as *Upatre*, which is 97% of total misclassifications. This data highlights the significant influence of the type of packer on the model's misclassification behaviour, suggesting that certain packers may introduce features more strongly associated with specific malware families, leading to higher misclassification rates.

In conclusion, the presence of packed samples significantly impacts the model's classification performance. When a particular malware family predominantly uses a packer in the training data, the classifier labels any sample using that packer as belonging to that family. This results in misclassifications, where malware from different families using the same packer is incorrectly identified as the dominant family. Our study highlights a notable challenge in accurately distinguishing between malware families based on their packer associations. Furthermore, some packers have a more pronounced effect on misclassification, indicating that different packers vary in their impact on the classifier's performance.

7. CONCLUSION AND FUTURE WORK

In this study, our approach to malware family classification centered around utilizing opcodes to detect malware families through effectively implementing CNN. We began by employing a widely recognized Microsoft dataset to establish a robust benchmark for our CNN model. Building on this foundation, we extended the applicability of our model by applying the same architecture to classify the BODMAS dataset. To explore the temporal aspect, we trained our model using data from the initial period,

Table 5
Misclassification Rates by Packer Type in Malware Detection.

Packer	Total Samples	Number of Misclassified Samples	Ratio of Misclassification
ASPack	83	32	38.55%
DxPack	1346	858	63.74%
MPRESS	617	133	21.56%
PECompact	280	126	45.00%
Petite	3280	2054	62.62%
UPX	3022	1379	45.63%

determined by the first submission (first-seen) date on VirusTotal. We subsequently tested its efficacy over the following months. Our observations revealed significant variations in the performance of our model across different periods, indicating the dynamic nature of malware behaviors over time. Our results demonstrate that concept drift is a significant challenge in malware detection. As the model encounters new data over time, it initially shows a decline in performance. However, the model adapts to the new data patterns through periodic retraining with recent data samples, resulting in improved performance. This highlights the importance of continuous learning and model updating in maintaining high detection accuracy.

The impact of packers on malware classification was also thoroughly examined. Packers, which compress or encrypt executable files, significantly affect the classification performance by obfuscating the malware's features. Our analysis showed that certain packers led to higher misclassification rates, suggesting that the type of packer used plays a crucial role in the effectiveness of malware detection systems. Addressing the challenges posed by packed malware is essential for developing robust detection models.

For future work, our research will focus on an in-depth exploration of drift detection techniques to enhance our model's adaptability. This involves identifying changes in data distribution over time, enabling us to detect when the model's performance might be compromised. Additionally, we plan to conduct a comprehensive study to explain the underlying reasons for observed performance fluctuations across different temporal segments. By understanding the specific features and patterns contributing to the behavior of malware families, including the impact of packers, we aim to fortify our model's robustness.

Acknowledgments

Numan Halit Guldemir is supported by the Republic of Türkiye Ministry of National Education (MoNE-1416/YLSY).

References

[1] AV-Test, Malware statistics & trends report, 2023. URL: https://www.av-test.org/en/statistics/malware/, accessed: 12 May 2024.
[2] A. G. Kakisim, S. Gulmez, I. Sogukpinar, Sequential opcode embedding-based malware detection method, Computers & Electrical Engineering 98 (2022) 107703.
[3] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, L. Cavallaro, Transcend: Detecting concept drift in malware classification models, in: 26th USENIX security symposium (USENIX security 17), 2017, pp. 625–642.
[4] K. Xu, Y. Li, R. Deng, K. Chen, J. Xu, Droidevolver: Self-evolving android malware detection system, in: 2019 IEEE European Symposium on Security and Privacy (EuroS&P), IEEE, 2019, pp. 47–62.

[5] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, L. Cavallaro, {TESSERACT}: Eliminating experimental bias in malware classification across space and time, in: 28th USENIX Security Symposium (USENIX Security 19), 2019, pp. 729–746.

[6] F. A. Aboaoja, A. Zainal, F. A. Ghaleb, B. A. S. Al-rimy, T. A. E. Eisa, A. A. H. Elnour, Malware detection issues, challenges, and future directions: A survey, Applied Sciences 12 (2022) 8482.

[7] P. O'Kane, S. Sezer, K. McLaughlin, Obfuscation: The hidden malware, IEEE Security & Privacy 9 (2011) 41–47.

[8] H. Aghakhani, F. Gritti, F. Mecca, M. Lindorfer, S. Ortolani, D. Balzarotti, G. Vigna, C. Kruegel, When malware is packin'heat; limits of machine learning classifiers based on static analysis features, in: Network and Distributed Systems Security (NDSS) Symposium 2020, 2020.

[9] A. Afianian, S. Niksefat, B. Sadeghiyan, D. Baptiste, Malware dynamic analysis evasion techniques: A survey, ACM Computing Surveys (CSUR) 52 (2019) 1–28.

[10] C. Rathnayaka, A. Jamdagni, An efficient approach for advanced malware analysis using memory forensic technique, in: 2017 IEEE Trustcom/BigDataSE/ICESS, IEEE, 2017, pp. 1145–1150.

[11] R. Sihwail, K. Omar, K. A. Zainol Ariffin, S. Al Afghani, Malware detection approach based on artifacts in memory image and dynamic analysis, Applied Sciences 9 (2019) 3680.

[12] R. Sihwail, K. Omar, K. A. Z. Arifin, An effective memory analysis for malware detection and classification., Computers, Materials & Continua 67 (2021).

[13] T. Carrier, P. Victor, A. Tekeoglu, A. H. Lashkari, Detecting obfuscated malware using memory feature engineering., in: Icissp, 2022, pp. 177–188.

[14] S. Jeon, J. Moon, Malware-detection method with a convolutional recurrent neural network using opcode sequences, Information Sciences 535 (2020) 1–15.

[15] X. Ling, L. Wu, J. Zhang, Z. Qu, W. Deng, X. Chen, Y. Qian, C. Wu, S. Ji, T. Luo, Adversarial attacks against Windows PE malware detection: A survey of the state-of-the-art, Computers & Security (2023) 103134. ISBN: 0167-4048 Publisher: Elsevier.

[16] D. Bilar, Opcodes as predictor for malware, International journal of electronic security and digital forensics 1 (2007) 156–168.

[17] E. S. Parildi, D. Hatzinakos, Y. Lawryshyn, Deep learning-aided runtime opcode-based windows malware detection, Neural Computing and Applications 33 (2021) 11963–11983.

[18] N. McLaughlin, J. Martinez del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, A. Doupé, Deep android malware detection, in: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, 2017, pp. 301–308.

[19] R. Lu, Malware detection with lstm using opcode language, arXiv preprint arXiv:1906.04593 (2019).

[20] A. S. Kale, V. Pandya, F. Di Troia, M. Stamp, Malware classification with word2vec, hmm2vec, bert, and elmo, Journal of Computer Virology and Hacking Techniques 19 (2023) 1–16.

[21] F. Barbero, F. Pendlebury, F. Pierazzi, L. Cavallaro, Transcending transcend: Revisiting malware classification in the presence of concept drift, in: 2022 IEEE Symposium on Security and Privacy (SP), IEEE, 2022, pp. 805–823.

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

[23] F. Ceschin, M. Botacin, H. M. Gomes, F. Pinagé, L. S. Oliveira, A. Grégio, Fast & furious: On the modelling of malware detection as an evolving data stream, Expert Systems with Applications 212 (2023) 118590.

[24] F. Zola, J. L. Bruse, M. Galar, Temporal analysis of distribution shifts in malware classification for digital forensics, in: 2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), IEEE, 2023, pp. 439–450.

[25] G. Dabah, gdabah/distorm, 2023. URL: https://github.com/gdabah/distorm, accessed: 13 September 2023.

[26] Horsicq, Detect it easy: Program for determining types of files for windows, linux and macos., 2024. URL: https://github.com/horsicq/Detect-It-Easy, accessed: 01 July 2024.

[27] R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov, M. Ahmadi, Microsoft malware classification challenge, arXiv preprint arXiv:1802.10135 (2018). arXiv:1802.10135.

[28] L. Yang, A. Ciptadi, I. Laziuk, A. Ahmadzadeh, G. Wang, BODMAS: An open dataset for learning based temporal analysis of PE malware, in: 2021 IEEE Security and Privacy Workshops (SPW), IEEE, 2021, pp. 78–84.

[29] M. Alaeiyan, A. Dehghantanha, T. Dargahi, M. Conti, S. Parsa, A multilabel fuzzy relevance clustering system for malware attack attribution in the edge layer of cyber-physical networks, ACM Transactions on Cyber-Physical Systems 4 (2020) 1–22.

[30] Q. Lu, H. Zhang, H. Kinawi, D. Niu, Self-attentive models for real-time malware classification, IEEE Access 10 (2022) 95970–95985.

[31] X. Zhu, J. Huang, B. Wang, C. Qi, Malware homology determination using visualized images and feature fusion, PeerJ Computer Science 7 (2021) e494.

[32] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, G. Giacinto, Novel feature extraction, selection and fusion for effective malware family classification, in: Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy, 2016, pp. 183–194.

[33] J. Hao, S. Luo, L. Pan, EII-MBS: Malware family classification via enhanced adversarial instruction behavior semantic learning, Computers & Security 122 (2022) 102905. ISBN: 0167-4048 Publisher: Elsevier.

[34] M. H. L. Louk, B. A. Tama, Tree-based classifier ensembles for PE malware analysis: A performance revisit, Algorithms 15 (2022) 332. ISBN: 1999-4893 Publisher: MDPI.