Malware Detection for Portable Executables Using a Multi-input Transformer-based Approach

Ting-Li Huoh*, Timothy Miskell†, Onur Barut†, Yan Luo*, Peilong Li‡ and Tong Zhang†
*Department of Electrical and Computer Engineering, University of Massachusetts Lowell, Lowell, MA, USA
†Intel Corporation, Santa Clara, CA, USA
‡Department of Computer Science, Elizabethtown College, Elizabethtown, PA, USA
Email: {tingli_huoh, yan_luo}@uml.edu, {timothy.miskell, onur.barut, tong2.zhang}@intel.com, lip@etown.edu

***Abstract*—Malware is one of the leading cybersecurity challenges, as it disrupts the normal use of devices, causes financial losses, and steals user information. Deep learning-based methods have been increasingly used in the malware analysis field recently. In this work, we propose a novel multi-input Transformer-based approach for detecting malicious Portable Executable (PE) files. The PE raw bytes were partitioned into different byte sequences as multiple inputs in our proposed multi-input framework. This framework is comprised of convolutional neural networks (CNNs) and Transformer networks and is capable of independent learning of each input, thereby enabling a more expressive representation of the data. As a result, it is possible to capture both local spatial and time-series features, resulting in greater data comprehension. Our proposed approach outperforms the two reference methods, a LightGBM and a CNN-based model, as indicated by four metrics: accuracy, recall, precision, and F1 score.**

***Index Terms*—malware detection, deep learning, Portable Executable (PE) files, Transformer, multi-input deep learning**

I. Introduction

Identifying malware is a critical priority in cybersecurity, as malicious software poses serious threats to computer systems, networks, and sensitive data. Extensive research has been conducted on malware analysis across different applications. However, conventional malware detection methods based on a set of known signatures show limited scalability when dealing with a vast number of applications, and they lack the ability to detect recently produced malicious software. Subsequently, researchers begin to adopt machine learning (ML) approaches in malware detection [1].

With the rapid growth of neural networks, deep learning (DL) techniques have emerged as the cutting-edge of innovation for a wide range of applications, including malware detection [2]. As Microsoft Windows is one of the dominant desktop operating systems (OS) worldwide, from a practical point of view, we would like to investigate DL solutions for detecting malicious Portable Executable (PE) files. PE is a file format that provides the Windows OS loader information to handle wrapped executable code. Recent studies in PE malware detection using DL-based techniques have investigated many types of neural network architectures [3]–[7], such as fully-connected (FC) networks, convolutional neural networks (CNNs), and long short-term memory (LSTM). CNNs and recurrent neural networks (RNNs) models are commonly used in related studies [3], [5], [7]. CNNs are the best at capturing spatial features, but they are unable to capture representational information in the time-series domain. As for RNNs, despite the fact that they have achieved significant success in the natural language processing (NLP) field, the inherent sequential nature of RNNs prevents parallelization across all time points in a training sample. Hence, the Transformer architecture [8], which adopts a multi-head self-attention mechanism that prevents recurrent processing in RNNs, and obviates the consideration of sequence distance, has emerged as a viable candidate in our work.

Compared to ML-based methods, DL-based methods, specifically neural networks, possess the capability to directly integrate raw data without requiring supplementary feature extraction procedures. This is because of their ability to identify and uncover meaningful hidden features and representations from the input data. Nevertheless, using extracted features rather than raw data to train deep learning networks is still the common methodology in the field of PE malware detection studies [5]–[7]. According to the literature review, it indicates that there is a very small amount of research on training a model with PE raw byte sequences, which leads us to conclude that the potential of PE raw bytes hasn't yet been fully exploited. As such, in this study, we utilize PE raw data as network inputs for our proposed approach. Then we design an end-to-end Transformer-based model to detect malicious PE files and train our model using two public datasets, DeepDetectNet [6] and Ransomary [11].

The main contributions are summarized as follows:

- We leverage CNNs and Transformer networks and propose a novel Transformer-based DL approach for static Windows PE malware detection. By incorporating both architectures, the model can leverage the strengths of each component for greater data representation and comprehension, and effectively identify whether the given unknown PE file is malicious or benign.
- We explore the potential benefits of leveraging PE raw byte sequences that are acquired based on PE headers in our model and introduce a multi-input framework that can independently learn from each input, thereby enabling a more expressive representation of the data.
- We experimentally evaluate the performance of our proposed approach against two reference methods with different data fusion strategies and demonstrate its viability and applicability using the two public datasets.

979-8-3503-7099-7/24/$31.00 ©2024 IEEE

TABLE I
NOTABLE RELATED WORKS IN WINDOWS PE MALWARE DETECTION TASKS

Reference	Raw Data	Classifier			Network Input		
		ML	DL		Static[1]	Dynamic[2]	
This paper	✔	-	✔	CNN + Transformer	✔	-	Raw bytes: PE file header (24B), and optional header (240B).
Raff et al., 2018 [4]	✔	-	✔	CNN	✔	-	Raw bytes: the entire PE file.
Raff et al., 2017 [3]	✔	-	✔	FC, LSTM	✔	-	Raw bytes (328B): MS-DOS, COFF, and optional header.
Alam et al., 2023 [9]	-	-	✔	Transformer	✔	-	8 groups of hand-crafted features.
Chaganti et al., 2023 [5]	-	-	✔	CNN	✔	✔	Import functions, 4 general info, API call sequences, and PE image.
Fang et al., 2020 [6]	-	-	✔	FC	✔	-	Import functions, 10 general information, and bytes entropy feature.
Zhang et al., 2020 [7]	-	-	✔	CNN + BiLSTM	-	✔	API call sequences.
Anderson & Roth, 2018 [10]	-	✔	-	LightGBM	✔	-	8 groups of hand-crafted features.

[1] Static features: Adopting static features has a crucial advantage of allowing for the collection of features without the need to execute the file, which could function as the first step in recognizing possible threats and shielding the system from risk of infection.
[2] Dynamic features: Collecting dynamic features requires executing the file in order to gather information about its operation. Even though the binary is typically executed in a virtual environment, it still poses a risk to potential threats.

II. RELATED WORK

A. Static vs. Dynamic Malware Analysis

Table I summarizes relevant works that employ various methods for Windows PE malware detection. Static PE binary files can be used to derive either hand-crafted features, bytes n-gram, or raw bytes for static malware analysis. For instance, in [6], Fang et al. employed feature engineering to extract hand-crafted features from raw data to perform static feature extraction of PE files. In [10], Anderson and Roth introduced the EMBER dataset, which is an open dataset that includes extracted features from $1.1M$ binary PE files. They also demonstrated a baseline model for malware detection using LightGBM, a ML-based gradient boosting framework, with the EMBER dataset. Kolter and Maloof [12] used byte-level n-grams as features to train various ML-based models, yet it is reported that the performance of the n-gram approach is relatively poor and computationally expensive [13]. Meanwhile, there are a few studies that apply raw data from PE files directly to the network [3], [4]. In brief, utilizing raw bytes has the advantage of requiring less domain expertise and effort than extracting hand-crafted features.

Static malware analysis has the benefit of being able to identify unsafe files prior to their execution, whereas dynamic analysis involves executing the PE file in order to acquire information on how it functions. However, the main drawback of dynamic analysis is that it requires executing the PE file in order to gather information about its operation. Even though the binary is typically executed in a virtual environment, it still poses a risk to potential threats. Identifying potential security threats without infecting the analytic environment is critical. Hence, it is crucial to detect or identify malicious PE files prior to execution.

On the other hand, a multimodal deep learning approach has been proposed for Windows PE malware detection tasks. In [5], Chaganti et al. employed a feature-fusion method, in which both static and dynamic features are applied concurrently. An early fusion input strategy is adopted in their work where the four types of features are concatenated as a $2,128$ by 1 feature vector before feeding into a single CNN network. However, in this scenario, the network will acquire features based on the complete input sequence, potentially limiting its ability to learn more expressive feature representations independently based on the four types of features.

To obviate the limitation, in our work, we utilize raw data, specifically the raw byte sequences derived from the PE header, to train our malware detection model. Each byte sequence is fed into a distinct sub-network, with the goal of learning each input separately to build more expressive feature representations of the data. As a consequence, the model with a multi-input approach may learn to generalize across multiple inputs and recognize shared patterns, resulting in improved generalization performance and better management of noise or variations in the data. As such, we would like to explore the potential benefits of leveraging PE raw byte sequences in our model and examine the advantages of a multi-input approach.

B. DL Architectures for Windows PE Malware Detection

As most studies use CNNs or LSTM for implementation, the variations of deep neural network architectures that are trained using PE raw byte sequences have not been widely researched. The Transformer architecture, proposed by Vaswani et al. [8], has emerged as a viable candidate in our work due to its success in the NLP domain, an area mostly dealing with sequential data. Transformer is a sequence to sequence model that typically consists of an encoder and a decoder sub-network. Transformer and its evolved approaches are now the state-of-the-art methods for almost all NLP tasks. There are two prior works that used the Transformer architecture for PE malware detection [9], [14], and both works show that using the Transformer architecture is advantageous.

As a result, in this work, in contrast to prior related works that used only CNN, LSTM, or Transformer as network architectures [3]–[5], [9], we adopt both the CNN and Transformer architectures and propose a novel Transformer-based model for Windows PE malware detection. CNNs are proficient in capturing local patterns, whereas Transformers are effective at modeling global dependencies in the time-series domain. By integrating both architectures, the model can leverage the strengths of each component, resulting in greater data representation and comprehension.

TABLE II
DISTRIBUTION FOR DDN AND RANSOMARY

Dataset	Type	Number of Files	Total
DDN	Malicious	3,628	
	Benign	3,746	7,374
Ransomary	Malicious	2,871	
	Benign	4,208	7.079

III. METHOD

A. Data Description

In this work, we use the two public Windows PE datasets, the DeepDetectNet [6] dataset and Ransomary [11] dataset for our experimental studies. They will be denoted as **DDN** and **Ransomary** in the remainder of this paper. DDN and Ransomary both contain benign and malicious PE binary files, and has $7,374$ and $7,079$ PE files in total, respectively. Table II shows the distribution of each class for DDN and Ransomary.

The PE format is a file format used in 32-bit and 64-bit Windows OS for executables, object code, dynamic link libraries, and other types of files. The data structure contains the details required by the OS loader to control the wrapped executable code. The structure of a typical PE file is shown in Fig. 1. Every PE file begins with the DOS header, a 64-byte structure that defines the PE file as an MS-DOS executable. The MS-DOS stub is a valid application that runs under MS-DOS. The file header contains a 4-byte signature that identifies the file as a PE file and a 24-byte common object file format (COFF) file header that holds some information about the PE file. The optional header is the most important header that provides information to the OS loader. The size of an optional header can be found in the $SizeOfOptionalHeader$ field of the COFF file header. The section table contains section headers, where the number of sections is recorded in the $NumberOfSections$ field of the COFF file header. Last, the sections are where the contents of the PE file are stored.

B. Data Preprocessing and Network Input

Data segmentation is performed to capture the raw data that will be treated as our network input. The file header and the optional header are the two header fields that are used as our network inputs. As prior studies have suggested, the header information is sufficient and has the ability to identify malicious PE samples [3]. Moreover, through ablation studies, we acquire various configurations of header fields and observe that using the file header and the optional header has superior performance compared to other variants.

Specifically, the network takes in two sequences of bytes as inputs, where each byte sequence is given to one separate CNN sub-network. They will be referred to as **Input 1** and

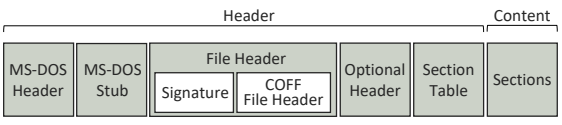

Fig. 1. The structure of a typical PE file.

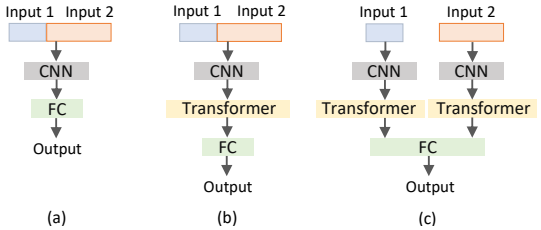

Fig. 2. Network architectures for (a) Study 1, (b) Study 2, and (c) Study 3.

Input 2 in the remainder of this paper. For both Input 1 and Input 2, the normalization is performed so that the values for each byte are scaled to the range of [0, 1].

1) Input 1: The length of Input 1 is 24 bytes, which is captured from the file header of a Windows PE file. A file header has 24 bytes in total, including 4 bytes of PE signature and 20 bytes of COFF file header. Location 0x3C of a PE file has the file offset to the PE signature.

2) Input 2: The length of Input 2 is 240 bytes. It is captured from the optional header of a Windows PE file. An optional header has a total of 224 and 240 bytes for a 32-bit and 64-bit PE file, respectively. The location of the optional header follows right after the file header. Input 2 is padded with 0s up to 240 bytes in the case of encountering a 32-bit PE file.

C. Experimental Design and Details

In the following, we describe three studies and two reference methods. Experiments using two datasets, and different network architectures with different data fusion strategies are aimed to evaluate our proposed approach. The overall architectures for the three studies are shown in Fig. 2, and the experimental studies are summarized in Table III. In our work, we conduct the experimental studies twice with DDN and Ransomary independently in order to show how the proposed approach generalizes to different datasets.

1) Study 1: A CNN and a FC network with an early fusion strategy comprise our baseline method, where Input 1, from the file header, and Input 2, from the optional header, are concatenated before they are fed into the network. We train our model using raw data with a length of 264 bytes for each input sample.

2) Study 2: A Transformer network is added after the CNN in Study 2. It also employs an early fusion strategy, as in Study 1. By making a comparison to Study 1, we would like to evaluate the performance of the Transformer-based model.

3) Study 3: Study 3 is implemented as a Transformer-based model with a different data fusion strategy from Study 1 and Study 2. It uses a late fusion strategy, where Input 1, from the

TABLE III
EXPERIMENTAL STUDIES AND INPUT DATA

Study Index	Network Architecture	Input Strategy
Study 1	CNN	Early fusion
Study 2	CNN + Transformer	Early fusion
Study 3	CNN + Transformer	Late fusion
Ref. [10]	LightGBM	Early fusion
Ref. [4]	CNN	Early fusion

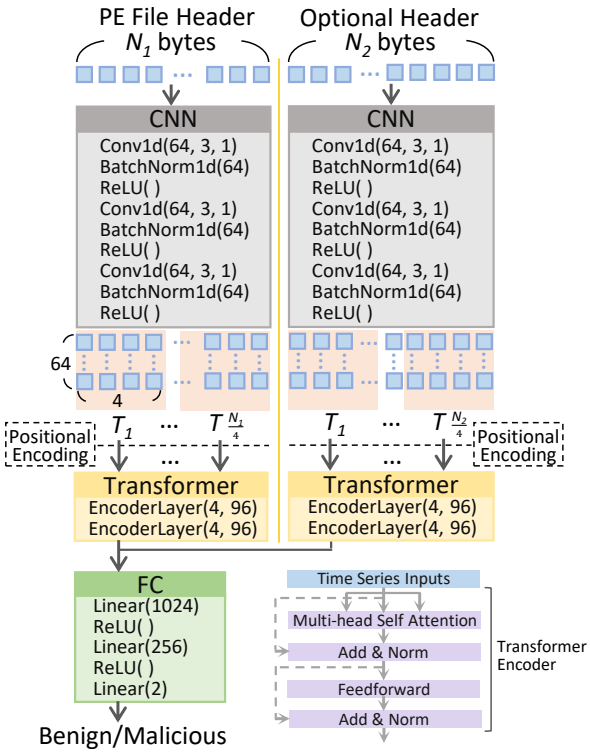

Fig. 3. The proposed end-to-end late fusion network architecture.

file header, and Input 2, from the optional header, are fed into two separate CNN sub-networks and two separate Transformer sub-networks. The output of two Transformer networks will be concatenated as the input of an FC layer. Via Study 3, we would like to evaluate whether the Transformer-based model can benefit from the late fusion strategy.

4) Reference Methods: Our comparison methods are based on two state-of-the-art architectures in Windows PE malware detection: an ML-based model, LightGBM [10], and a CNN-based model, MalConv [4]. In its original form, the LightGBM model uses eight groups of extracted features as the network input, while the MalConv model uses the entire raw bytes of a PE file as the network input. To ensure a fair comparison, we adjust the network input for both reference methods, as in Study 1 and Study 2.

D. End-to-end Network Architecture

The deep learning model used in this study is based on the Transformer network design. In the Transformer network architecture, we only employ the Transformer encoder since the goal of our research is to use the input sequence to identify malware rather than to forecast a sequence. A schematic of the Transformer network is depicted in Fig. 3. The proposed network is composed of two identical CNN, two identical Transformer and an FC sub-networks. One pair of a CNN and a Transformer sub-network is responsible for Input 1, and the other is for Input 2. The two Transformer sub-networks have two Transformer encoder layers that use 4 heads for the multi-head self attention, and 96 for the dimension of the feedforward network.

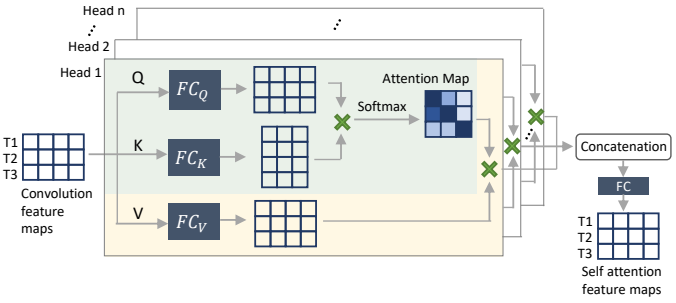

Fig. 4. Multi-head self attention layer architecture.

1) Multi-head Self Attention: The multi-head self attention mechanism is a module that calculates attention for each time point so that the network can concentrate on the most important time points during training. The multi-head self attention layer architecture is depicted in Fig. 4. The underlying method is the scaled dot-product attention, that takes these queries (Q), keys (K), and values (V) as inputs. Q, K, and V are identical and are convolution feature maps extracted from the CNN sub-networks. There are three FC sub-networks, performing a linear projection without changing the dimensions, for the inputs, Q, K and V, denoted as FC_Q, FC_K, and FC_V. It first computes the dot-product of $FC_Q(Q)$ and $FC_K(K)$. The result is then scaled by the square root of the dimensions of the keys, denoted as d_k, and put into a softmax function to produce an attention map. The attention map is then utilized to scale the values $FC_V(V)$ to generate attention. The formula of attention is defined as:

$$\text{Attention}(Q,K,V) = \text{softmax}(\frac{FC_Q(Q)(FC_K(K))^T}{\sqrt{d_k}})FC_V(V). \quad (1)$$

Multi-head self attention can be seen as multiple attention that are concatenated. Then, the concatenated multiple attention is fed into a FC network that performs linear projection and finally, generate self attention features maps. The whole process of the multi-head attention is defined as:

$$\text{MultiHead}(Q,K,V) = \text{FC}(\text{Concat}(head_1,...,head_n)), \quad (2)$$

where $head$ is Attention(Q,K,V), and n is the total number of heads.

E. Model Training and Validation

The Transformer-based network is implemented using PyTorch with an NVIDIA V100 graphics card. The PE files from both DDN and Ransomary are split into two portions that 70% is for the training dataset, and the remaining 30% is for validation. The network is trained using cross-entropy as the loss function. The parameters of the network are updated by minimizing the network loss using Adam optimizer with the learning rate 0.001. The batch size is set to 128, where it is the sample size from the training dataset that will be used to calculate a loss for updating the weights once. With these hyper-parameters, we train our network for 300 epochs.

TABLE IV
PERFORMANCE COMPARISON

Dataset	Study Index	ACC	Recall	Precision	F1
DDN	Study 1	0.963	0.962	0.971	0.966
	Study 2	0.969	0.969	0.973	0.971
	Study 3	**0.974**	**0.974**	**0.978**	**0.976**
	LightGBM	0.949	0.926	0.957	0.941
	MalConv	0.930	0.929	0.930	0.930
Ransomary	Study 1	0.940	0.936	0.965	0.950
	Study 2	0.944	0.939	**0.969**	0.954
	Study 3	**0.948**	**0.945**	**0.969**	**0.957**
	LightGBM	0.940	0.938	0.912	0.925
	MalConv	0.908	0.901	0.906	0.904

TABLE V
INFERENCE TIME COMPARISON

	Study 3	LightGBM	MalConv
Inference Time (per sample)	0.105 ms	0.075 ms	0.742 ms

IV. RESULTS

The performance metrics used for evaluating our baseline and proposed Transformer-based models that trained with DDN and Ransomary are tabulated in Table IV. As shown in Table IV, all metrics in Study 1 from DDN outperform the two reference methods. Similarly, Study 1 from Ransomary also performs better than the MalConv model in all evaluation metrics. While the LightGBM model from Ransomary has the same accuracy as Study 1, its precision and F1 score are significantly lower than Study 1.

Comparisons of the performance metric pairs (Study 1, Study 2) show that Study 2 outperforms Study 1 in every metric for both cases in DDN and Ransomary. In light of the quantitative results, it can be inferred that the Transformer network is able to supply additional representation information captured in the time-series domain. In other words, the model can take advantage of the capabilities of each component by incorporating both the CNN and Transformer architectures, resulting in better data representation and comprehension.

As shown in Table IV, our proposed method, Study 3, outperforms Study 1, Study 2, and the two reference methods in terms of accuracy, recall, precision, and F1 score for both DDN and Ransomary. The evaluation results indicate that, when taking into consideration the underlying structure of PE header fields, the two CNN sub-networks and two Transformer sub-networks with a late fusion strategy may benefit from learning and receiving additional independent representation information from spatial and temporal localities, respectively.

In addition, the inference time of our proposed model and the two reference methods are reported in Table V. The inference time of our proposed model, Study 3, is slightly longer than the LightGBM model since the trainable parameters of DL-based methods are typically larger than ML-based algorithms. On the other hand, given that the MalConv model's architecture was initially created for longer network input sequences, it is anticipated that the inference time will be much longer.

V. CONCLUSION

In this work, we implemented a multi-input Transformer-based model for Windows PE malware detection, where the model can learn each input independently using different sub-networks in order to acquire more expressive feature representations. Our proposed model demonstrates superiority in performing binary classification on PE files by leveraging the CNN and Transformer architectures and outperforms the two reference methods, a LightGBM and a CNN-based model, quantitatively, as indicated by the four metrics: accuracy, recall, precision, and F1 score. In our future work, we will characterize how the proposed model performs under datasets with various PE malware types or families and validate our model on a larger dataset for robustness.

ACKNOWLEDGMENT

This work was supported by a research grant from Intel Corporation.

REFERENCES

[1] C. Raghuraman, S. Suresh, S. Shivshankar, and R. Chapaneri, "Static and dynamic malware analysis using machine learning," in *First International Conference on Sustainable Technologies for Computational Intelligence: Proceedings of ICTSCI 2019*. Springer, 2020, pp. 793–806.

[2] M. Sahin and S. Bahtiyar, "A survey on malware detection with deep learning," in *13th International Conference on Security of Information and Networks*, 2020, pp. 1–6.

[3] E. Raff, J. Sylvester, and C. Nicholas, "Learning the PE header, malware detection with minimal domain knowledge," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, 2017, pp. 121–132.

[4] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole EXE," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[5] R. Chaganti, V. Ravi, and T. D. Pham, "A multi-view feature fusion approach for effective malware classification using deep learning," *Journal of Information Security and Applications*, vol. 72, p. 103402, 2023.

[6] Y. Fang, Y. Zeng, B. Li, L. Liu, and L. Zhang, "DeepDetectNet vs RLAttackNet: An adversarial method to improve deep learning-based static malware detection model," *Plos one*, vol. 15, no. 4, p. e0231626, 2020.

[7] Z. Zhang, P. Qi, and W. Wang, "Dynamic malware analysis with feature engineering and feature learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 01, 2020, pp. 1210–1217.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[9] M. M. Alam, E. Raff, T. Oates, and J. Holt, "Recasting self-attention with holographic reduced representations," in *2023 Proceedings of the 40th International Conference on Machine Learning*, 2023.

[10] H. S. Anderson and P. Roth, "Ember: An open dataset for training static PE malware machine learning models," *arXiv preprint arXiv:1804.04637*, 2018.

[11] O. Barut, T. Zhang, Y. Luo, and P. Li, "A comprehensive study on efficient and accurate machine learning-based malicious PE detection," in *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*. IEEE, 2023, pp. 632–635.

[12] J. Z. Kolter and M. A. Maloof, "Learning to detect malicious executables in the wild," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004, pp. 470–478.

[13] E. Raff, R. Zak, R. Cox, J. Sylvester, P. Yacci, R. Ward, A. Tracy, M. McLean, and C. Nicholas, "An investigation of byte n-gram features for malware classification," *Journal of Computer Virology and Hacking Techniques*, vol. 14, no. 1, pp. 1–20, 2018.

[14] E. M. Rudd, M. S. Rahman, and P. Tully, "Transformers for end-to-end InfoSec tasks: A feasibility study," in *Proceedings of the 1st Workshop on Robust Malware Analysis*, 2022, pp. 21–31.