$MGAP^3$: Malware Group Attribution Based on PerceiverIO and Polytype Pre-training

Yuxia Sun, *Member, IEEE,* Shiqi Chen, Song Lin, Aoxiang Sun, Saiqin Long, and Zhetao Li, *Member, IEEE*

Abstract—The escalating prevalence of Advanced Persistent Threat (APT) malware demands more effective methods to accurately attribute malware to specific APT groups. Traditional manual attribution processes are labor-intensive and error-prone, while existing automated methods are hampered by small dataset sizes, inadequate representation learning, and poor noise reduction during preprocessing. To address these challenges, we introduce the AMG25 dataset, which expands the pool of malware samples labeled with APT group affiliations. Concurrently, we propose the MGAP³ model (Malware Group Attribution based on PerceiverIO and Polytype Pre-training), which enhances attribution performance by incorporating hierarchical pre-training for disassembled codes and leveraging multi-view statistical features, all within a unified PerceiverIO architecture. This model adeptly captures complex program structures and interactions cross multiple code granularities, through a series of innovative polytype pre-training tasks. Additionally, we have developed a novel noise filtering technique that focuses on user-defined function codes, substantially reducing overfitting and boosting performance. Furthermore, a streamlined version of the model, MGAP³-Lite, has been developed to accelerate training while preserving robust performance. Extensive experiments have validated the effectiveness of our models and underscored the importance of the proposed pre-training technique.

Index Terms—APT malware, group attribution, PerceiverIO, pre-training task, static analysis

1 Introduction

As is widely recognized, Advanced Persistent Threat (APT) attacks have become a significant cybersecurity concern for global organizations, governments, and individuals [1]. APT groups carry out targeted attacks by deploying malicious software covertly, often driven by political, economic, or strategic motives. Attributing the malicious software discovered in APT attacks can help identify the responsible entities behind the attacks. This attribution is essential not only for better detection and defense against future attacks, but also for holding malicious actors accountable, as well as facilitating global cooperation among security professionals to combat the ever-evolving cybersecurity threats [2]. However, attributing APT malware poses significant challenges. APT attackers frequently employ various obfuscation techniques to evade analysis and hide their identities, making the attribution task complex and error-prone [3], [4]. Manual APT attribution is labor-intensive, time-consuming, and requires security experts with a high level of expertise and experience. In recent years, with the increasing frequency and sophistication of APT attacks, there is an urgent need in the cybersecurity field to develop effective automated APT malicious software attribution techniques.

Over the years, significant research has been conducted on the automatic group attribution of APT malware. These attribution approaches can be broadly categorized into dynamic and static methods. Dynamic methods [5], [6], [7], [8], [9], [10] involve tracking and analyzing the runtime behaviors of APT malware, while static methods [11], [12], [13], [14], [15], [16] directly analyze the malware codes. However, dynamic approaches heavily rely on virtual execution environments, which can make them time-consuming and less effective when dealing with malware that employs virtual machine escape techniques. In contrast, static approaches can overcome these limitations as they do not require executing the malware [17]. Therefore, this paper focuses on developing static group attribution techniques for APT malware. Early static attribution works [11], [12], [13] extract features from malware PE-files and utilize traditional machine learning models for group classification. In recent years, deep learning techniques have been increasingly employed to attribute malware functions or classify malware into groups [14], [15], [16]. For example, a state of art malware attribution method known as BinMLM [16] extracts n-gram opcode sequences from malware disassembly codes, pre-trains an LSTM encoder for each APT group, conducts joint training to generate malware encoding vectors, and subsequently attributes malware instances based on vector similarity. To generate high-quality malware code representations, pre-training-based malware attribution methods typically pre-train a sequential model on a large-scale external corpus, followed by fine-tuning

- *Yuxia Sun is with the College of Information Science and Technology, Guangdong Key Laboratory of Data Security and Privacy Preserving, Jinan University, Guangzhou 510632, China. Email: tyxsun@email.jnu.edu.cn.*
- *Shiqi Chen, Song Lin, Saiqin Long, and Zhetao Li are with the College of Information Science and Technology, Jinan University, Guangzhou 510632, China. Email: {shiqichen2000, simplesen960813}@gmail.com, saiqinlong@jnu.edu.cn, liztchina@hotmail.com.*
- *Aoxiang Sun is with the College of Information and Computational Science, Jilin University, Changchun, China. Email: 2835604014@qq.com.*

Manuscript received July 28, 2023; revised XX XX, 20XX.
This work was supported by National Key Research and Development Program of China (Grant 2021YFB3101201), National Natural Science Foundation of China (Grant No. 62032020 and No. U23B2027), Guangdong Key Laboratory of Data Security and Privacy Preserving (Grant No. 2017B030301004).
(Corresponding author: Zhetao Li)

this model on a local dataset. For instance, BinMLM pre-trains an LSTM model on an extensive binary file corpus [16], while SecureBERT_Mal utilizes a large malware dataset to fine-tune the SecureBERT model that has been pre-trained on a comprehensive security text corpus [41], [42].

However, current group attribution approaches for APT malware [5]-[16] are constrained by dataset size, representation learning, and data preprocessing techniques, which result in suboptimal attribution capabilities. Next, we will detail these challenges and further outline how we overcome them in this paper to enhance malware attribution:

Firstly, the datasets used in the existing malware attribution approaches are limited in size and not publicly accessible. For instance, the only openly accessible dataset, APTMalware [49], comprises merely 12 APT groups and approximately 3,000 malware MD5s. To address this issue, we introduce a novel dataset of malware with APT group labels, which is larger in size compared to APTMalware.

Secondly, malware attribution approaches often utilize pre-training techniques to enhance code representation, yet they face several challenges: (1) a reliance on a singular mask-based pre-training task similar to BERT's MLM (Masked Language Model), which fails to capture complex program structures and interactions at different syntactic levels; (2) the use of LSTM or Transformer architectures, which not only struggle to manage complex and long-distance dependencies within code sequences but also incur high computational costs [18], [19], [23]; (3) a high dependence on large external corpora, necessitating extensive efforts to build such databases. To overcome these obstacles, we propose a novel malware code representation technique using hierarchical pre-training tasks and the PerceiverIO architecture [26]. This approach extracts features at three different levels of abstraction (i.e., instruction, block, and function). Inspired by the varied pre-training tasks in NLP [24], [39], our approach employs 17 polytype pre-training tasks to learn intricate relationships between various granular code elements (e.g., opcodes, operands, blocks, functions). Our PerceiverIO-based model, equipped with mechanisms like cross-attention, effectively handles complex and long-distance dependencies within a fixed-size latent space, and reduces computational demands. This approach significantly boosts the performance of attribution models, even with a small local training set containing only a few thousand samples, thus eliminating the need for large external corpora.

Thirdly, existing malware attribution methods typically analyze single-modal code data, such as n-gram opcode sequences, visualized images of code features, or images of code bytes. To fully leverage the complementarity and integration of multimodal data to enhance attribution performance, developing a multimodal malware attribution technique is particularly crucial. In this context, this paper introduces a novel multimodal malware attribution model to integrate two distinct data modalities: the disassembled text and statistical feature images of malware, based on the PerceiverIO architecture [26]. We employ unified PerceiverIO architecture for both modalities, not only enhancing the internal consistency of the data representations but also facilitating potential synergies during the representation fusion process, ultimately leading to significant improvement in model performance.

Finally, in malware codes, the implementation of user-defined functions (UDFs) typically reflect the unique programming habits or intentions of specific attack groups, featuring distinct techniques like unique encryption or persistence mechanisms. In contrast, the implementation of non-user-defined functions (non-UDFs), such as standard library functions or system calls, is typically generic and lack specific group identifiers. Thus, for a malware attribution model, the code implementing non-UDFs can be viewed as noise. However, existing attribution models typically use data including this noise, potentially leading to overfitting on irrelevant code features and unnecessary data analysis. To address these challenges, we introduce a novel UDF filtering technique that enhance the focus on user-defined code sections more indicative of specific APT groups, thus improving classification performance and processing efficiency. Our technique targets only the implementation code of non-UDFs while preserving calls to these functions, maintaining crucial contextual information to boost model effectiveness.

In summary, this paper makes the following key contributions to advancing the field of malware attribution:

- **Dataset:** We introduce the AMG25[1] (APT Malware with Group-label) dataset, a new public resource with 5,188 malware samples from 25 APT groups, significantly expanding available resources and providing a more comprehensive foundation for further research compared to existing datasets.
- **Attribution Approach:** We propose a novel approach for malware group attribution named $MGAP^3$ (Malware Group Attribution based on PerceiverIO and Polytype Pre-training). This method effectively filters out code noise from non-UDFs, generates textual representations via hierarchical pre-training, extracts statistical representations from multi-view features, and integrates the multimodal code data, all utilizing the PerceiverIO architecture.
- **Pre-training Tasks:** We introduce innovative hierarchical code pre-training tasks, categorized into instruction-related, block-related, and function-related tasks. These enhance assembly-like code representations by delving into complex program structures and semantic relationships among code elements.
- **Filtering Strategy:** We propose a unique filtering technique that targets non-UDFs in the code to reduce noise, enhancing model performance and preventing overfitting.
- **Implementation & Validation:** We implement the full $MGAP^3$ model and its streamlined variant, $MGAP^3$-Lite, and validate their attribution effectiveness through extensive experiments. Additionally, we demonstrate the importance of key components of the model and the proposed pre-training tasks through experimental results.

The remaining sections of this paper are structured as follows. Section 2 provides an overview of the related

1. https://https://github.com/Yuxia-Sun/MGAP-AMG25

work in the field. Section 3 presents our proposed $MGAP^3$ method. In Section 4, we discuss the experimental setup and results. Section 5 discusses the limitations of our malware attribution approach. Finally, Section 6 concludes the paper and outlines future research directions.

2 Related Work

2.1 Malware group attribution

Automatic group attribution of APT malware, as a critical task of malware analysis and APT attack tracing, has been studied by many researchers in recent years. The group attribution approaches can be categorized into dynamic and static ones. The existing dynamic attribution methods [5]-[10] track and analyze the runtime behaviors of APT malware, such as process activities and network operations [5], [6], and the actual sequence of function calls [7]-[10]. However, dynamic approaches heavily depend on a virtual execution environment, such as a sandbox or virtual machine, to capture malware behavior data. Consequently, these dynamic methods tend to be time-consuming and ineffective when malware utilizes virtual machine escape techniques. Static attribution approaches can overcome the above limitations of dynamic ones by analyzing malware codes directly without executing them. As early works of APT malware group attribution, Laurenza et al. and Rosenberg et al. extract static features from malware PE-files over such file properties on PE-file header, import tables, section tables, and so on, and utilize traditional machine learning models as group classifiers [11], [12]. Liang et al. extract malware PE-file features of entry point, DLL (Dynamic Link Library), resource, and the number of sections, and train an RF (Random Forest) classifier for group attribution [13]. Wei et al. statically extract API system calls as malware features, and generate the vector representation of features based on LSTM and attention mechanism [14]. To attribute malware functions (rather than malware) to APT groups, Liu et al. extract word feature sequences from the disassembly code of each function's control flow graph, and generate the function embedding using an lstm model to learn the dependencies in the code segment [15]. Recently, Song et al. analyze the disassembly codes of APT malware, and extract consecutive opcode sequences as input of a group classifier, called BinMLM, to classify APT malware into 10 groups [16]. As the most recent static attribution model, BinMLM utilizes an LSTM encoder for each APT group and a gate layer for each malware sample to obtain malware encoding vectors, trains a decoder for each APT group, and finally classifies malware samples into APT groups based on vector similarity [16].

The aforementioned group attribution approaches of APT malware enable automated classification of malware into different groups. However, these approaches have limitations, including limited publicly available datasets, limited dataset size, noise interference from non-user-defined functions, reliance on LSTM-based models to learn representations of sequential data, and separate use of sequential code text or various code features rather than merging them into one representation.

2.2 Pre-trained model

Various deep learning-based sequence models, such as RNN, Encoder-Decoder[20], LSTM[21], GRU[22], Transformer[23], and more, have the potential to generate profound representations of sequential data, such as malware codes. However, it is worth noting that, currently, no pre-training techniques have been applied to generate malware representations specifically for APT malware attribution.

Compared with traditional recurrent neural networks (such as LSTM), Transformer has a self-attention and parallel training mechanism, which can capture long-distance dependencies better and have good performance when processing long sequences. Currently, Transformer is the most concerned sequence model and finds extensive application in tasks like NLP. One notable example is the renowned BERT model, which utilizes a multi-layer Transformer as its encoder [24]. In 2021, Jaegle et al. proposed the Perceiver model [25] and the PerceiverIO model [26] that can replace Transformer. The series of Perceiver models retain the self-attention and residual connections of the Transformer, and then use the cross-attention mechanism to convert high-dimensional input data into a fixed-dimensional latent space [25]. Compared with Perceiver, PerceiverIO is more general and can process multi-modal input, such as text and image, and perform better in language understanding and visual understanding [26]. Furthermore, PerceiverIO outperforms Transformer-based BERT on the GLUE language benchmark [26]. In view of the above advantages of PerceiverIO, this paper will construct a pre-training model of malicious code by stacking PerceiverIO encoders, that is, a code text representation model.

2.3 Pre-training task

The pre-training task is usually a self-supervised learning task in which the model is trained with unlabeled samples. The model learns the internal structural information and semantic relationships of the data by accomplishing the goal of the pre-training tasks, thereby generating a generic data representation. In 2018, Google introduced two pre-training tasks for BERT: MLM (Masked Language Model) and NSP (Next Sentence Prediction) [24]. They predict relationships between masked words or sentence pairs. BERT and its pre-training tasks have achieved success in NLP and computer vision [28], [29], [30]. Recently, in the field of source-code analysis, some works have been conducted on pre-training tasks over source codes [31], [32], [33]. Wan et al. provide evidence suggesting that integrating the syntax structure of code into the pre-training process could enhance code representations [33]. Guo et al. extracted the data flow diagram of the source code, and proposed the following three pre-training tasks: masked task, edge detection task, and pairing detection task of source code variables and data flow diagram variables [31]. Bui et al. created an abstract syntax tree(AST) for source codes and proposed a pre-training task to predict the subtree structure of the AST [32]. However, these source-code-oriented pre-training tasks are not suitable for malware for which source code is not available.

Currently, there have been some works on pre-training tasks in the realm of malware analysis [34]-[38], which aim to train a binary classifier for malware detection or a multiple-way classifier for malware families, rather than a classifier for APT groups. The MalBERT model proposed by Rahali et al. directly uses a malware dataset to fine-tune BERT's parameters [34]. This makes the corpus used for MalBERT's pre-training task not a malware corpus. In order to detect Windows malware, Xu et al. extracted the API call sequence of malware and defined two API-related pre-training tasks, namely the API mask task and the next API prediction task, for pre-training the Malbert model based on Transformer [35]. In order to make family classification of malware, Wu et al proposed a pre-training task NBP (Next Block Prediction) to encourage the model to learn the relationship of basic blocks [36]. For the detection and family classification of malware, Alvares et al. used BERT to generate word embedding for each opcode of the malware sample [37]. Oak et al. used BERT to generate an embedding of an Android program's API call as an input to a malware detector [38].

In contrast to the aforementioned approaches, we will introduce and utilize a novel set of pre-training tasks that focus on capturing the syntax and syntactic representations of disassembly codes in programs.

3 $MGAP^3$ Approach

To elevate the performance of malware group attribution, we introduce a novel approach named $MGAP^3$ (Malware Group Attribution based on PerceiverIO and Polytype Pre-training) in this section. $MGAP^3$ improves the preprocessing and representation learning of malware codes, addressing the limitations of current attribution methods: (1) To mitigate code noise from non-UDF implementations, we developed a UDF identifier to filter out this noise. This preprocessing strategy aims to sharpen the focus on UDF code sections, which are more indicative of specific APT groups, thereby enhancing attribution performance and processing efficiency. (2) To overcome the challenges associated with singular pre-training tasks and less effective architectures in traditional pre-training for code representation, we introduce a suite of hierarchical pre-training tasks coupled with a PerceiverIO-based model. These tasks are designed to deeply understand the program structure and the complex interactions among various code elements at different granularities and levels. Our model employs PerceiverIO encoders to efficiently capture complex, long-range dependencies within code sequences, thereby optimizing computational performance. (3) To address the limitations of unimodal code data, we combine malware representations from disassembled text and statistical feature images, both processed using PerceiverIO encoders. This unified PerceiverIO architecture aims to enhance data consistency and fosters synergies during fusion, ultimately boosting model performance.

Fig. 1 illustrates the architecture of our $MGAP^3$, which takes APT malware samples in binary codes as input and generates the predicted APT group label as the output. The $MGAP^3$ framework consists of four modules: (1) Pre-processing module, which initially acquires the assembly code of a malware sample, and then transforms it into specified pseudo-code text; (2) Statistical representation module, which generates deep representations of statistical features extracted from binary and assembly malware code; (3) Text-representation module, which generates malware text representations based on the pseudo-code; (4) Representation-fusion & Group-attribution module, which fuses malware text and statistical representations into the final representation, and classifies the malware into an APT group. Our $MGAP^3$ approach firstly runs the pre-processing module, then executes the statistical representation module and the text-representation module in parallel, and finally performs the Representation-fusion & Group-attribution module. In the following four subsections, we will provide detailed explanations of each of these four modules/steps.

3.1 Pre-processing module

As shown in Fig. 1, the pre-processing module of $MGAP^3$ firstly converts the binary code of a malware instance into its disassembly code using a disassembly tool such as IDA pro, and then normalizes the assembly code. The pre-processing module also attempts to filter out the non-user-defined functions (i.e., non-UDFs) in the code to reduce potential noise from non-UDF code when analyzing the characteristics of user code.

3.1.1 Normalizing code text

The pre-processing module of $MGAP^3$ normalizes the assembly code of each malware sample into pseudo-assembly text, as illustrated in Fig. 2, through the following key process:

1) **Normalize sample-size, function-size & block-size:** For each malware sample, the module establishes predefined constants (denoted as S, F, and B, respectively) for sample size (i.e., the total number of functions in each sample file), function size (i.e., the total number of blocks in each function), and block size (i.e., the total number of instructions in each basic block) by padding the missing ones with "PAD" flags. The constant values for S, B, and F can be determined using either a "majority strategy" or a "full strategy." The full strategy sets S, B, and F to encompass all the malware samples, blocks, and functions in the dataset, respectively, while the majority strategy sets S, B, and F to represent the majority of them, respectively. Compared to the full strategy, the majority strategy yields shorter input text, resulting in faster model training and reduced computational resource consumption. Thus, in our experiments, we adopted the majority strategy.

   Taking the dataset of AMG25 (detailed in Section 4.1) used in our experiment as an example, we employed the majority strategy and set S, F, and B to 127, 15, and 8, respectively. As illustrated in Appendix A, due to the commonly observed long-tail distribution, 76% of the functions, 73% of the blocks, and 72% of the instructions of the original malware samples in the dataset can be covered by this normalization strategy. Additionally, we conducted an extra experiment to evaluate the impact of S, F, and B values on our model and report the results in Appendix D. As Appendix D shows, higher values