ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) | 979-8-3503-4485-1/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICASSP48485.2024.10446565

DISCOVERING MALICIOUS SIGNATURES IN SOFTWARE FROM STRUCTURAL INTERACTIONS

Chenzhong Yin[1,*] *Hantang Zhang*[1,*] *Mingxi Cheng*[1]
Xiongye Xiao[1] *Xinghe Chen*[1] *Xin Ren*[1] *Paul Bogdan*[1]

[1]University of Southern California, Los Angeles, CA, USA

ABSTRACT

Malware represents a significant security concern in today's digital landscape, as it can destroy or disable operating systems, steal sensitive user information, and occupy valuable disk space. However, current malware detection methods, such as static-based and dynamic-based approaches, struggle to identify newly developed ("zero-day") malware and are limited by customized virtual machine (VM) environments. To overcome these limitations, we propose a novel malware detection approach that leverages deep learning, mathematical techniques, and network science. Our approach focuses on static and dynamic analysis and utilizes the Low-Level Virtual Machine (LLVM) to profile applications within a complex network. The generated network topologies are input into the GraphSAGE architecture to efficiently distinguish between benign and malicious software applications, with the operation names denoted as node features. Importantly, the GraphSAGE models analyze the network's topological geometry to make predictions, enabling them to detect state-of-the-art malware and prevent potential damage during execution in a VM. To evaluate our approach, we conduct a study on a dataset comprising source code from 24,376 applications, specifically written in C/C++, sourced directly from widely-recognized malware and various types of benign software. The results show a high detection performance with an Area Under the Receiver Operating Characteristic Curve (AUROC) of 99.85%. Our approach marks a substantial improvement in malware detection, providing a notably more accurate and efficient solution when compared to current state-of-the-art malware detection methods. The code is released at https://github.com/HantangZhang/MGN.

***Index Terms*—** Malware detection, Graph neural network, Complex network

1. INTRODUCTION

Malware refers to malicious software designed to cause damage to the Internet, computers and cyber-physical systems. Malware could intrude into PC, collect user's personal information and sensitive data, and gain administrative privileges on a host, trigger havoc on PC's operating system, and lead to the loss of billions of dollars. For instance, malware-based cyber-attacks can target industrial control systems (ICS) to slow down industrial processes or destroy critical components, causing significant financial damages, safety, and life-threatening events [1].

To prevent catastrophic events, researchers focus on malware detection using static and dynamic analyses. Static analysis involves scanning software binary byte-streams to generate signatures, like printable strings, n-grams, and instructions [2]. Kim et al. proposed a multimodal deep learning scheme to detect Android mobile malware by analyzing various factors such as strings, method APIs, permissions, components, and environment [3]. While static analysis is accurate for specific malware types, it struggles with "zero-day" malware, where signatures change due to updates [4].

Dynamic analysis monitors and controls malware during execution. Deep learning models are now applied in this approach. Classic convolutional- (CNN) and recurrent- neural network (RNN) can learn features from the sequential data extracted from software [5] and image data [6]. For instance, Zhang et al. [7] designed a CNN-LSTM model to analyze features from each API call for detecting malicious files. Despite overcoming static analysis limitations, deep learning based dynamic approaches' inherent black-box nature hinders the interpretability of its feature attribution [8, 9, 10].

This paper introduces the Malware Graph Network (MGN), a robust deep learning framework addressing challenges in static and dynamic analysis for malware detection. MGN involves adopting a Low-Level Virtual Machine (LLVM) intermediate representation (IR) compiler [11] to transform suspected malware software into a complex weighted network, where nodes represent instructions, and edges indicate data and control dependencies among LLVM instructions. The weights associated with the network can represent data sizes or latency values. This compiler approach eliminates the need to execute suspected applications on a virtual machine. This complex network representation allows capturing the spatiotemporal characteristics of software and interpreting the differences between benign and malicious files by revealing the intrinsic correlations between APIs and the software's instructions. To process these intricate network structures

*Both authors contributed equally to this work

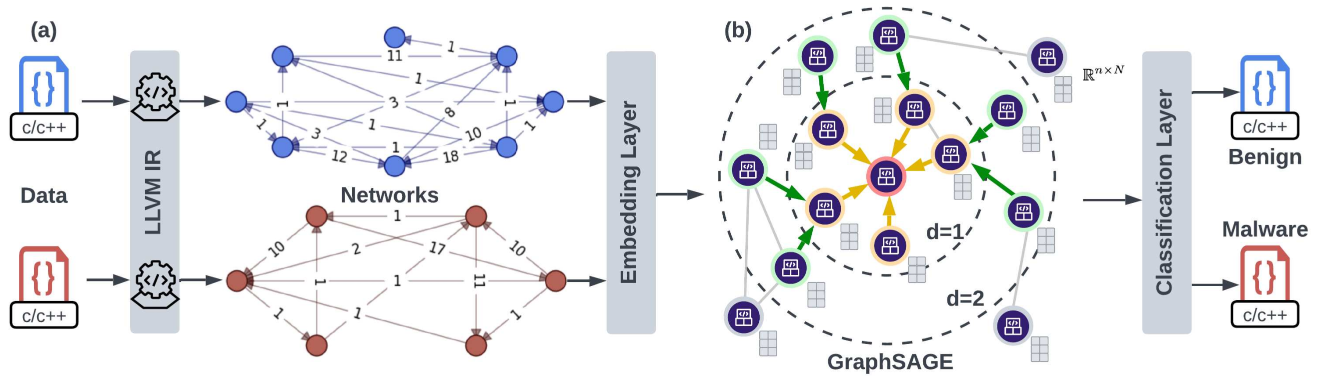

Fig. 1: Overview of the Proposed Method: (a) C/C++ files are transformed into LLVM IR and converted into a weighted complex network by capturing operations from the source code. The generated weighted networks are passed into an embedding layer to compute node features, resulting in an $n \times N$ matrix representation. (b) Leveraging these matrices, the GraphSAGE architecture extracts information from the neighbors of each node. Lastly, the classification layer predicts whether the file is malicious or benign.

that have been extracted, we incorporate a GNN-based deep learning architecture called GraphSAGE into MGN for malware classification. This approach surpasses state-of-the-art baselines, such as N-gram Analysis [12], Grayscale Image Analysis [13, 14], Operation Code Analysis [15], and Byte File Analysis [16, 7].

2. METHOD

To create a robust and interpretable malware detection strategy capable of effectively handling malware evolution and obfuscation, we present MGN, illustrated in Figure 1. MGN comprises two primary components: (1) utilizing a compiler approach built on LLVM to compile source code into a sophisticated network representation and (2) feeding the constructed networks into a GNN-based deep learning architecture for the purpose of malware detection. This two-part framework forms the foundation of our proposed approach for enhanced malware detection.

2.1. Compiler Analysis and Network Modeling

In this section, we leverage code compilation tools to transform source code into a graph representation. Specifically, we transform input applications into a corresponding dependency graph [17] through LLVM compiler and transparent by introducing IR as a common model for analysis, transformation, and synthesis. The LLVM is a compiler framework which makes program analysis lifelong. The resulting graph encompasses the structure of the graph itself, which includes nodes and the connections between them, as well as node features. For each graph, its node features are derived from the operation names present in the code, such as “store”, “getelementptr”, and “load”.

We first run the applications with representative inputs to collect dynamic LLVM IR traces to resolve dynamic memory dependencies. Next, we analyze the traces to figure out whether source registers of the current instruction depend on destination registers of the previous instructions. If such dependencies exist, we add edges between these two nodes to represent dependencies. Then, these instructions are profiled to get the precise data sizes as edge weights in the graph. For example, a part of dynamic traces has been shown as follows: $\%3$ = sub $\%1$, $\%2$; $\%5$ = sub $\%3$, $\%4$. The second instruction has the register $\%3$, which depends on the destination register $\%3$ of the first instruction.

Hence, we find a dependency between the last two instructions and an edge is inserted between these two nodes. After following these compiled steps, a suspected application is profiled into a complex weighted network that will be analyzed in the following section to predict whether this application is benign or malicious. Figure 1 shows the steps of dynamically profiling the applications into graphs.

After the transformation of the suspected files into graph representations, we encode the node features, represented by operation names, into one-hot vectors. Each graph’s node features can be denoted by a matrix of dimensions $n \times N$, where n signifies the number of distinct operation names present in a given graph and N denotes a constant that represents the union of operation names across the entire dataset. Moreover, each graph is assigned a label indicating whether it represents malware or benign software.

2.2. GraphSAGE Neural Network Modeling

In this study, we choose GraphSAGE to analyze the preprocessed features represented as complex networks. GraphSAGE can efficiently analyze large graph structures, where many software source codes often consist of numerous nodes [18]. Furthermore, GraphSAGE employs inductive learning and ensures excellent generalizability, which allows it to generate embeddings for previously unseen nodes. This character can be highly effective for classification tasks. Instead of relying on manually crafted features to describe malware, this property becomes invaluable in malware detection, given the frequent emergence of new malware variants and families.

Through an in-depth examination of GraphSAGE's structure and the graphs representing malware and benign software, we have devised our deep learning architecture. It comprises of the following components: an embedding layer, 6 GraphSAGE layers, 1 global pooling layer, and 1 output layer. In the input layer, we incorporate node features and edge indices, where the node features denote operation name for the instruction and the edge indices represent edge lists. The GraphSAGE employs the mean aggregator as its initial step, which calculates the mean of node features within the neighborhood. Subsequently, the number of neighbors sampled in each layer depends on the average degree of nodes.

While GraphSAGE layer achieved the most exceptional performance in graph neural network model, the number of SAGE layers and the choice of activation functions have a significant impact on the final model's performance. This was evident in our ablation experiments.

3. EVALUATION

Following the same training and testing strategy with all the baselines, our model is trained and evaluated on 24,376 applications (12,815 malware and 11,561 benign software). All these applications are source code gathered from real-world programs. For malware representation, we select a diverse array of types, including Spyware, Botnet, Trojan, among others. As for benign software, we choose programs from gaming, system development, application software, mobile apps, and artificial intelligence to guarantee that our dataset comprehensively includes various domains. For every software, we identify their main execution functions as data points. These are then compiled using our toolset and transformed into a graph, serving as individual data entries. The entire dataset is randomly split into 80% as training set and the remaining 20% is testing set. The model performance is assessed via the AUROC and accuracy (ACC).

3.1. Ablation study

To gain a comprehensive understanding of the significance of different components in our malware detection model and to determine the most effective architecture, we performed an ablation study, varying hyperparameters and settings. Table 1 presents a summary of our findings. In the table, "EL" and "LReLU" refer to the embedding layer and Leaky-ReLU activation function, respectively. The presence of a ✓ indicates that the embedding layer or Leaky-ReLU was utilized, while a ✗ denotes their absence (for activation function, we use ReLU instead). The first column represents the different numbers of SAGE layers chosen for the ablation study.

Table 1: Comparison of malware detection results using different hyperparameters and settings. The best results are highlighted in red.

SAGE	EL	LRelu	ACC	AUROC	F1-score
4layers	✗	✗	96.29%	99.38%	96.81%
	✓	✓	97.45%	99.72%	97.76%
	✓	✗	97.17%	99.57%	97.54%
6layers	✗	✗	95.79%	99.33%	96.08%
	✓	✓	98.55%	99.85%	98.72%
	✓	✗	97.45%	99.73%	97.78%
8layers	✗	✗	96.51%	99.61%	96.96%
	✓	✓	98.18%	99.79%	98.40%
	✓	✗	97.45%	99.59%	97.78%
10layers	✗	✗	95.65%	99.37%	95.95%
	✓	✓	96.03%	99.46%	96.30%
	✓	✗	95.82%	99.40%	96.11%

In this study, we introduce a fully-connected embedding layer before the GNN architecture, which plays an essential role in capturing initial feature representations. The absence of this embedding layer results in a performance drop of approximately 2.76%. Furthermore, we incorporate 6 SAGE layers, each connected to a hidden layer with 128 hidden units. We choose Leaky-ReLU as the activation function for each SAGE layer.

3.2. Experimental results

The comparison results for malware detection are presented in Table 2 and Fig. 2. In this section, we select state-of-the-art deep learning-based malware detection models as our baselines, as referenced in previous works [13, 19, 20, 7, 21, 16]. All these models are trained and tested on our malware dataset to ensure a fair comparison.

As indicated in Table 2, our MGN demonstrates remarkable performance in classifying malware and outperforms all other models in terms of both ACC and AUROC. To provide a visual representation of these comparisons, Fig. 2 panels (a) and (b) display quantitative results for ACC and AUROC, respectively. Panels (c) and (d) respectively present the learning curve comparisons between MGN and selected baselines (specifically, ARI-LSTM, EE-DNN, and H-CNN, which utilize N-gram, Bytes, and Opcode as features, respectively). Hence, our findings suggest that network-based features used in MGN can significantly enhance the performance of distinguishing between malware and benign software within the context of deep learning. For comprehensive information, please refer to Table 2.

Table 3 presents a detailed breakdown of various malware types, along with their respective performance metrics obtained through our method and ARI-LSTM [19]. We choose to include ARI-LSTM in this table because of its notable accuracy, as well as the similarity of its features to our node-based features.

Table 2: Comparison of malware detection results (ACC and AUROC) across different models. The best results are highlighted in red.

Baselines	Features	ACC	AUROC
CNN [13]	Image	93.09%	98.31%
ARI-LSTM [19]	N-gram	98.28%	99.5%
Bi-GRU-CNN [16]	Bytes	96.36%	99.43%
EE-DNN [20]	Bytes	97.45%	99.72%
SA-CNN [7]	Opcode	97.38%	99.66%
H-CNN [21]	Opcode	98.18%	99.74%
MGN	Networks	98.55%	99.85%

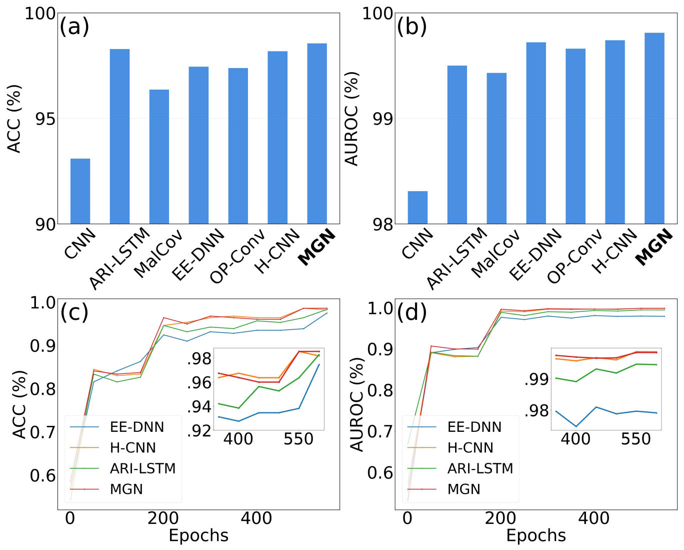

Fig. 2: Performance comparison of MGN against baselines on the test set, showing (a) ACC and (b) AUROC; (c) ACC and (d) AUROC comparisons of MGN against baselines at different epochs.

Table 3: Classification accuracy comparison between our model (MGN) and the state-of-the-art model (ARI-LSTM) across various malware types. The best results are highlighted in red.

Malware	Samples	Text Samples	MGN	ARI-LSTM
Spyware	4757	950	98.32%	98.11%
Botnet	1548	300	98.67%	97.00%
Trojan	4645	900	99.11%	98.89%
Rootkit	3048	600	98.33%	97.67%
Trojan-Backdoor	3097	600	98.50%	97.83%
Worm	1548	300	100.00%	99.00%
Ransomware	900	180	100.00%	100.00%
Injection	900	180	98.89%	98.33%
Mixed	3933	750	100.00%	99.60%

3.3. Interpretability Analysis

Figure 3 provides a clear two-dimensional t-SNE visualization, illustrating the clustering behavior of network topological features which includes the number of nodes and edges, average closeness-, average degree-, and average betweenness-centrality. These features are derived from the network representations of source files. We randomly selected around 900 instances from both the benign and malicious network classes for visual clarity. This visualization effectively highlights the capability of our proposed methods in discerning the inherent topological distinctions between benign and malicious codes. Consequently, it enhances the interpretability of our approach.

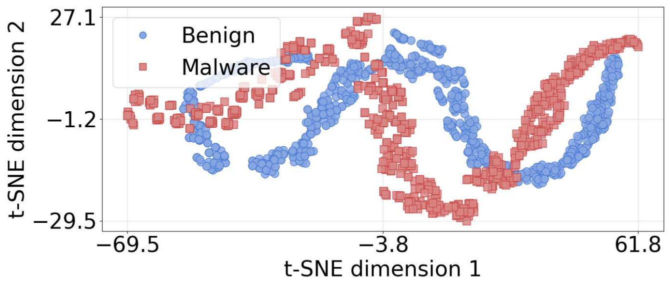

Fig. 3: Two-dimensional t-SNE visualization of network topological features. Blue circles and red squares represent benign and malicious networks, respectively.

4. CONCLUSION

In this paper, we propose an innovative deep learning framework known as MGN, designed for malware detection. MGN utilizes a graph-based representation and harnesses the power of the LLVM compiler to capture intricate software dependencies, resulting in the creation of a complex network. This network is then subjected to classification using a GNN architecture, significantly improving the precision of distinguishing between benign and malicious programs. Through careful design of the model architecture, MGN demonstrates superior performance compared to state-of-the-art baselines, achieving higher accuracy and AUROC measurements. It is important to highlight that MGN's ability to transform software into the network's topology enhances its interpretability, setting it apart from other baseline approaches.

5. ACKNOWLEDGEMENT

The authors acknowledge the support by the National Science Foundation (NSF) under the Career Award CPS-1453860, CCF-1837131, MCB-1936775, CNS-1932620 and award No. 2243104, Center for Complex Particle Systems (COMPASS), U.S. Army Research Office (ARO) under Grant No. W911NF-23-1-0111, DARPA Young Faculty Award and DARPA Director Award under Grant Number N66001-17-1-4044, and a Northrop Grumman grant. The views, opinions, and/or findings in this article are those of the authors and should not be interpreted as official views or policies of the Department of Defense or the National Science Foundation.

6. REFERENCES

[1] Scott Steele Buchanan, "Cyber-attacks to industrial control systems since stuxnet: A systematic review," 2022.

[2] Mamoru Mimura and Ryo Ito, "Applying nlp techniques to malware detection in a practical environment," *Intl. Journal of Information Security*, vol. 21, no. 2, 2022.

[3] TaeGuen Kim, BooJoong Kang, Mina Rho, Sakir Sezer, and Eul Gyu Im, "A multimodal deep learning method for android malware detection using various features," *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 3, 2018.

[4] Leyla Bilge and Tudor Dumitraş, "Before we knew it: an empirical study of zero-day attacks in the real world," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012.

[5] Weizhong Qiang, Lin Yang, and Hai Jin, "Efficient and robust malware detection based on control flow traces using deep neural networks," *Computers & Security*, 2022.

[6] Charlotte Pelletier, Geoffrey I Webb, and François Petitjean, "Deep learning for the classification of sentinel-2 image time series," in *IGARSS 2019-2019 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, 2019, pp. 461–464.

[7] Bin Zhang, Wentao Xiao, Xi Xiao, Arun Kumar Sangaiah, Weizhe Zhang, and Jiajia Zhang, "Ransomware classification using patch-based cnn and self-attention network on embedded n-grams of opcodes," *Future Generation Computer Systems*, vol. 110, 2020.

[8] Chenzhong Yin, Phoebe Imms, Mingxi Cheng, Anar Amgalan, Nahian F Chowdhury, Roy J Massett, Nikhil N Chaudhari, Xinghe Chen, Paul M Thompson, Paul Bogdan, et al., "Anatomically interpretable deep learning of brain age captures domain-specific cognitive impairment," *Proceedings of the National Academy of Sciences*, vol. 120, no. 2, pp. e2214634120, 2023.

[9] Chenzhong Yin, Mihai Udrescu, Gaurav Gupta, Mingxi Cheng, Andrei Lihu, Lucretia Udrescu, Paul Bogdan, David M Mannino, and Stefan Mihaicuta, "Fractional dynamics foster deep learning of copd stage prediction," *Advanced Science*, vol. 10, no. 12, pp. 2203485, 2023.

[10] Xiongye Xiao, Defu Cao, Ruochen Yang, Gaurav Gupta, Gengshuo Liu, Chenzhong Yin, Radu Balan, and Paul Bogdan, "Coupled multiwavelet operator learning for coupled differential equations," in *The Eleventh International Conference on Learning Representations*, 2022.

[11] Chris Lattner and Vikram Adve, "Llvm a compilation framework for lifelong program analysis transformation," in *Proceedings of the international symposium on Code generation and optimization feedback-directed and runtime optimization*. IEEE Computer Society, 2004, p. 75.

[12] Enmin Zhu, Jianjie Zhang, Jijie Yan, Kongyang Chen, and Chongzhi Gao, "N-gram malgan: Evading machine learning detection via feature n-gram," *Digital communications and networks*, vol. 8, no. 4, pp. 485–491, 2022.

[13] Daniel Gibert, Carles Mateu, Jordi Planes, and Ramon Vicens, "Using convolutional neural networks for classification of malware represented as images," *Journal of Computer Virology and Hacking Techniques*, 2019.

[14] Iman Almomani, Aala Alkhayer, and Walid El-Shafai, "An automated vision-based deep learning model for efficient detection of android malware attacks," *IEEE Access*, vol. 10, pp. 2700–2720, 2022.

[15] Junwei Tang, Ruixuan Li, Yu Jiang, Xiwu Gu, and Yuhua Li, "Android malware obfuscation variants detection method based on multi-granularity opcode features," *Future Generation Computer Systems*, vol. 129, pp. 141–151, 2022.

[16] Rajasekhar Chaganti, Vinayakumar Ravi, and Tuan D Pham, "Deep learning based cross architecture internet of things malware detection and classification," *Computers & Security*, vol. 120, pp. 102779, 2022.

[17] Yao Xiao, Yuankun Xue, Shahin Nazarian, and Paul Bogdan, "A load balancing inspired optimization framework for exascale multicore systems a complex networks approach," in *Proceedings of the 36th Intl. Conf. on Computer-Aided Design*, 2017, pp. 217–224.

[18] William L. Hamilton, Rex Ying, and Jure Leskovec, "Inductive representation learning on large graphs," in *NIPS*, 2017.

[19] Rakshit Agrawal, Jack W Stokes, Karthik Selvaraj, and Mady Marinescu, "Attention in recurrent neural networks for ransomware detection," in *ICASSP 2019-2019 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2019.

[20] Daniel Gibert, Carles Mateu, and Jordi Planes, "An end-to-end deep learning architecture for classification of malware's binary content," in *International Conference on Artificial Neural Networks*. Springer, 2018, pp. 383–391.

[21] Daniel Gibert, Carles Mateu, and Jordi Planes, "A hierarchical convolutional neural network for malware classification," in *2019 International joint conference on neural networks (IJCNN)*. IEEE, 2019, pp. 1–8.