Tractable Executable Binary Provenance Signalling through Vision Transformers

Mohammad Nauman
Computer Science Department, Effat College of Engineering
Effat University
Jeddah, KSA
mnauman@effatuniversity.edu.sa

***Abstract*—Provenance signaling involves tracing the source information of digital artifacts. It is a valuable intermediate output that greatly facilitates upstream tasks, including but not limited to malware analysis. Existing approaches to provenance signaling either rely on fully manual analysis or machine learning-based models that heavily depend on manually curated input features. This curation process requires the involvement of human experts, which is not only time-consuming but also infeasible on a large scale. In this paper, we present a novel model for provenance signaling that takes raw binaries as input and provides provenance signals with high efficacy. Our model is based on the state-of-the-art vision transformer architecture. We create a novel pipeline of efficiently encoding any binary into 2D sequences, capturing large-scale spatial relations hidden among binary opcodes. This allows our model to extract meaningful information about provenance without requiring the involvement of a human expert. Therefore, our work produces high-accuracy results and provides insights into the learning process, thus making the results more explainable.**

I. Introduction

Provenance, a term widely recognized and embraced in various fields, plays a vital role in the process of identifying and tracing the *source information* of any given piece of data. [1] By delving into the intricate details of provenance, one can uncover essential insights into the origin, authenticity, and reliability of information. This invaluable process not only ensures transparency and accountability but also enables experts to identify and uncover extremely useful artifacts about the target. Provenance for executable binaries – referred to from hereonin as *binaries* – allows us to uncover some very important information about the binary including artifact useful in malware analysis, among others.

However, the task of extracting provenance information from binaries that span different architectures and derive from the same source code, compiled using various compilers, is undeniably a formidable challenge. This intricate process requires not only expertise but also advanced tools and methodologies to ensure accurate results.In order to tackle this complex challenge, it is crucial to rely on cutting-edge techniques that can effectively navigate through the intricate web of differences caused by architecture variations and compiler nuances. [2]

979-8-3503-9356-9/24/$31.00 ©2024 IEEE

This process is typically carried out with the help of sophisticated analysis tools, and diving deep into the intricacies of binary code variations, extracting meaningful insights that would otherwise remain hidden. These tools enable us to compare binaries across different architectures and compilers, identifying common elements and distinguishing them from unique characteristics. This not only streamlines development processes but also strengthens security measures by detecting potential vulnerabilities or malicious components lurking within the codebase.

Owing to the latest achievements of machine learning research in various fields, the realm of provenance identification has also begun to utilize modern approaches for a highly effective and comprehensive methodology of solving these issues. These approaches primarily focus on meticulous *manual* extraction of essential attributes from assembly instructions, which are then seamlessly integrated into *detailed* machine learning models. This integration ensures a seamless flow of information, empowering these models to accurately determine the origin and history of an item or artifact in binaries. By combining the precision of manual extraction with the power of machine learning, contemporary methods excel in providing reliable and detailed insights into the provenance of various objects. Typically, three types of manual attributes are targetted by a human expert in this case:

1) Syntactic attributes: refer to the distinct program characteristics observed within its code such as instruction patterns, function signatures, and the frequency of opcodes.

2) Structural attributes: enable effective encapsulation of these flows by utilizing concepts such as the *Control Flow Graph (CFG)* and the *Function Call Graph (FCG)*.

3) Semantic attributes: go beyond surface-level analysis and dive into the intricacies of code including graph-based composite features and machine learning-based embedding representations. [1]

However, there are two major issues with all these approaches: they rely heavily on artifacts *manually* extracted using tools like disassemblers. The process of disassembling binaries may initially yield output that is straightforward and clear. However, extracting meaningful syntactic, structural, and semantic attributes from this output can be a daunting and labor-intensive endeavor. It requires a significant amount of expert time, effort, and expertise to unravel the intricate details hidden within binary code.

The primary issue is that binary files lack the essential high-level abstractions that are necessary for providing meaningful features. These features include clear function boundaries, which are crucial for efficient coding and software development. As a result, it requires considerable effort on part of the human expert to extract these features.

Moreover, the features currently employed are carefully selected and processed using specialized knowledge that is customized for specific compilation methods and CPU architectures. [3]

Due to the inherent limitations, it may be challenging for these features to seamlessly transition into various architectural realms. This calls for the painstaking process of reimagining and recreating feature sets specifically tailored for the new environment.

Finally, modern compilers also provide the ability to perform optimization at different levels. So, output produced as a result of one optimization level is usually quite different from output produced by running the compiler at a different optimization level. Provenance identification at such levels thus becomes an extremely daunting task.

Some recent works have tried to leverage end-to-end machine learning to address the issue of human expertise requirement. [4] However, these end-to-end machine learning models have failed to provide a scalable architecture due to the nature of the models in use. For instance, *Long Short Term Memory (LSTM)*-based models are notoriously difficult to train and are usually unable to identify spatially displaced artifacts due to the vanishing gradient problem. Some recent works have tried to leverage attention-based models such as BERT. However, while these models work well on some binaries, their outputs are completely intractable and improvement is therefore quite difficult.

In order to address this issue, we utilize the concept of mapping 1D binary sequences to 2D bitmaps and then utilizing a modified version of the seminal work by VisionTransformers to predict provenance signals. Our contributions in this paper are two-fold:

1. We propose a novel approach towards provenance signalling using a modified veriant of the Vision Transformer model that is not susceptible to limitations faced by other deep learning models such as Convolutional Neural Networks (CNNs).

2. We Study the approach on a real-world, large scale dataset and demonstrate the efficacy of our approach in not only making predictions about signalling but also that this approach is highly tractable and leads to explainable results.

In the rest of the paper, we first provide a brief review of the background work (cf Section II) and then move on to the details of our contribution (cf. Section III) and finally to our results in Section IV.

II. Background

A. Vision Transformers

Deep learning, a branch of machine learning, has proven to be effective in diverse domains, such as provenance singalling and malware analysis. One of the primary challenges associated with conventional methods of malware analysis is their dependence on manually designed characteristics. [5] These characteristics are usually crafted by human experts and demand substantial time and expertise for development. As the number of new types of software as well as malware strains continues to rise each day, experts are finding it progressively arduous to keep up with the pace.

Deep learning tackles this problem by employing end-to-end learning, wherein the network acquires the ability to directly extract characteristics from the unprocessed data. Consequently, the network becomes capable of recognizing malicious software samples without relying on manually engineered features, which can be both time-consuming and labor-intensive to develop.

Traditionally, machine learning relied on statistical models to understand the relationship between input features and outputs. However, this approach was challenging and not easily scalable. The introduction of fully connected deep neural networks brought about a revolution by allowing for the automatic extraction of features from large datasets. Despite their effectiveness with image-based datasets, CNNs have two significant limitations. Firstly, the features learned by CNNs, such as edges, lines, curves, and regions of high and low contrast, can be visualized and interpreted in the context of image-based data. However, when applied to other types of problems like provenance signal detection, these features become difficult to interpret or understand.

CNNs have a significant limitation related to their architecture. They are designed to be rotation- and translation-invariant, meaning that a feature present in one location is considered equally valuable in any other location. While this enhances scalability and computational efficiency, it also presents drawbacks. For example, if an image shows scattered tires, doors, and a car hood, a CNN might classify it as a "vehicle" due to these features. However, this rotation and translation invariance makes it difficult for the network to identify specific image features since it cannot distinguish between different orientations or positions.

Transformers [6] are cutting-edge deep learning models utilized for attention-based processing of real-world data in both generative and discriminative branches of machine learning. These models serve as the basis for renowned deep learning models Vision transformers, introduced by Dosovitskiy et al. [7], take the concept of transformers to the next level, allowing for image modeling using a mechanism similar to traditional transformers. While comprehending the mathematics behind vision transformers necessitates a more detailed examination, we will present the main concepts of the model and their relevance to our work to aid readers in understanding our contributions.

Transformers address the limitations of CNNs by integrating attention mechanisms into the hidden layers of deep neural networks (Vaswani, 2017). This enables the model to prioritize specific aspects of the input, instead of treating all parts equally like in CNNs. As a result, transformers provide more interpretable predictions and achieve better performance in tasks like malware analysis that require understanding the specific features leading to a classification. Moreover, transformers are

more proficient in handling sequential data, such as sequences of instructions.

The attention mechanism is determined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1)$$

where Q is the feature of interest in the image, K is the mask collection, the set of all possible keys is denoted by d_k and that of all values is V. The deep learning model thus computes the attention based on Q, K and V from the given dataset. Owing to the length limitation of the paper, we refer the reader to the original Vision Transformer paper [7] for details regarding the mechanism of this *multi-head attention model*.

Our model incorporates this multi-head attention mechanism that serves to enhance both its efficiency and interpretability. We return to a discussion of this later in the paper after we discuss the issues related to provenance signalling itself.

B. Variation in Provenance Signals

The impact that compilers and their optimization levels have on binary code variance should not be underestimated. Due to the varying implementations and optimization techniques used by different compilers, the resulting binary code can differ significantly. It is crucial to understand these differences as they can greatly affect the performance and efficiency of your software.

To understand the impact of compiler and optimization level differences, one effective method is to compare the binary code generated by different compilers for identical source code. For instance, let's take a simple C program and analyze the binary code outputted by both GCC and Clang. This will provide valuable insights into how each compiler handles optimization and ultimately impact the performance of your code.

```
int main() {
  int a = 1;
  int b = 2;
  int c = a + b;
  return c;
}
```

By utilizing the latest versions of *Gnu C Compiler (GCC)* and *Clang* with the default optimization level (-O2), yields the following binary code output.

```
48 83 c4 08
48 83 c4 10
48 8d 5c 24 08
c3
```

This hexadecimal representation of binary sequences denotes the opcode (first byte) followed by the operands. Each line thus forms one machine instruction. For the same instructions, Clang produces the following output.

```
55
48 89 ec
48 8b 4d 08
40 03 4d 08
48 89 c2
5d
```

The binary code produced by these two compilers is noticeably different, which can be clearly observed. This disparity arises due to the usage of distinct implementations and advanced optimization techniques by each compiler.

The impact of optimization levels can be observed by comparing the binary code generated from the same source code, but with different optimization levels. To illustrate, let's compare the binary code produced by utilizing *-O0* and *-O2* optimization levels for the aforementioned program. GCC with *-O0* for instance produces the following:

```
55
48 89 e5
41 57
48 83 ec 28
48 8b 45 08
48 05 45 08
48 89 c5
5d
```

Compare this to the first output in this section which was produced with the default option of *-O2*. The binary code generated with the *-O2* optimization level exhibits a significant reduction in length compared to the binary code produced at the *-O0* optimization level. This disparity arises from the compiler's enhanced ability to execute more extensive optimizations when operating at a higher optimization level.

When we compare cases 2 and 4, specifically using different compilers under the O2 optimization level, it is notable that they have identical assembly code and machine code. However, the distinction arises in the storage addresses, or offsets. The reason for this discrepancy lies in the fact that different compilers employ diverse memory allocation strategies.

When the compiler operates at the *-O1* optimization level, it has the capability to execute various optimizations, including but not limited to constant folding, elimination of dead code, and prediction of branching. At the *-O2* optimization level, the compiler has the capability to implement more aggressive optimization techniques like loop unrolling and instruction reordering. At the *-O3* optimization level, the compiler has the capability to execute even more assertive optimizations, including but not limited to function inlining and tail call elimination.

However, it's important to recognize the significance of enhancing the optimization level on provenance signalling. It is imperative to understand that such enhancements can potentially introduce complexities in the analysis and debugging processes of binary code. This arises from the fact that the compiler may execute intricate transformations on the code, rendering it arduous to comprehend or reverse engineer for diagnostic purposes. This becomes even more pronounced if human expertise is required to extract meaningful syntactic, structural and semantic attributes from binaries.

III. Proposed Architecture: Signalling Provenance for Binaries

In this section, we present the details of our suggested framework for our model, along with the rationale behind our research decisions. We discuss the network design, embedding

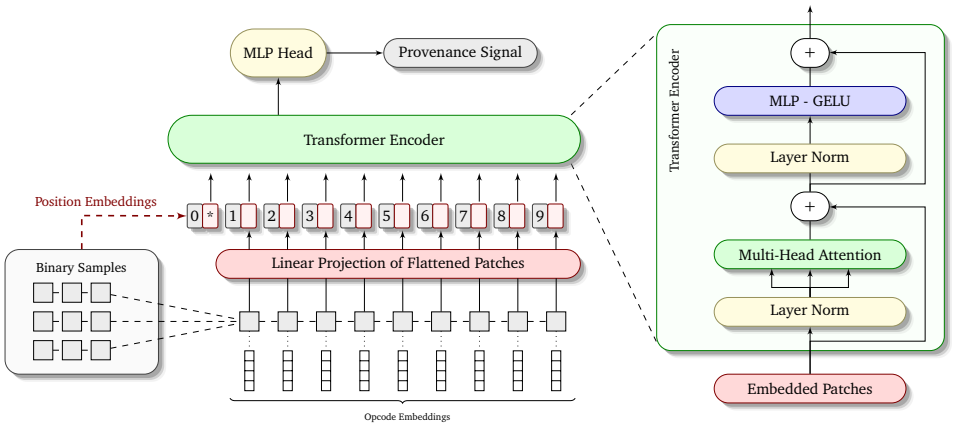


Fig. 1: Proposed Architecture of the Vision Transformer Model to Signal Provenance

choices, and hyperparameter selections, all aimed at enabling readers to reproduce our work and results autonomously.

Our proposed approach seeks to leverage the capabilities of vision transformers in order to examine provenance signals in binaries. The primary goal of this study is to create a model that not only achieves superior accuracy, but also offers results that can be easily interpreted. In other words, the model should not only label an executable file with accurate provenance signals, but also identify the specific portion of the opcode sequence that influenced the final determination. This aspect of interpretability represents a significant breakthrough in the realm of malware analysis, as it has not been extensively achieved before.

We have implemented the network architecture in a highly modular manner to accomplish our goal. The complete layout of the network can be seen in Figure 1. Below, we describe the three aspects of this model in detail.

A. Binary to 2D Mapped Sequence Transforms

In our methodology for examining malware, we employ static analysis techniques by utilizing the opcode sequences found in executable files as input for our advanced machine learning system. Based on our extensive expertise in binary analysis, we have devised a process for transforming opcode sequences into compact vector embeddings with reduced dimensions.

To accomplish this, we implemented a methodical and automated approach that involved extracting and manipulating executable opcodes. Initially, we obtained the raw bytecode from our dataset. Using a disassembler, we then parsed the bytecode to extract the essential sequence information of opcodes, while disregarding any additional metadata that could obscure the underlying behavioral patterns. These specific opcodes, which represent the programmatic actions within the applications, were subsequently transformed into encoded vector embeddings through a learning process.

To ensure a fair comparison between our model and previous methods for detecting malware, we have chosen to utilize the same embeddings that were used in our earlier work. It is important to note that previous techniques limited the number of dimensions in opcode embeddings in order to reduce the time required for training and predicting with their models.

To achieve optimal performance in detecting malware, our WAMPA model incorporates vision transformers as its core architecture. By using this advanced model core, we are able to capture higher-dimensional embeddings, which allows for a deeper understanding of the opcode sequences. This is a critical feature in provenance analysis as it leads to improved accuracy, reliability, and interpretability in identifying malicious software.

Furthermore, our decision to utilize vision transformers as the fundamental framework distinguishes it from conventional recurrent neural networks. Unlike RNNs, vision transformers do not depend on backpropagation through time (BPTT) and instead process the complete input sequence as a cohesive unit, rather than individually. This methodology allows the model to take into account the entire context of the sequence and generate more precise forecasts.

In spite of the advantages offered by vision transformers in malware analysis, there is a potential downside related to the loss of positional information within the opcode sequences. To overcome this limitation, we have integrated position encodings into the input embeddings. By doing so, the model gains a better understanding of the relative position of each opcode, thus enabling more precise predictions based on the overall sequence context. This combination of vision transformers and the reintroduction of positional information enhances the efficiency and effectiveness of our malware analysis process through our model.

B. Embedding Generation

After preparing the input as embeddings, we input them into our multi-head vision transformer model. This particular model is our inhouse variant of the transformers and it can effectively identify binary behavior in the code, even when it spans a long range. For instance, provenance signals may be scattered across the opcode sequence in a spatial manner. We represent the opcodes as a 2D plane of 1D opcodes, with a width of length Ω, which is a hyperparameter chosen based on empirical evidence. We experimented with different values for this hyperparameter to observe its impact on detecting malicious behavior. Narrower widths can detect patterns when the spatial separation between two sections of malicious code is

small, whereas wider widths are necessary to identify patterns when the separation is larger.

To assess the effectiveness of our model, we adjusted the variable Ω and carried out experiments on various binaries within established datasets. These experiments revealed that different values of this parameter yield favorable results for different malware families. This discrepancy is likely due to the distinct latent behavior patterns exhibited by each family, which become apparent at different values of Ω. Section IV provides an overview of our preliminary findings on this matter. However, it is crucial to conduct a thorough analysis in this research direction. Exploring this avenue further holds promise for future studies, as it can enhance our comprehension of the underlying behavior patterns and contribute to more precise and efficient detection methods.

C. Provenance Classification

The embedded patches that have been normalized are subsequently fed into the multi-head attention module, which is tasked with the actual process of learning attention. We have conducted various experiments, exploring different quantities and dimensions of multi-heads, in order to enhance the performance of the model. Details of our findings can be found in Section IV. These findings unequivocally demonstrate the effectiveness of our proposed architecture in capturing both the temporal and spatial relationships among the various opcodes within the input sequence. Furthermore, our architecture is adept at handling diverse input distributions, resulting in detection methods that are more precise and efficient.

After incorporating the multi-head attention layers, we introduce an extra normalization layer in the shape of a secondary level of Layer Norm. Subsequently, we execute a 2-layer Multi-Layer Perceptron (MLP) employing the Gaussian Error Linear Unit (GeLU) activation function. The GeLU function is precisely defined as follows:

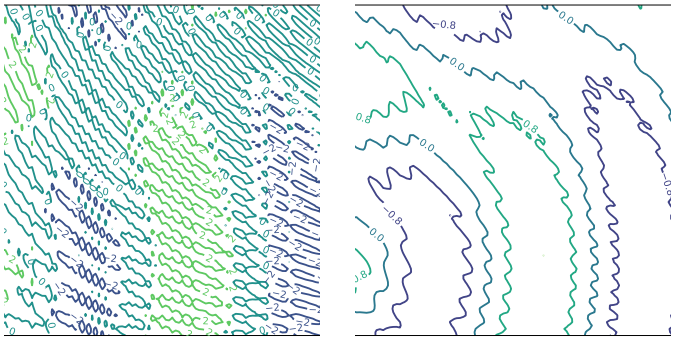
$$\text{GELU}(x) \quad = x \cdot \tfrac{1}{2}\left[1 + \text{erf}(\tfrac{x}{\sqrt{2}})\right] \tag{2}$$

where *erf* is the Gaussian error function [8].

GeLU has been shown to be resilient to the vanishing gradient problem and forms a critical aspect of our work.

Finally, we utilized the concept of skip connections that ensures that we do not overfit our model to the dataset and thus the final trained model generalizes well. The strength of the skip connection is specified by the hyperparameter k. In Figure 2, it can be observed that modifying the hyperparameter k for skip connections has the effect of simplifying the energy landscape of our total loss function. This, in turn, leads to quicker convergence. It is worth mentioning that these skip connections are different from the skip matrices utilized in the previously mentioned multi-head attention model.

Once the data has been processed by the transformer encoder, it is then input into a single-layer perceptron head to classify the inputs as either malicious or benign. The final layer produces the label as the output, while the attention matrices are obtained from the multi-head attention layers within the transformer encoder. In Section IV, we assess both the accuracy of our model's predictions and the efficacy of the attention vectors.



(a) No Skip Connection (b) Skip Connection: $k = 7$

Fig. 2: Effect of Skip Connections on Energy Lanscape of Loss in Transformer Encoder

IV. Experiments and Results

A. Implementation

In order to obtain the opcode sequences from binary files, our initial step involves disassembling the executable by utilizing a disassembler tool. Subsequently, we parse the assembly code to extract the opcode sequences, which constitute the actual machine code instructions. To prepare these opcode sequences for input into the model, we perform cleaning and preprocessing to eliminate unnecessary information and ensure their proper formatting.

The aforementioned pipelines were executed using custom scripts developed in-house to prepare the datasets. Additionally, the deep learning models, which integrated vision transformers, GeLU normalization, and skip connections, were implemented using the widely-used PyTorch Library [9].

KerasTuner [10] is a Python package that enables the automation of hyperparameter tuning. Hyperparameters are the configurations of a machine learning model that govern its learning process. KerasTuner can assist in finding the optimal hyperparameter values for a specific model, thereby enhancing its performance.

B. Dataset Description

Provenance signalling can be learned from any binary. However, to enable reproducibility of our work, we decided to use standard datasets which are publicly available. These datasets have been widely used and thoroughly analyzed in the field, making them an ideal testing ground for our model.

We acquired various publicly available datasets and merged them. Subsequently, we divided the combined dataset into training and test sets using random sampling. These datasets consist of over 218,000 malware samples from MalwareBazaar [11]. Additionally, we included more than 120,000 recent malicious samples from the reputable Ember Dataset [12]. Furthermore, we incorporated 55,000 of the most recent malware samples from VirusTotal. As for benign samples, we collected them by downloading executables from github releases pages and sourceforge. Overall, our dataset comprised over 390,000

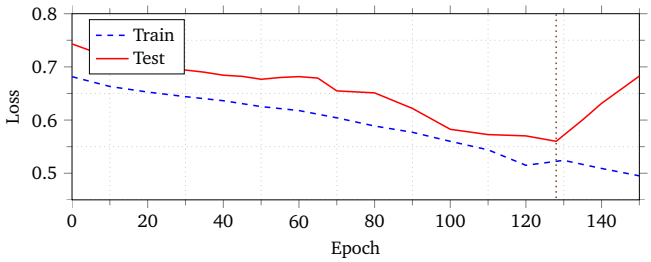


Fig. 3: Train/Test Loss and Early Stopping Point

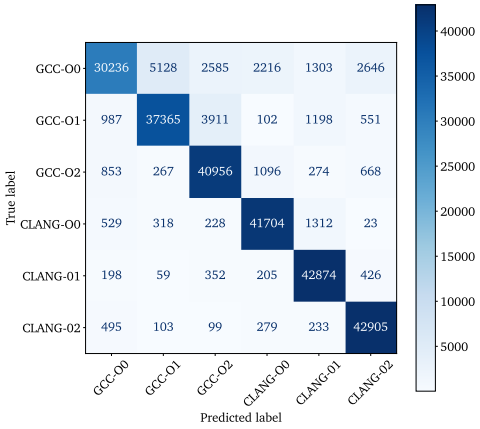


Fig. 4: Confusion Matrix

malicious samples and nearly 400,000 benign samples. We slightly trimmed this extensive dataset to eliminate some exceptionally large files. A further random sampling of 10% was performed to reduce the dataset size since we would need to increase the size again for preparing provenance signals.

All the resulting binaries were then compiled using both GCC and Clang and with all three levels of optimzation (-O1, -O1 and -O2) labelling each of the resulting binary sequences for upstream feeding into the training pipeline.

C. Results

The performance of our model can be observed through the loss plot shown in Figure 3. We conducted training for more than 150 epochs, witnessing a gradual decline in loss during the initial 85 epochs. Afterwards, the training loss decreased rapidly until the 128th epoch. However, at this stage, the test loss started to increase. To prevent overfitting, we implemented an *early stopping* mechanism, which halted training around the 130th epoch. The confusion matrix for all six classes (two compilers times three optimization levels) can be seen in Figure 4 and the classification report can be seen in Table I.

Finally, we have converted the 1D opcode sequence into a 2D representation by adjusting the hyperparameter Ω, which determines the width of the 2D matrix. This adjustment enables us to identify different features based on the value of Ω. By varying the hyperparameter Ω, which represents the width of the 2D matrix, we can unveil various features in binaries. This, however, is omitted from this paper due to space limitations and forms a basis of our future work.

Class	Precision	Recall	F1-Score
GCC-O0	0.9080	0.6854	0.7812
GCC-O1	0.8641	0.8470	0.8555
GCC-O2	0.8509	0.9284	0.8880
Clang-O0	0.9145	0.9454	0.9297
Clang-O1	0.9085	0.9719	0.9391
Clang-O2	0.9086	0.9726	0.9395

TABLE I: Classification Report for Provenance Signals

V. Conclusion

Provenance signaling, which entails the tracking of source information for digital artifacts, plays a crucial role in enabling various upstream tasks such as malware analysis. However, current methods for provenance signaling either rely on labor-intensive manual analysis or machine learning models that heavily rely on manually curated input features. This curation process necessitates the involvement of human experts, which is both time-consuming and impractical for large-scale applications. In this paper, we introduced a new model for provenance signaling that leverages raw binary data to overcome these limitations. Future directions for this research include studying the effect of the hyperparameters Ω and k on the results as well as studying the explainability of the results produced by our modified vision transformer model.

References

[1] B. Pan, N. Stakhanova, and S. Ray, "Data provenance in security and privacy," *ACM Computing Surveys*, 2023.

[2] S. Alrabaee, M. Debbabi, P. Shirani, L. Wang, A. Youssef, A. Rahimian, L. Nouh, D. Mouheb, H. Huang, A. Hanna *et al.*, "Compiler provenance attribution," *Binary Code Fingerprinting for Cybersecurity: Application to Malicious Code Fingerprinting*, pp. 45–78, 2020.

[3] J. F. Pimentel, J. Freire, L. Murta, and V. Braganholo, "A survey on collecting, managing, and analyzing provenance from scripts," *ACM Computing Surveys (CSUR)*, vol. 52, no. 3, pp. 1–38, 2019.

[4] X. He, S. Wang, Y. Xing, P. Feng, H. Wang, Q. Li, S. Chen, and K. Sun, "Binprov: Binary code provenance identification without disassembly," in *Proceedings of the 25th International Symposium on Research in Attacks, Intrusions and Defenses*, 2022, pp. 350–363.

[5] H. Chefer, S. Gur, and L. Wolf, "Transformer interpretability beyond attention visualization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 782–791.

[6] 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.

[7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2020.

[8] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[9] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[10] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi *et al.*, "Kerastuner," https://github.com/keras-team/keras-tuner, 2019.

[11] Abuse.ch, "Malware Bazaar," https://bazaar.abuse.ch/, 2023.

[12] H. S. Anderson and P. Roth, "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models," *ArXiv e-prints*, Apr. 2018.