PEMS-API: Malware Classification Using Parameter-Enhanced Multi-dimensional API Sequence Features

Han Miao[1,2], Wen Wang[1(✉)], Wanqian Zhang[1], and Feng Liu[1]

[1] Institute of Information Engineering, CAS, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
{miaohan,wangwen,zhangwanqian,liufeng}@iie.ac.cn

Abstract. The abuse of encryption and obfuscation in malware poses a significant threat to cybersecurity. Dynamic API call sequences, which directly reflect malware behavior and are hard to falsify, offer more robust and reliable features for classification and detection than static ones. Based on our analysis, we identify the following key characteristics in API call sequences: (1) The implementation of malicious functionality often involves the allocation and interaction of resources. API calls to the same resource object, such as files or registries, typically exhibit contextual dependencies, regardless of whether they are adjacent; (2) Multi-process interleaved execution is common in malware, and API sequences can be organized by execution order or process grouping. The sorting method can impact model performance, especially for multi-process malware; (3) API sequences often contain many consecutive repeated API names, but their parameters may differ. Therefore, we can distinguish these repeated calls by their parameters, rather than simply removing redundancy through truncation. Based on these observations, we propose a malware classification ensemble model that integrates multi-dimensional API sequence features. Specifically, we train separate classification models based on three different feature perspectives: the API resource graph, multi-process API sequence representation, and parameter-enhanced API name sequences. The outputs of these three base models are then aggregated using K-Nearest Neighbors (KNN) soft voting. Training and evaluation on three classification tasks demonstrate that the three base models outperform existing API sequence-based detection techniques, and the ensemble model further enhances the detection performance.

Keywords: Malware Classification · API Sequence · Neural Networks.

1 Introduction

The rampant use of encryption and obfuscation techniques in malware poses significant challenges for detection and comprehension. Malware authors employ these techniques to conceal their true intentions and evade analysis, making

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_4

it extremely difficult for security researchers to understand and mitigate the threats. Running malware in controlled sandbox environments and analyzing the generated API call sequences has become a widely adopted approach in the field of malware detection. API calls provide direct insights into the underlying behavior of malware. By examining these API call traces, analysts can gain valuable intelligence about the functionality, goals, and potential impact of the malware.

There are numerous malware detection methods that utilize dynamic behavioral features, which can be categorized into the three types: API sequences-based models [2,3,9,18,21,22] , API graph-based models [5,6,10,12,19] and API arguments enhanced meodel [7,17,23]. API2Vec [5]and API2Vec++ [6] consider the relationships between processes by constructing process relationship graphs and leveraging NLP techniques to represent API sequences. [10] and [12] also construct graph models to describe API call relationships and further employ pattern recognition and deep learning techniques for malware detection. However, most methods above adopt a single feature perspective and have a relatively coarse granularity, primarily focusing on the names of API sequences and statistic features of API arguments. This would result in a partial representation of the API sequence, focusing only on a subset of the information. To extract more implicit information from API sequence parameters, we have made several observations based on extensive analysis. The findings are as follows:

Resource Allocation and Interaction: Malicious functionality in malware often requires the allocation and interaction of various system resources. API calls that operate on the same resource object, such as files or registries, typically exhibit contextual dependencies, even if they are not adjacent in the sequence. This is particularly important in the analysis of malware, as understanding the context of resource interactions can provide valuable insights into its behavior. Ignoring these dependencies may lead to a less accurate representation of the malware's dynamic execution.

Multi-process Interleaved Execution: Malware often executes across multiple processes that interleave their activities. In such scenarios, the API sequences can be organized in two primary ways: by the order of execution or by grouping based on the process. The method of sorting API calls can significantly affect the performance of classification models, especially those that rely on the order of sequence. When applied to multi-process malware, models that depend heavily on the sequence order may experience a performance decline, as the interleaved nature of the processes introduces complexity that is not adequately captured by a simple linear sequence. This observation underscores the need for more sophisticated approaches to handling multi-process malware and highlights the importance of considering process grouping in the analysis.

Consecutive Repeated API Calls with Varying Parameters: API sequences often contain consecutive repeated API calls, where the API names are the same, but their parameters may differ. These repeated calls, while seemingly redundant, can provide important contextual information. Rather than simply applying a truncation method to eliminate redundancy, distinguishing these re-

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_4

peated calls based on their parameters can provide additional insights into the functionality of the malware. By considering the variation in parameters, we can more accurately capture the underlying behavior of the malware, as different parameters may represent different stages or types of malicious activity.

Based on these three observations, we propose a malware classification ensemble model that integrates multi-dimensional features from API sequences. Our approach incorporates three distinct perspectives of API sequence features: the API resource graph, which models the interactions and dependencies between API calls and resource objects; the multi-process API sequence representation, which accounts for the interleaved execution of multiple processes; and the parameter-enhanced API name sequences, which distinguishes repeated API calls by considering their parameters. To combine the outputs of these models, we use K-Nearest Neighbors (KNN) soft voting, which allows us to leverage the strengths of each individual model and make a more robust classification decision.

The contributions in this paper are as follows:

- We propose a malware classification ensemble model that combines multi-perspective API sequence features: API resource graph, multi-process API sequence representation, and parameter-enhanced API name sequences. Both individual models and the ensemble model achieve outstanding results in classification tasks.
- We first propose using the handle parameter, which uniquely identifies resource objects, to build the API resource graph. This approach is more accurate and efficient compared to the previous method, which relied on identifying objects through strings.
- We first propose using API parameters to address the issue of redundant repetition in API sequences.
- We first propose a cross-attention-based Multi-process API Sequence Representation method and a process state transition graph to address the representation challenge of multi-process malware.

2 Related Work

2.1 API Sequence-based malware detection

Sequence-based malware detection [2–4, 8, 13] utilizes deep learning to directly extract features from API call sequences for malware detection. Various methods have been proposed in this domain: Kwon et al. [14] and Yazi et al. [15]apply LSTM to capture API calling patterns for classification, while Tobiyama et al. [16] use an RNN for feature extraction, followed by CNN classification of the generated feature images. ASSCA [17] employs a bidirectional residual network to classify API sequences, removing redundant information. Additionally, Li et al. [18] utilize Bi-LSTM to capture and combine intrinsic API sequence features. While sequence-based methods are simple and do not require extensive prior knowledge, they face challenges: long sequences can obscure key malicious

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_4

behavior features, adversarial malware can insert noise APIs to evade detection, and these methods often require large datasets while struggling to exploit API relationships effectively.

2.2 API Graph-based Malware Detection

Researchers have proposed graph-based malware detection methods to model the behavior of APIs using graphs [10, 12, 19], then apply Graph Neural Networks (GNNs) for feature extraction in malware detection. API2Vec [5] and API2Vec++ [6] designs Temporal Process Graphs (TPG) and Temporal API Property Graphs (TAPG) to model inter- and intra-process behaviors for detection and using random walk to extract meta path to represent the API sequence for classification. DMalNet [10] converts API call sequences into call graphs to model API relationships and enhance malware classification. MINES [12] extracted the API existence feature of malware by graph contrastive learning between two API graphs. Graph-based approaches generally outperform sequence-based methods by using GNNs to capture the complex behavior of software through API relationships. However, these methods often fail to fully account for the diverse API relationships, as they typically focus on only one type of relationship.

2.3 API Arguments Enhanced Methods

DMDS [7] firstly propose a feature engineering method by utilizing a feature hashing trick to encode the API call arguments. CTIMD [11] integrates Cyber Threat Intelligence (CTI) to enhance sequence learning with runtime parameters. DMalNet [10] and Malatt [8] both used feature encoder that uses different encoding strategies according to the characteristics of different types of data to represent API names and arguments as semantic feature vectors in their works. These methods enhance the model's expressive power and feature information by incorporating parameters, yielding relatively good results. However, using specific parameter values, such as IP addresses and file paths, may reduce the model's generalization ability and make it more vulnerable to evasion attacks.

3 Methodology

As shown in Fig. 1, our approach to malware detection encompasses multiple perspectives and leverages advanced techniques to capture intricate details and behavioral patterns. The proposed method consists of four key components:

3.1 API Resource Graph

Motivation The implementation of malicious functions requires resource management, allocation, and interaction through APIs, such as reading and writing

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_4

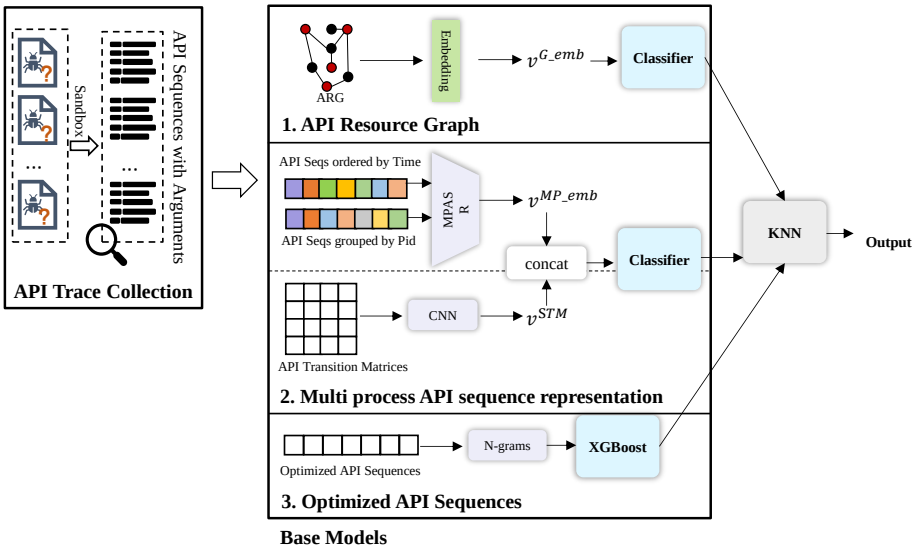


Fig. 1. System architecture of PEMS-API

files, reading and writing registry, etc. The interactions between processes and resources are quite different between malicious and benign programs.

The challenge to construct API Resource Graph is identifying resource nodes. Different resource nodes may have different representations in various APIs. For example, a file node can be represented as absolute path or file name, among others. Recogning resources by strings like file path or file name may result in multiple duplicate resource nodes. API Resources Graph constructed by Heternet [19] remains the problem.

In the Windows operating system, handles are unique 32-bit or 64-bit unsigned integers used to identify objects created or used by applications. These objects include windows, modules, threads, processes, files, mutexes, sockets, and more. Within the same process, the mapping between objects and handles is one-to-one, regardless of the object type. Therefore, by examining the handles present in the API call parameters, we can identify the different objects involved in the program's execution. As shown in Fig. 2, resource objects could be indentified from API arguments.

For the same reason, socket is also an important identifiers for network objects in a program. Under the assumption that handles and socket serve as bridges connecting API calls and resources, we can construct dependencies between discrete APIs by leveraging the affinity of resource.

The Framework of classifier based on is shown in Fig. 3).

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_4

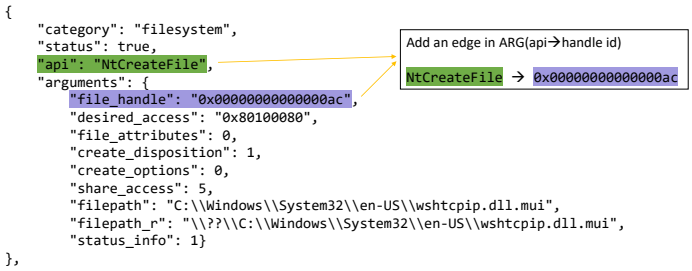


Fig. 2. An Example of API and corresponding arguments.

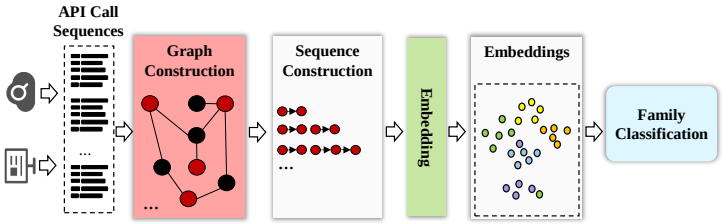


Fig. 3. Overview of classifier based on ARG.

Graph Construction The constructed ARG can be represented as:

$$G = (V_{\text{API}} \cup V_{\text{Resource}}, E) \tag{1}$$

where V_{API} contains the API nodes, V_{Resource} contains the resource object nodes, and E contains the edges representing interactions between the APIs and the resources.

The set V_{Resource} contains objects that are identified through the parameters of API calls. Specifically, the resource objects are extracted from parameters such as: file_handle, key_handle,process_handle, socket. These parameters correspond to specific system resources, such as open files, registry keys, processes, and network sockets.

From the perspective of API Resource Graph, we gain valuable insights into the behavior and resource usage patterns of the program. The ARG allows us to analyze the dependencies between API calls and the resources they access, providing a comprehensive view of the program's execution flow.

By focusing on resource usage in API arguments, we can uncover hidden dependencies and interactions that may not be visible through API name sequence alone. The ARG enables us to identify suspicious or malicious resource usage patterns.

Graph Representation We adopted a simple yet effective method to represent the Application Resource Graph (ARG). For each object node, we gather all adjacent API nodes and order them chronologically, forming a set of API paths.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_4

To obtain meaningful representations, we use Doc2Vec [22] to learn embeddings for both the API paths and individual APIs from a large corpus. Each path and API is represented as a 64-dimensional vector, capturing their interactions with resources. This approach enables effective analysis and comparison of API behaviors.

Let P represent the set of paths. For the i-th path p_i and a specific API p_j^i within p_i, we first collect the context APIs within a window of size C, denoted as $\delta = \{p_{j-C}^i, \ldots, p_{j-1}^i, p_{j+1}^i, \ldots, p_{j+C}^i\}$. The representation of p_j^i is then computed as the embedding of p_i combined with the embeddings of the APIs in δ, given by:

$$E(p_j^i) = W \cdot \frac{1}{2C+1} \left(E(p_i) + \sum_{k \in \delta} E(p_k^i) \right) \tag{2}$$

where $E(\cdot)$ denotes the embedding function for an API or path, and W is the weight matrix learned during training.

The objective is to minimize the following loss function, which represents the average negative log-likelihood of each API across all paths:

$$-\frac{1}{N_p} \sum_{i=0}^{N_p} \frac{1}{N_{p_i}} \sum_{j=0}^{N_{p_i}} \log P(p_j^i | \delta, p_i) \tag{3}$$

Here, N_p is the total number of paths in P, N_{p_i} is the length of path p_i, and $P(p_j^i|\delta, p_i)$ represents the probability of a context API p_j^i given the current API p_i and its surrounding context δ. This formulation aims to optimize the embeddings by minimizing the log-likelihood of context API predictions within the defined window.

3.2 Multi process API sequence representation

Motivation Multi-process and inter-process interactions are major characteristics of malware behavior, such as process injection attacks, process replacement attacks, and others. Existing methods rarely capture the process-related profiles of API sequences. Typically, past approaches organize API sequences as a single, long sequence, ignoring the fact that they may contain multiple distinct functional processes and subsequences. In this section, we organize the input API sequence in two ways: a globally ordered sequence and process-grouped sequences,illustrated in Fig. 4. The global API sequence is first sorted by time and then divided into blocks based on process. Each block can be seen as accomplishing an independent function. As illustrated in the figure below, this clearly exhibits a hierarchical structure: API words, API groups (sentences), and complete API sequences (documents). Furthermore, to directly express the process structural information, we propose the concept of a Process State Transition Graph (PSTG) and utilize a convolutional network to obtain the transition matrix representation. The architecture of Multi process API sequence representation is shown in Fig. 4.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_4

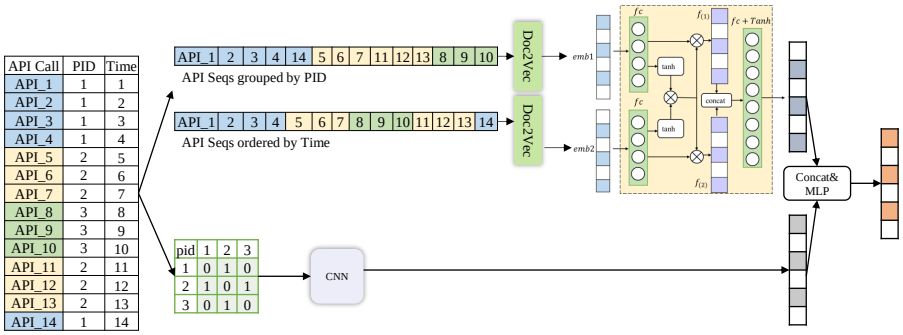


Fig. 4. Architecture of Multi process API sequence representation.

Based on the above insights, we propose the Multi-process API Sequence Representation (MPASR) method. The overall structure of the method is shown in the diagram below. It includes two primary modules: the *Cross-Attention-based Semantic Representation Module* and the *Process State Transition Structure Graph Module*. These modules extract the semantic information from the API sequence and the process structural information, respectively, and serve as input for the final deep learning classifier.

Cross-Attention-Based Semantic-aware Modeling In the semantic-aware module, we propose a cross-attention-based multi-process API sequence semantic enhancement representation learning. This approach has several advantages. First, grouping the API sequences by process allows the sequence to be represented from the perspective of individual processes. Second, cross-attention enhances the feature representation. Third, the Doc2Vec representation can effectively handle longer API sequences.

Semantic-aware Encoder We analogize API subsequences to sentences and view the complete API sequences as documents. Therefore, Doc2Vec is an efficient representation method that aligns with this characteristic. Based on this, we obtain the representations of the two types of API sequences using Doc2Vec. The relevant mathematical expressions have been presented above.

Cross-Attention Semantic Enhancement As shown in the Fig. 4, after obtaining the embeddings of the API call sequences under two organizational structures, the cross-attention mechanism is applied to establish connections between the two API sequences, thereby enhancing the semantic representation of the multi-process API sequences. In the figure, emb_1 represents the embedding of the API sequence sorted by execution time, and emb_2 represents the embedding of the API sequence grouped by process ID and then sorted by the first appearance time of each process. The cross-attention module consists mainly of three fully connected layers: l_1, l_2, and l_3, with parameter matrices defined as $W_1 \in \mathbb{R}^{d_f \times d_{f_1}}$,

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_4

$W_2 \in \mathbb{R}^{d_f \times d_{f_2}}$, and $W_3 \in \mathbb{R}^{d_{f_a} \times d_{f_h}}$, where d represents the feature vector dimensions, and d_{f_1} and d_{f_2} are equal, representing the output dimensions of fully connected layers l_1 and l_2, respectively. d_{f_a} denotes the dimension of the attention features after concatenation, and d_{f_h} represents the output feature dimension of the linear layer l_3. First, we obtain f_1 and f_2 as latent features from the intermediate layer, and the calculation is as follows:

$$f_1 = f \cdot W_1, \quad f_2 = f \cdot W_2 \tag{4}$$

where $\cdot$ denotes matrix multiplication. Specifically, we apply the activation function tanh to map the intermediate feature values to the range $(-1, +1)$, obtaining the corresponding feature sign vectors v_1 and v_2 to represent the latent feature values. Finally, the feature vector α is obtained by multiplying the two sign vectors as shown below:

$$\alpha = v_{s_1} \cdot v_{s_2} \tag{5}$$

After obtaining α, we combine it with the latent feature vectors f_{a_1} and f_{a_2}. The two new feature vectors are concatenated into a high-dimensional vector and passed as input to the final linear layer l_3. The output f_h is the final output after the cross-attention operation, as shown in the following equation:

$$f_h = \text{Concat}(\alpha f_1, \alpha f_2) \cdot W_3 \tag{6}$$

Finally, the embeddings enhanced by cross-attention retain the information of the original two types of API sequences while also capturing the representations of API calls related to inter-process interactions.

Process State Transition Graph The Process State Transition Graph (PSTG) is constructed based on the time-ordered API sequence. We further divide the sequence into blocks based on the process ID, encoding each block with its corresponding process number. A directed edge is drawn between adjacent blocks based on their sequence in the API flow. The resulting structure captures the transitions between processes, providing a state transition matrix, shown in Fig. 4. In the time-ordered API sequences,there may be cases where API sequences are interleaved. Let the state transition matrix be T. If two APIs are adjacent in execution order but belong to different processes, a process state transition occurs, and we update $T_{i,j}$ += 1, where i and j are the indices of the two processes involved in the context switch. Note that the process index is different from the PID—it starts from 0, with the first encountered process labeled as 0, and so on. This approach helps reduce the dimensionality of the state transition matrix.

After getting the state transition matrix $PTSG$, we use ResNet [24] to process it. ResNet facilitates efficient information transmission through shortcut connections. We employ an 11-layer ResNet with 3 residual blocks. All convolution kernels have a size of 3×3, as we aim to capture subtle changes in the graph. Next, we use a global max pooling layer to compute the embedding of the

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_4

state transition graph. Specifically, the embedding of the state transition graph is given by:

$$\mathbf{e_{PTSG}} = Maxpooling(\text{ResNet}(PTSG)) \tag{7}$$

We only apply pooling methods in the last layer to handle varying input sizes.

3.3 Argumewnts Enhanced API Sequences

By analyzing the API sequences and their parameters, we found that malicious software's dynamic API call sequences often contain many repeated API names, but with different parameters. Previous methods treated these as noise and simplified the sequences by removing duplicates. Based on this insight, we propose a set of heuristic strategies based on parameter content to optimize the representation of API sequences. As shown in the Fig. 5, 'LdrGetProcedureAddress' is used for loading functions at runtime and typically appears consecutively in the sequence. Its parameter, 'function_name', indicates the function whose address needs to be obtained. Therefore, we replace the original API sequence with 'API_function_name'. Similarly, we use 'sleep_dwMilliseconds' instead of 'sleep', and 'NtDelayExecution_DelayInterval' instead of 'NtDelayExecution'.

The benefit of this approach is that it enriches the representation of the API sequences without reducing the model's generalization ability or causing adversarial detection.

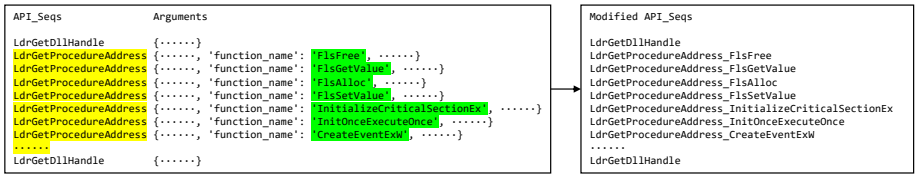


Fig. 5. Example of simplifying a repeated API subsequence based on parameters

After getting modified API sequences, we use N-grams to represent the sequence and apply machine learning techniques for classification.First, we extract N-grams from the API sequences. N-grams are contiguous sequences of N items from a given sample of text, and they are used to capture local patterns in the API sequences. We choose the most frequent N-grams in the training data to construct feature vectors for each sample. Specifically, for each API sequence, we compute a feature vector where each element corresponds to the frequency of a particular N-gram in that sequence. Then, we select features with mutual information to gain the top K_2 most relevant features, where mutual information $I(X, Y)$ between features X and label Y is defined as:

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_4

$$I(X,Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \tag{8}$$

The top K_2 features are selected based on the highest mutual information scores. After selecting the features, we apply the XGBoost classifier to train a model on the selected features.

3.4 Ensemble Learning

After obtaining the results from the three models, we use ensemble learning to enhance the performance of the classification model, improving its accuracy and robustness by combining the predictions from multiple models. Ensemble learning, using the voting method, integrates these outputs to provide a final prediction. The voting method can be classified into hard voting (majority vote) and soft voting (based on class probabilities). Here, K-Nearest Neighbors (KNN) is used for soft voting. For optimal performance, the base models should meet two conditions: 1. Similar accuracy: Models should have comparable accuracy to avoid one dominating the voting process. 2. Low homogeneity: Models should be diverse to capture different aspects of the data, providing complementary insights that enhance overall performance.

4 Experiments

4.1 Experimental Setup

Dataset In our paper, both API name sequences and called arguments of dynamic excution are required. Although there are some publicly available API sequences dataset, they are not suitable for our approach due to the absence of API arguments. Thus, we collect malware and goodware from various sources by ourselves and execute each sample in cuckoo sandbox [20] environment for 2 minutes. Then, we get API call sequences and arguments from the dynamic execution logs. Finally, the experiment was conducted on a dataset containing 19088 malware from 62 families and 11030 goodware. Malware are categorized into 12 classes based on attack intent and techniques. We construct three classification tasks — malware family classification, malware functionality classification and malware detection.

Evaluation Metrics We evaluate our method and compared methods using five widely used metrics, including Accuracy and F1-Score.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

$$Precision = \frac{TP}{TP + FP} \tag{10}$$

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_4

$$Recall = \frac{TP}{TP + FN} \tag{11}$$

$$F1 - Score = 2 \times \frac{\text{Precision} \times Recall}{Precision + Recall} \tag{12}$$

, where TP and TN are correct detections of positives and negatives, while FN and FP are misclassified positive and negative samples, respectively.

4.2 Results and Analysis

Comparative results The comparative results are presented in Table 1. The following observations can be made from the experimental results:

Firstly, in the comparison of individual models, MPAR achieved the best performance in terms of the metrics. ARG and Modified_API also outperformed most other models. Although these methods only extract partial features of the API sequences, such as resource dependency information, process semantics and structural information of the API sequences, and the existence information of optimized APIs containing parameter information, they still perform well, indicating that these partial features are useful for classification tasks. MalAtt exhibits comparable performance to the three standard models we proposed; however, considering that it integrates static opcode information, which is vulnerable to code obfuscation attacks, its performance will degrade significantly when samples are obfuscated. Additionally, the graph-based method API2Vec did not perform as expected. We believe this could be due to the large number of duplicate nodes hindering the random walk process from exploring more diverse paths. Furthermore, methods that incorporate parameter information (such as DMDS, MalAtt, Dmalnet, and Agrawal et al.) generally outperform methods that rely solely on the API name sequence. Finally, we observed that as the complexity of the classification task increases and the number of categories grows, classification performance tends to decrease, as capturing more intricate details becomes necessary to further distinguish between samples.

Ablation study To evaluate the contribution of each model to overall accuracy, we compared individual models as well as their pairwise combinations. Among the individual models, Multi-process API Sequence Representation (MPAR) achieved the highest accuracy, followed by API Resource Graph (ARG). As illustrateed in 1,when ARG and MPAR were used together, the accuracy on the family classification task increased by 0.169% compared to using MPAR alone. We hypothesize that although there is a large accuracy gap between the two base models, their low homogeneity allows for performance improvement when combined. No combination led to a decrease in accuracy, which we attribute to the low homogeneity and relatively small accuracy differences between the three models, making this an ideal scenario for deep ensemble learning. When all three models were used together, the accuracy improved by as much as 1.98%.

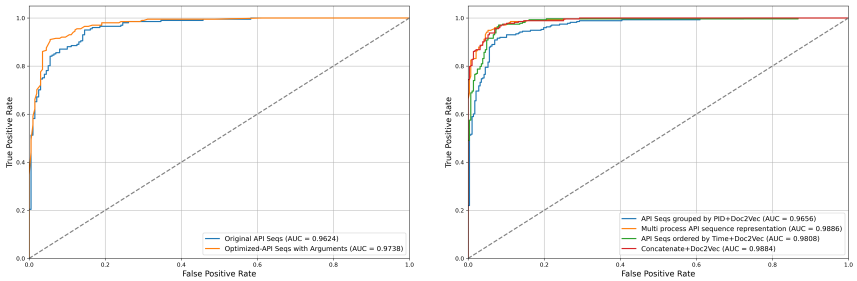
Additionally, we explored whether optimizing the processing of API sequences through parameter tuning enhances classification performance, and how different API sorting strategies influence the results. We also assessed whether our

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_4

Table 1. Comparisons of different models on three tasks. The best results are in boldface.

Model	Family Classification		Function Classification		Malware Detection		Inference Time (ms/sample)
	Accuracy	F1-Score	Accuracy	F1-Score	Accuracy	F1-Score	
API2Vec [5]	0.7651	0.7134	0.8017	0.7533	0.8929	0.8552	919.9
DMDS [7]	0.8937	0.8383	0.917	0.8443	0.9314	0.882	21.2
TextCNN [9]	0.613	0.5847	0.6521	0.6019	0.7244	0.6915	19.8
Agrawal et al. [1]	0.9087	0.9182	0.9331	0.924	0.9551	0.9291	34.5
MalAtt [8]	0.9126	0.9122	0.9363	0.9166	0.972	0.9481	28.3
Dmalnet [10]	0.9136	0.9097	0.9214	0.9138	0.9371	0.9257	20
MPAR	0.9213	0.9057	0.9372	0.9237	0.9796	0.9556	21.1
ARG	0.9157	0.91	0.9353	0.9113	0.9543	0.9342	25.3
Modified_API	0.9172	0.891	0.9297	0.9077	0.9712	0.9466	**18.5**
ARG+MPAR	0.9382	0.9191	0.949	0.9345	0.9874	0.9565	46.7
ARG+Modified_API	0.9255	0.9032	0.9448	0.9179	0.9836	0.9546	39.8
Modified_API+MPAR	0.9318	0.9089	0.9417	0.9111	0.9866	0.9547	44.2
Ensemble Model	**0.9411**	**0.9373**	**0.9574**	**0.9411**	**0.9884**	**0.9579**	65.4

cross-attention mechanism offers superior performance compared to simply concatenating the two embeddings. As illustrated in Fig. 6, the optimized API sequences yielded better results than the original sequences when used as input. Moreover, the cross-attention-based API sequence representation method outperformed both individual API sequences and the concatenated version of the two sequences, which aligns with our expectations.



(a) ROC of methods with different API processing strategies

(b) ROC of methods with different API Sequence order

Fig. 6. Comparison of ROC Curves

Inference Time Overhead The inference time overhead of this method comes from three components: ARG, MPAR, and Modified_API. The experiments were conducted on an Ubuntu (20.04.2 LTS) server with a 32-core AMD EPYC 9654 96-Core Processor and 120GB of RAM. The average processing time per sequence for each of the three components was 21.1ms, 26.3ms, and 18.5ms, respectively, while the time for processing with the three models combined was

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_4

65.4ms. In comparison, API2Vec, DMDS, TextCNN, Agrawal et al., MalAtt, and Dmalnet took 919.9ms, 21.2ms, 19.8ms, 34.5ms, 28.3ms, and 20ms, respectively, to process one sequence. Although our method takes relatively longer, the 65.4ms detection time is still reasonable. Therefore, we believe this method is feasible for malware detection in real-world scenarios.

5 Conclusion

This paper proposes a hybrid model that combines deep learning and machine learning, leveraging ensemble learning to integrate different deep learning base models and obtain final results through a voting mechanism. The model extensively explores API information from three aspects: API resource graphs, multi process API sequence representation, and argumewnts Enhanced API sequence n-grams. These heterogeneous pieces of information are integrated to further improve accuracy. We evaluated the model on a large dataset composed of real-world software. Experimental results show that the proposed model outperforms other comparison models in terms of accuracy and f1 score.

Acknowledgments. This work was supported by the Program with No.E3YY131112.

References

1. Agrawal, R., Stokes, J.W., Marinescu, M., Selvaraj, K.: Neural sequential malware detection with parameters. In: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 2656–2660. IEEE (2018)
2. Li, C., Zheng, J.: Api call-based malware classification using recurrent neural networks. Journal of Cyber Security and Mobility pp. 617–640 (2021)
3. Catak, F.O., Yazı, A.F., Elezaj, O., Ahmed, J.: Deep learning based sequential model for malware analysis using windows exe api calls. PeerJ Computer Science 6, e285 (2020)
4. Demirkıran, F., C¸ ayır, A., Unal, U., Da g, H.: An ensemble of pre-trained transformer models for imbalanced multiclass malware classification. arXiv preprint arXiv:2112.13236 (2021)
5. Cui L., Cui J., Ji Y, Hao Z., Li L., Ding Z. API2Vec: Learning Representations of API Sequences for Malware Detection In Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2023).,pp. 261–273, https://doi.org/https://doi.org/10.1145/3597926.3598054 (2023)
6. Cui L., Yin J., Cui J., Ji Y., Liu P., Hao Z., Yun X. API2Vec++: Boosting API sequence representation for malware detection and classification IEEE Trans. Softw. Eng., 50 (8) (2024), pp. 2142-2162, 10.1109/TSE.2024.3422990 Conference Name: IEEE Transactions on Software Engineering
7. Li C., Cheng Z., Zhu H., Wang L., Lv Q., Wang Y., Li N., Sun D. DMalNet: Dynamic malware analysis based on API feature engineering and graph learning Comput. Secur., 122 (2022), Article 102872, https://doi.org/10.1016/j.cose.2022.102872
8. H. Bao, W. Li, H. Chen, H. Miao, Q. Wang, Z. Tang, F. Liu, and W. Wang Stories behind decisions: Towards interpretable malware family classification with hierarchical attention.Comput. Secur.,144 (2024), Article 103943, https://doi.org/10.1016/j.cose.2024.103943

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_4

9. Qin B., Wang Y., Ma C., API Call Based Ransomware Dynamic Detection Approach Using TextCNN, 2020 International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE), Fuzhou, China, 2020, pp. 162-166, https://doi.org/10.1109/ICBAIE49996.2020.00041.
10. Li C., Cheng Z., Zhu H., Wang L., Lv Q., Wang Y., Li N., Sun D. DMalNet: Dynamic malware analysis based on API feature engineering and graph learning Comput. Secur., 122 (2022), Article 102872, 10.1016/j.cose.2022.102872
11. Chen T., Zeng H., Lv M., Zhu T. CTIMD: Cyber threat intelligence enhanced malware detection using API call sequences with parameters Comput. Secur., 136 (2024), Article 103518, https://doi.org/10.1016/j.cose.2023.103518
12. Wu P., Gao M., Sun F., Wang X., Pan L. Multi-perspective API call sequence behavior analysis and fusion for malware classification, Comput. Secur., 148 (2025), Article 104177, https://doi.org/https://doi.org/10.1016/j.cose.2024.104177.
13. Kolosnjaji, B., Zarras, A., Webster, G.D., Eckert, C., 2016. Deep learning for classification of malware system call sequences. In: Advances in Artificial Intelligence. pp. 137–149.
14. Kwon, I., Im, E.G., 2017. Extracting the representative API call patterns of malware families using recurrent neural network. In: International Conference on Research in Adaptive and Convergent Systems. pp. 202–207.
15. Yazi, A.F., Catak, F.O., Gul, E., 2019. Classification of methamorphic malware with deep learning (LSTM). In: IEEE Signal Processing and Communications Applications Conference. pp. 1–4.
16. Tobiyama, S., Yamaguchi, Y., Shimada, H., Ikuse, T., Yagi, T., 2016. Malware detection with deep neural network using process behavior. In: IEEE Annual Computer Software and Applications Conference. COMPSAC, pp. 577–582.
17. Lu X., Jiang F., Zhou X., Yi S., Sha J., Li P. ASSCA: API sequence and statistics features combined architecture for malware detection Comput. Netw. (2019), pp. 99-111
18. Li C., Lv Q., Li N., Wang Y., Sun D., Qiao Y. A novel deep framework for dynamic malware detection based on API sequence intrinsic features Comput. Secur., 116 (2022), Article 102686, 10.1016/j.cose.2022.102686
19. R. Song, L. Li, L. Cui, Q. Liu, J. Gao, Binary Malware Detection via Heterogeneous Information Deep Ensemble Learning 2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS), Ocean Flower Island, China, 2023, pp. 1147-1156 https://doi.org/10.1109/ICPADS60453.2023.00168.
20. Cuckoo Sandbox. Website, https://cuckoosandbox.org/, last accessed 2025/2/27
21. T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space arXiv preprint arXiv:1301.3781(2013). https://doi.org/10.48550/arXiv.1301.3781
22. Q. Le, T. Mikolov Distributed representations of sentences and documents. In International conference on machine learning. 2014 PMLR, 1188–1196.
23. N. T. Javan, M. Mohammadpour, S. Mostafavi Enhancing Malicious Code Detection With Boosted N-Gram Analysis and Efficient Feature Selection in IEEE Access, vol. 12, pp. 147400-147421, 2024, https://doi.org/10.1109/ACCESS.2024.3476164.
24. He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, 770–778.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97573-8_4