applied sciences

MDPI

Article

Dynamic Malware Analysis Based on API Sequence Semantic Fusion

Sanfeng Zhang [1,2], Jiahao Wu [1], Mengzhe Zhang [1] and Wang Yang [1,2,*]

[1] School of Cyber Science and Engineering, Southeast University, Nanjing 211189, China; sfzhang@seu.edu.cn (S.Z.)
[2] Key Laboratory of Computer Network and Information Integration, Ministry of Education, Southeast University, Nanjing 211189, China
* Correspondence: wang.yang@seu.edu.cn

Abstract: The existing dynamic malware detection methods based on API call sequences ignore the semantic information of functions. Simply mapping API to numerical values does not reflect whether a function has performed a query or modification operation, whether it is related to network communication, the file system, or other factors. Additionally, the detection performance is limited when the size of the API call sequence is too large. To address this issue, we propose Mal-ASSF, a novel malware detection model that fuses the semantic and sequence features of the API calls. The API2Vec embedding method is used to obtain the dimensionality reduction representation of the API function. To capture the behavioral features of sequential segments, Balts is used to extract the features. To leverage the implicit semantic information of the API functions, the operation and the type of resource operated by the API functions are extracted. These semantic and sequential features are then fused and processed by the attention-related modules. In comparison with the existing methods, Mal-ASSF boasts superior capabilities in terms of semantic representation and recognition of critical sequences within API call sequences. According to the evaluation with a dataset of malware families, the experimental results show that Mal-ASSF outperforms existing solutions by 3% to 5% in detection accuracy.

Keywords: malware; dynamic analysis; API call sequence; semantic feature; fusion

Citation: Zhang, S.; Wu, J.; Zhang, M.; Yang, W. Dynamic Malware Analysis Based on API Sequence Semantic Fusion. *Appl. Sci.* **2023**, *13*, 6526. https://doi.org/10.3390/app13116526

Academic Editor: Giacomo Fiumara

Received: 15 March 2023
Revised: 20 May 2023
Accepted: 23 May 2023
Published: 26 May 2023

1. Introduction

Background of Malware. Malware presents significant challenges to the security of network services and data assets and causes substantial economic losses to enterprises and individuals. An explosive growth trend is being observed in various types of high-risk malware, including spyware, botnets, ransomware, rootkits, and mining programs [1]. In the first quarter of 2021 alone, McAfee reported that more than 87 million new malicious samples were captured, involving about 930,000 new maliciously signed binary files [2]. Over one million suspicious files are uploaded to VirusTotal on a daily basis [3]. The explosive growth in the number of new variants and malware samples presents a serious challenge to the existing detection methods. It is difficult to cope with it through detection methods that simply use manual signatures [4] and feature code matching [5].

Background of Dynamic Analysis. Malware analysis methods can be classified into static and dynamic malware analysis [6]. Static analysis methods are seriously challenged when they face obfuscation techniques [7,8] and zero-day or polymorphic malware. Static analysis approaches tend to be low-cost but unreliable. Dynamic analysis refers to the execution and analysis of malicious samples in a controlled environment. Malicious behaviors must be implemented through underlying system API calls [9]. System APIs are usually called to obtain the relevant system permissions, modify the registry, establish network communication, monitor GUI operations, and detect sandboxes. Dynamic analysis methods are widely considered to be more resistant to interference by monitoring these

sensitive operations [10]. The existing dynamic analysis methods commonly trace these sensitive system API call behaviors, establish API call sequences, or directed behavioral graphs, and then analyze them by machine learning or deep learning methods [11,12].

Limitation of Existing Works. In recent decades, deep learning models, including CNN, RNN, LSTM, and BiLSTM, have been widely used to automatically obtain sequence features and locate malicious behaviors. However, recent studies [13–15] revealed that these models can be deceived through packing and black-box attacking technologies for both static and dynamic features.

There are two reasons for this issue. Firstly, simply mapping APIs to numerical values ignores the inherent semantic features of a function. An API function may perform a query or modification operation, and it may be related to network communication, the file system, or other factors. Simple numerical values cannot reflect these features. Secondly, existing models cannot reflect the sequential characteristics well. These models are not well equipped to handle large datasets. There is a universal decrease in model performance when these models are presented with a dataset containing excessive types of APIs, oversize feature sets, and overlong sequences. Therefore, the API call sequences are usually modified in such a way that key API information is ignored, resulting in degraded detection performance.

Goals and Approaches. It is considered that the sequence context and implicit semantics of functions play a crucial role in the classification of API call sequences. The function name of APIs implies several semantics, indicating various actions such as reading, writing, searching, and downloading, as well as various associated resources such as system privileges, networks, registry, and GUI interfaces. Encoding the API call sequence with one-hot vectors not only results in high-dimensional vectors but also leads to the loss of key semantic information. Techniques such as processing text sequences [16] and word embedding in natural language processing [17] can be employed to aid in dimension reduction and semantic representation.

As a deep learning-based malicious malware detection method, feature selection is not required. Starting from the sequence of API functions, the goal of this paper is to map each function into different operation types and resource-associated types according to its textual name, encode the API sequence into a high-dimensional matrix in the preprocessing stage, reduce matrix dimension through API embedding operations, obtain fused sequence features based on BiLSTM and TextRNN models, and perform classification output at last. By integrating the sequential features of API sequences and the semantic features of functions, this method can achieve better classification performance.

Contributions. Focusing on dynamic features, we proposed a malicious code detection model based on API-Sequence-Semantic Fusion (Mal-ASSF). We utilize various technologies, including API2Vec, TextRNN, BiLSTM, and self-attention mechanisms, to enhance the model. Our research has made the following contributions:

- API2Vec is a class of neural network models that can produce a corresponding vector for each unique API element in a continuous space in which the linguistic contexts of APIs can be observed. It is used to obtain the dimensionality reduction representation of the API call sequences. When establishing the correlation between APIs, we use a bidirectional long short-term memory network (BiLSTM) to capture the behavioral features of segments of different lengths.
- A pair of operation and type, separately representing the verb and the objective in the function name, is designed to represent the API functions. To fully discover the implicit semantic information of API functions, we construct the core model based on TextRNN and the self-attention mechanism, where the sequence feature and semantic feature of API are fused, and the suspected malicious segments of the sequences are focused on adaptively.
- To evaluate the effectiveness of Mal-ASSF, we applied a systematic experiment to a large dataset of malware families. We build up a confusion matrix to analyze the performance of Mal-ASSF under different classification tasks. We perform experimen-

tal comparisons of different sequence lengths and determine whether to deduplicate. We compare Mal-ASSF with machine learning and other deep learning methods for classifying malicious code. We conduct ablation experiments to verify the effectiveness of each module in Mal-ASSF. The experimental results show that it achieves higher detection accuracy than related work, especially in the case of malware family classification and newer malware samples.

The rest of this paper is organized as follows: Section 2 reviews related work on malware detection based on dynamic analysis; Section 3 describes our proposed model in detail; Section 4 presents our experimental setup and results; Section 5 discusses the advantages and limitations of our approach, and concludes this paper and suggests future work.

2. Related Work

Dynamic Malware Analysis. API call sequences are considered a representative technique for understanding the behavioral characteristics of malware. To obtain API call sequences, the malicious code is executed within a regulated environment such as Cuckoo or Virmon sandboxes [18]. Throughout the execution process, dynamic behaviors are continuously observed and subsequently traced. Typically, malware requires the launch of multiple processes to accomplish specific behaviors. The process may additionally entail the utilization of multiple threads. From the perspective of the API sequence of a single thread, it may be benign, but the API sequence of all threads as a whole may pose risks. Dynamic analysis considers the API calls of all related threads as a whole. The report generated from dynamic analysis outlines the conduct of malicious executables during their operation on affected hosts. This behavior can encompass downloading infected files from the Internet, performing harmful OS tasks, altering or erasing files, configuring or modifying system registries, remotely extracting vulnerable data, preventing authorized users from accessing resources, and slowing down the network, among other potential activities.

Dynamic Malware Analysis Based on Feature Engineering and Machine Learning. From these dynamic features, machine learning models, including K-Nearest Neighbors (KNN), Naive Bayes (NB), Decision Trees (DT), and Support Vector Machines (SVM), are utilized to classify the samples. [19–23]. Based on the sequence of API calls, a C4.5 decision tree was applied to build a supervised learning model that distinguishes malware from benign files [19]. Singh et al. [22] extracted API calls as well as other behaviors such as file operations, registry modifications, and network activities and used a random forest (RF) method for classification. AL Ahmadi [23] built a supervised malware classifier based on KNN and RF by mining features from network flow sequences. Despite their efficacy on the test set, these approaches heavily depend on feature engineering and necessitate the expertise of practitioners to select and refine suitable features. With the evolution of malicious code and the improvement of resistance capabilities, the refinement of feature engineering becomes an ongoing necessity.

Dynamic malware analysis based on deep learning. Deep learning has the capability to detect sequence features and autonomously identify malicious behaviors [24]. Therefore, it is an effective tool for tackling the issue of malware classification [6,25,26].

Vinayakumar et al. [25] proposed a DNN-based model to develop a flexible and effective IDS to detect and classify unforeseen and unpredictable cyberattacks. Their research paper has been widely cited as it introduced deep learning techniques to the field of dynamic malware analysis. However, the model is insufficient for representing the sequential dependencies among the API calls made by malware. Jha et al. [27] used RNN and word2vec to identify malware instances. The experimental outcomes demonstrated that optimal classification accuracy is attainable when utilizing relatively short sequence features (50–500). Conversely, inadequate classification performance was observed with long sequences. Catak et al. [28] used word embeddings and a two-tier LSTM model to capture the correlation among API calls in sequences. To carry out a control experiment,

they employed the TF-IDF method and vectorized the textual dataset using term frequency and inverse document frequency techniques. The experimental outcomes demonstrated that LSTM-based malware classification performed better than conventional machine learning algorithms, such as TF-IDF-based classification. However, this approach employs a unidirectional LSTM and only considers historical API relationships while ignoring API information in the future. Abusnaina et al. [29] investigated the behavioral features of Windows APIs invoked by malware during runtime. They introduced the BiLSTM network and utilized it to model the sequential dependencies between API calls of malware in both forward and backward directions. The model presented demonstrated favorable effectiveness in classifying malware families and exhibits the capability to combat heuristic-based malware that leverages uncorrelated API sequence embedding. However, the sequence length of the dataset in this method is too short to objectively describe the API calls in the real world. All these methods above directly use the sequence information of the API for behavior modeling and do not take the semantic relationship between APIs into consideration.

Amer et al. [30] used word embedding to capture the contextual relationship that exists between API functions in malware call sequences. Based on the contextual similarity, related APIs are clustered, and a simple behavioral graph is constructed to characterize malware. To detect such malware, they utilized a Markov chain model. Their work is instructive in proposing a technique for representing the semantic characteristics of API functions within a contextual framework. Kang et al. [17] proposed a word2vec-driven technique for word embedding to evaluate opcodes and API function names with fewer dimensions. Every API incorporates a substantial amount of semantic information, including but not limited to categories, operations, parameters, and return values. This information can facilitate detection models comprehension of API sequences with greater precision. Zhang et al. [30] proposed a predefined method to transform the API call semantic vector. This technique employs a hash method, which is capable of extracting information pertaining to the API call's name, category, and parameters. The latest research has been conducted by Daeef et al. [31] to implement deep graph convolutional neural networks (DGCNNs) for malware analysis. However, the result showed that it performed quite similarly to the traditional method of RF. All these methods are inadequate in terms of extracting API features as well as including excessive superfluous information, ultimately resulting in ineffective identification and the learning of implicit data pertaining to function APIs.

Dynamic malware detection based on semantic and sequential features of API call sequences. Chen et al. [32] present a deep neural network-based malware detection approach where API call sequences are augmented with parameters. Clustering-based classification is employed to select malicious behavior-sensitive parameters. Through this method, the semantic relationship of APIs in terms of security is presented. Furthermore, Li et al. [33] extract semantic features from API names and arguments and adopt an API call graph to convert the relationship between API calls into the structural information of the graph. Balan et al. [34] present an ANN-based model focusing on sequence-pattern feature mining. Their experimental results show that the sequence pattern of OS API-related call sequences plays an important role in malware detection. Nawa et al. [35] also explore the sequential pattern of API call sequences, and maximal and closed frequent API call patterns are utilized for malware classification. Compared to these works, in this paper we not only focus on exploring the semantic features of API functions but also integrate sequential features to obtain better performance with the classification of families of malware.

In summary, current work on dynamic feature analysis using functional calls of API sequences is inadequate for accurately representing the behavior patterns of malicious families. The performance of the existing classification approach fails to fulfill the requirements for accurately classifying various families of malicious codes. In Mal-ASSF, we fuse the semantic and sequence features and adopt a self-attention mechanism to enhance the effectiveness of malware family detection in the case of an excessive number of API types, oversized feature sets, and excessively long sequences. The experimental results reveal that

Mal-ASSF has achieved a detection enhancement of approximately 3% to 5% in comparison to contemporary methods for dynamic analysis.

3. Methodology

In this section, we introduce the key design of Mal-ASSF to determine the API behavior for dynamic malware analysis.

3.1. Overview

The overall structure of the Mal-ASSF model is shown in Figure 1. The input of the Mal-ASSF framework is a well-labeled API sequence (for the stage where the trained model is used) or an unknown sample (for the stage where the trained model is used for classification). API sequences are extracted from the operation analysis report of executable file samples. API sequences are processed in parallel by the sequence feature module and the TextRNN feature module. The API sequence is converted into two-dimensionally reduced sequences. Then, the classification results will be drawn through the fusion module, the attention module, and the final MLP classifier module.

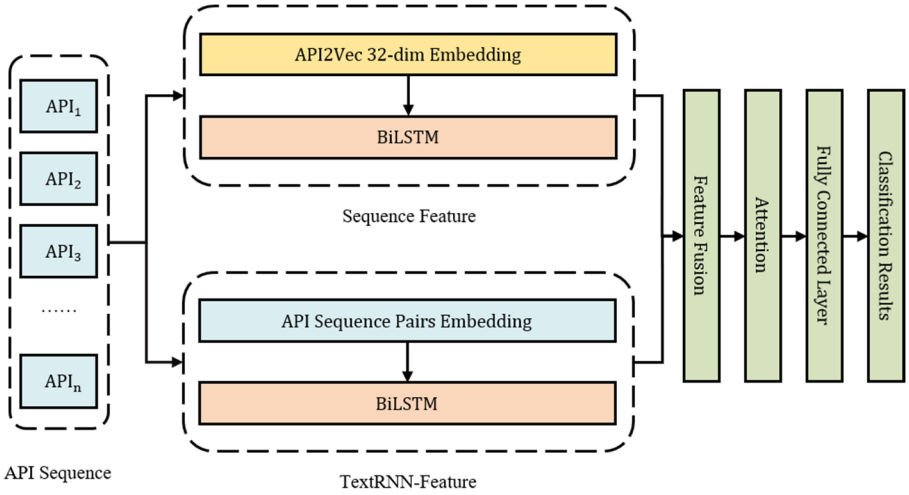


Figure 1. Overall architecture of the Mal-ASSF framework.

In the ***preparation process***, we run the executable file samples in a security sandbox environment to obtain the operation analysis report in JSON format. We extract the API call information, which is recorded in order, from the report.

In the ***sequence feature module***, to represent API behavior, we adopt API2Vec embedding and BiLSTM. Through dimensionality reduction and convolution operations, functional behavior with different lengths can be captured. The APIs are then converted into a sequential chain.

In the ***TextRNN feature module***, the API semantic chain of function names is transferred into a vectorized representation containing operation type encoding and operation object encoding by the embedding layer, and the implicit semantic features are extracted by the BiLSTM layer.

In the ***fusion module***, the outputs of the API embedding module and the API implicit semantic extraction module are concatenated together to obtain all the features of the API sequences.

In the ***attention module***, the attention mechanism is used to calculate the weight corresponding to each positioning element so that the model can focus on some malicious segments.

In the ***MLP classifier module***, we apply a fully connected layer MLP for classification. Dropout is adopted to reduce overfitting. Softmax is used to calculate the probability of malware classification. The module uses an Adam optimizer with binary cross-entropy as the loss function.

The subsequent paragraphs of this section will provide a detailed account of the data processing mechanism.

3.2. Detailed Archetecture

3.2.1. Data Preparation

In the preparation process, the API call information is extracted from the operation analysis report. However, the API call information should be preprocessed. To evade existing feature detection, malicious code is often inserted into benign code or obfuscated with a considerable number of APIs with no actual function. Therefore, the API call sequence obtained by sandbox sampling must be of sufficient length (thousands at least), which means the length of the API call information samples far exceeds common sequence length for classification problems such as emotion recognition and comment classification. Additionally, there are some short-call sequence samples for classification.

Therefore, truncation and padding operations should be performed for the API sequences to be trimmed to the same length. The data preparation provides convenience for the subsequent process.

3.2.2. Vectorized Representation Based on API2Vec

To express the relationship between functions in API sequences and reduce the amount of computation, we map the API call sequences from strings into vectors.

One-hot encoding is well known as a traditional method for vectorization. However, this method is associated with several problems. Firstly, the relationship between APIs is independent through one-hot encoding, which means that the lack of contextual connections is evident. Additionally, the feature vectors will be quite sparse and bring challenges to the subsequent process. To address this problem, we use an API2Vec vectorization method based on Word2Vec to perform word embedding operations on APIs.

In the API2Vec vectorization method, we vectorize string data through a shallow neural network, and the semantic information of the words is represented in the form of word vectors. Through this approach, we reduce the dimensionality of word vectors, and the word vectors corresponding to APIs with similar semantics are also similar. For example, the system functions NtOpenFile and NtReadFile both belong to file operations. NtOpenFile is used to open a file, and NtReadFile is used to read a file, which means the two functions are semantically similar. Then, in the new multi-dimensional space, the expressed word vector should also be relatively close. On the contrary, the RegOpenKeyEx function is used to open a specified registry key, which is different from the former file operation functions, and the distance of the word vectors is relatively far.

The training process of API2Vec is shown in Figure 2. The detailed illustrations are as follows:

- We first encode the API information through one-hot encoding to obtain a high-dimensional sparse feature vector.
- We then train a shallow neural network to obtain the hidden layer weight of each API. The input layer and the hidden layer weights are jointly calculated to obtain the output vector, which is also the word vector form of the API and can uniquely identify each API. We use the Skip-Gram model for weight training due to the relatively small association between APIs. Assuming that an API is represented by a 32-dimensional vector feature, for 295 kinds of APIs, compared to the 295-dimensional one-hot encoding form, the API2Vec method can be used to reduce the dimension of the word vector to 32 dimensions.
- The word vector representation of each API name with a fixed dimension is finally calculated and stored in the form of a dictionary.

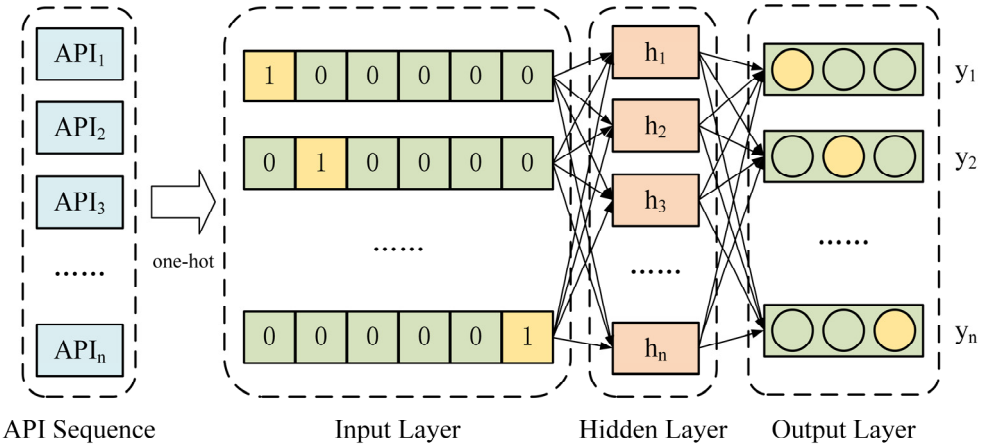


Figure 2. Schematic diagram of the API2Vec training process.

Through the vectorization of the name of each function in API sequences, the distance relationship between functions can be better expressed, and the amount of computation is reduced through moderate dimensionality reduction, which facilitates the model to efficiently learn features in a specific functional sequence.

3.2.3. API Implicit Semantic Sequence Feature Extraction

To capture the implicit information behind APIs, we use triples for descriptions. Assuming that P_i is a pair representation of the i-th API in the sequence, then the representation of P_i can be shown in Equation (1).

$$P_i =< O, T > \tag{1}$$

where O stands for ***operation*** and T stands for ***type***.

Operation refers to the actions of APIs. According to the analysis of common functions, we extracted about 50 verbs from common API names to establish the operation set shown in Table 1. Elements of the set basically outline the core operations of API function calls. The operations can be extracted and stored in the form of a dictionary through a string extraction algorithm.

Table 1. The operation set of APIs.

Operation Set
'allocate', 'accept', 'bind', 'close', 'compress', 'connect, 'control', 'copy', 'crack', 'decode', 'decrypt', 'delete', 'download', 'draw', 'encode', 'exec', 'exit', 'export', 'free', 'find', 'get', 'hash', 'initialize', 'listen', 'ls', 'load', 'lookup', 'make', 'map', 'move', 'open', 'put', 'query', 'read', 'recv', 'register', 'remove', 'save', 'search', 'send', 'select', 'set', 'shutdown', 'socket', 'start', 'suspend', 'unhook', 'unload', 'write'

Type refers to the API classification given in the Cuckoo sandbox. The APIs can be divided into over 10 types, including system, network, process, registry, interface, and others. The detailed type list is shown in Table 2.

Table 2. The types of APIs.

Type Set
'system', 'network', 'process', 'file', 'registry', 'service', 'ui', 'crypto', 'ole', 'exception', 'none', 'certificate', 'misc', 'netapi', 'resource', 'iexplore'

According to the descriptions above, each function of the API sequence can be described as a pair P, according to the implicit information. Examples are shown in Table 3.

Table 3. Examples of API implicit semantic extraction.

API	Operation	Type
RegOpenKeyExW	open	registry
RegQueryValueExW	query	registry
NtCreateThreadEx	create	process
GetSystemInfo	get	system
DeleteFileW	delete	file

The pairs formed by each API are connected in sequence to form a pair sequence to generate an implicit semantic chain of sequence information. The process of converting each sample into a semantic chain and performing feature mapping is shown in Figure 3.

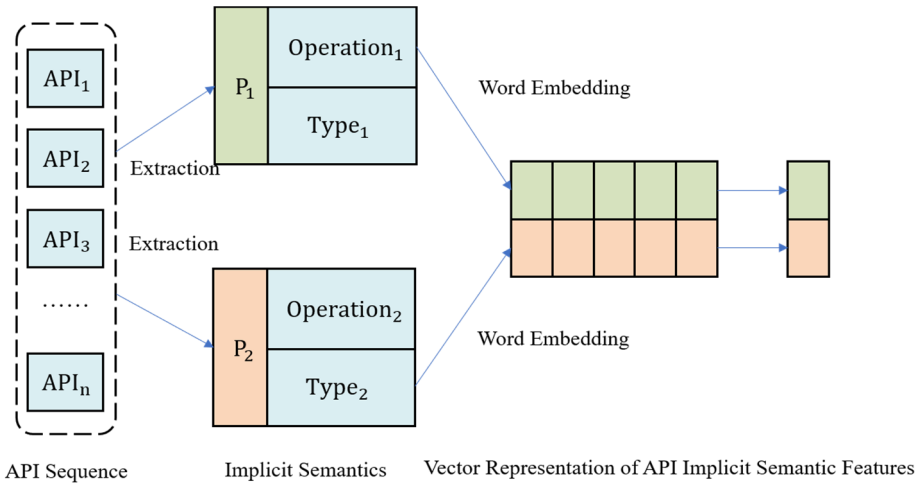


Figure 3. Schematic diagram of API implicit semantic feature mapping.

3.2.4. Sequence Feature Extraction Based on BiLSTM

Mal-ASSF adopts BiLSTM [36] to capture the complex calling relationship between APIs. The two LSTMs are connected, and each input API sequence feature will be trained and learned by the network in both forward and reverse directions. The network weights of BiLSTM are shared, which can accurately reflect the sequence features of malicious code families. Figure 4 shows the schematic diagram of BiLSTM unfolding at time $t-1, t, t+1$ according to the time axis, where x is the input, h is the hidden layer state, and y is the current output.

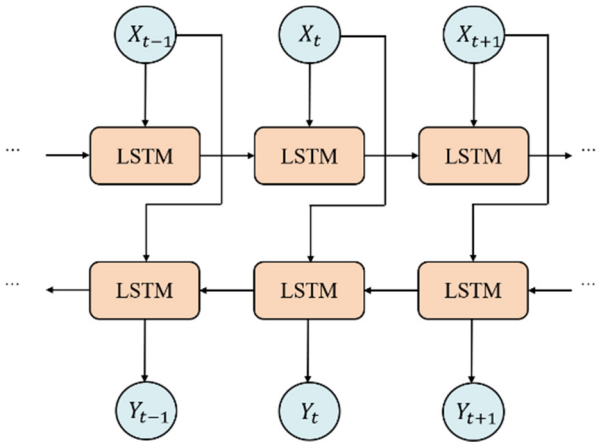


Figure 4. Schematic diagram of the BiLSTM model.

The two directions of BiLSTM at time t can be calculated as shown in Equations (2) and (3).

$$\overrightarrow{h_t} = \overrightarrow{LSTM}\left(x_t, \overrightarrow{h_{t-1}}\right) \tag{2}$$

$$\overleftarrow{h_t} = \overleftarrow{LSTM}\left(x_t, \overleftarrow{h_{t+1}}\right) \tag{3}$$

After processing the features of API sequences and implicit semantics by embedding operations, these vectorized data can be input into the BiLSTM networks.

The API embedding and BILSTM convolution operations within two parallel modules actually extract the sequence features and TextRNN features of API sequences, respectively. While learning API sequential relationships and semantic features, Mal-ASSF can also pay attention to the calling sequence. Certain types of malicious code families have distinct sequence features, so using the BiLSTM network to capture the sequence features of API calls is helpful for the identification of malicious code family features.

3.2.5. Key API Sequence Recognition Based on the Attention Mechanism

Mal-ASSF applies an attention module for the recognition of key API sequences. The module consists of three computational stages, as follows:

(1) The dot product is used to calculate the correlation between the two sides, as shown in Equation (4).

$$Sim(Query, Key_i) = Query \cdot Key_i \tag{4}$$

(2) The Softmax function is introduced to numerically transform the scores calculated in the first stage. As shown in Equation (5), normalization is performed, and important factor weights are highlighted.

$$a_i = Softmax(Sim_i) = \frac{e^{Sim_i}}{\sum_{j=1}^{L_x} e^{Sim_j}} \tag{5}$$

(3) The a_i calculated in the second stage is the weight coefficient corresponding to $Value_i$. The attention value can be calculated by weighted summation. In this way, the attention value of each element can be calculated. as shown in Formula (6).

$$Attention(Query, Source) = \sum_{i=1}^{Lx} a_i * Value_i \tag{6}$$

Mal-ASSF regards the constituent elements in the data as a series of $< Key, Value >$ data pairs. Given an element $Query$ in the target, by calculating the similarity between $Query$ and each Key, the weight coefficient of each Key corresponding to the value can be calculated. After the weighted summation of the values, the final attention value can be obtained. Figure 5 is a computational process diagram of the attention mechanism.

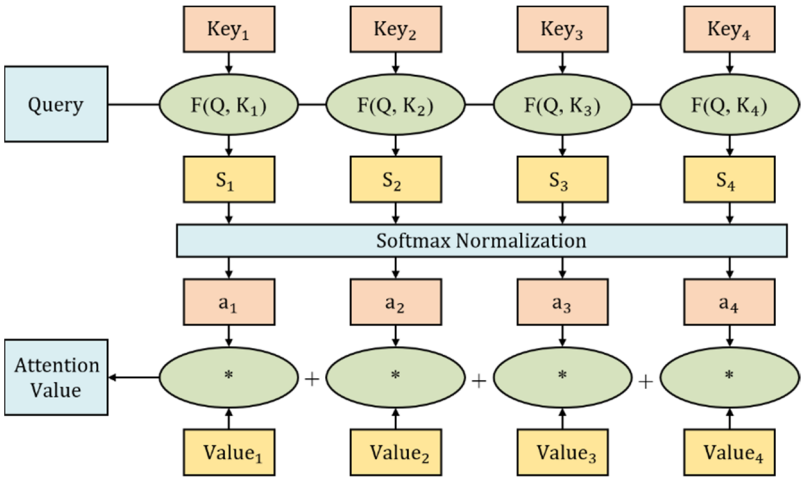


Figure 5. Schematic diagram of the calculation process of the attention mechanism.

4. Experiment

In this section, we conduct a series of experiments to verify the effectiveness of our Mal-ASSF model.

- A confusion matrix is built to analyze the performance of Mal-ASSF under different classification tasks.
- The sequence length and whether to deduplicate the adjacent API may affect the performance of Mal-ASSF. Therefore, this paper performs experimental comparisons of different sequence lengths and determines whether to deduplicate.
- Mal-ASSF is compared with other methods for classifying malicious code based on API sequences, including deep learning models such as TextCNN, CNN-LSTM, and TextCNN-LSTM, as well as machine learning methods such as K-Nearest Neighbors (KNN), support vector machine (SVM), decision tree (DT), random forest (RF), and gradient boosting (XGB). For the comparison method, we use publicly available algorithms to run on the same dataset and evaluate the experimental results.
- Ablation experiments are performed to verify the effectiveness of each module in Mal-ASSF. The performance of Mal-ASSF is compared with that of a model using only one-hot encoding for word embedding, a model using one-way LSTM, and a model without an attention mechanism.

4.1. Dataset

In our experiment, we adopt the dataset from the Alibaba Cloud Security Malware Detection Competition [37], where the benign samples are collected from regular software stores and the malicious samples are collected from the Internet and virus databases. The dataset includes 12,390 samples from more than 70 million API calls after data filtering and cleaning. The malicious samples include infectious viruses, Trojans, mining programs, ransomware, and others, and the labels have been verified by the Virus Total platform. It should be noted that the official only provides samples in the form of API instruction sequences, which have been desensitized to ensure that these data will not be used by attackers with malicious motives.

The category and number of samples are shown in Table 4. There are 7 types of samples, and the normal category accounts for the largest proportion, accounting for 40% of all. Worm viruses accounted for the least, at only 0.8%. There is a problem of data imbalance in each category of samples, which is of great significance for testing the classification effect of the model.

Table 4. The quantitative relationship and category attribution of malicious program samples.

Label	Category Attribution	Quantity
0	Normal software	4978
1	Ransomware	502
2	Mining program	1196
3	DDoS attack	820
4	Worm virus	100
5	Infectious virus	4279
6	Backdoor	515

Shown as Table 5, the table of the dataset includes the API sequences of all samples. Each record includes 5 fields, *file_id*, label, API, tid, and index. All records of one sample have the same file_id. The label field takes different values to represent benign samples or different malware families. One malware or benign sample occupies thousands of records in the table. The field API means the textual name of the API function. The 'tid' field represents the thread that called the API function. The 'Index' field represents the sequence number of the API call in the same thread. The APIs inside the thread are arranged according to the index number, and there is a sequential relationship. If there is multithreading concurrency, API functions will not be sorted by thread sequence number. According to statistics, the API call sequence in the dataset contains a total of 295 different API names.

Table 5. Alibaba Cloud dataset inclusion list.

File_id	Label	API	Tid	Index
5	0	SetErrorMode	2500	0
5	0	LdrGetDllHandle	2500	1
5	0	LdrGetProcedureAddress	2500	2
5	0	GetSystemDirectoryA	2500	3
...	...	...	...	...
5	0	SetErrorMode	2596	0
5	0	LdrGetDllHandle	2596	1

4.2. Results and Analysis

4.2.1. Confusion Matrix

Figure 6 illustrates the confusion matrix of the Mal-ASSF model proposed in this paper for the seven-class classification of malicious code types.

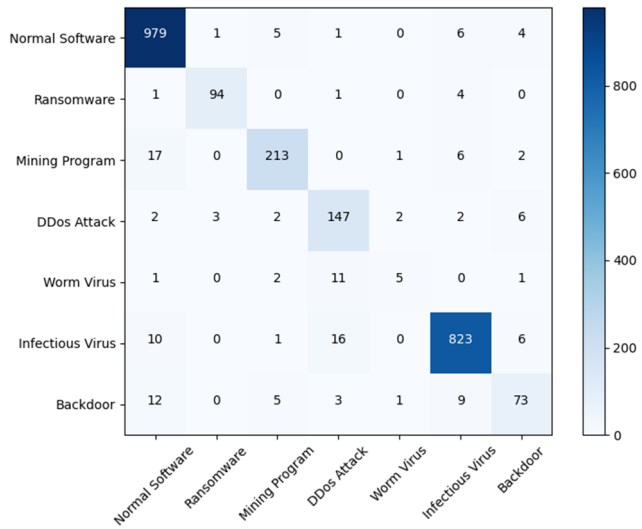


Figure 6. Multiclass confusion matrix of the Alibaba Cloud dataset.

The confusion matrix indicates that Mal-ASSF achieves high performance in most categories. A small number of DDoS viruses are mistakenly classified as worm viruses. This can be attributed to the fact that worm viruses have a similar mechanism of action as DDoS viruses, and feature learning is insufficient due to the limited number of training samples for worm viruses. Furthermore, some benign software samples are misclassified as malicious categories, such as mining programs and backdoors, because they contain mechanisms that resemble malicious behavior.

4.2.2. Effect of the API Sequence Length on the Performance of the Mal-ASSF Model

Figure 7 demonstrates the impact of varying sequence lengths on accuracy.

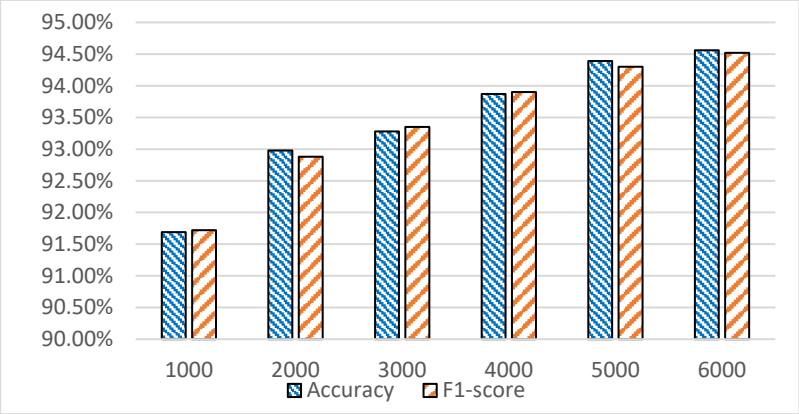


Figure 7. Accuracy with different sequence lengths.

It can be concluded that within the range of 1000 to 6000, increasing the sequence length enables more sequence features to be captured and thus enhances accuracy. Within the range of 5000 to 6000, the accuracy gain is marginal (0.17%), while the cost of sampling and training has become very high. Hence, the optimal sequence length for the experiments in this paper is 5000.

4.2.3. Comparison with API Frequency-Based Machine Learning Models

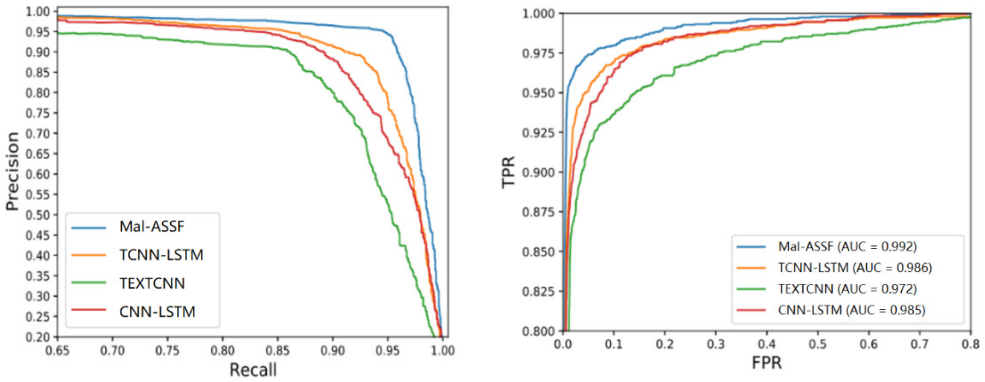
The proposed Mal-ASSF model achieves an accuracy of 94.49%, a precision of 94.01%, a recall of 94.19%, and an F1-score of 0.9402 on the test set. Table 6 demonstrates that the classification performance is markedly superior to other machine learning models. This indicates that the classification based on the API frequency in the sequence fails to capture the features in the sequence adequately, resulting in a relatively low multi-classification performance. The classification accuracies of KNN, SVM, and DT are generally below 90%. RF and XGB employ the concept of ensemble learning to enhance accuracy to some degree, achieving 90.4% and 91.8%, respectively, which illustrates the effectiveness of feature integration. Mal-ASSF also leverages the concept of feature combinations. The integration of API correlation and implicit semantic features enables the implicit model to attain optimal classification outcomes.

Table 6. Comparison with machine learning model results.

Model	Accuracy	Precision	Recall	F1
KNN	86.46%	87.85%	85.76%	0.8681
SVM	88.46%	89.85%	85.76%	0.8775
DT	89.78%	89.17%	89.13%	0.8914
RF	9 0.45%	88.95%	91.54%	0.9023
XGB	9 1.84%	91.95%	91.54%	0.9177
Mal-ASSF	94.49%	94.01%	94.19%	0.9402

4.2.4. Comparison with Other Deep Learning Models

Figure 8a depicts the comparison between Mal-ASSF and other models. The precision-recall curve is plotted with the precision value on the vertical axis and the recall value on the horizontal axis. The curvature located in close proximity to the upper-right quadrant suggests a positive relationship between the recall rate and the precision rate, whereby an increase in the recall rate is associated with a corresponding increase in the precision rate.



(**a**) PR curve (**b**) ROC curve

Figure 8. Comparison of the PR and ROC curves of different deep learning models.

The PR curve diagram illustrates that, in comparison to other models examined, the Mal-ASSF model proposed in this study exhibits significant superiority. Figure 8b depicts the ROC curves of the Mal-ASSF model in contrast with other models investigated in this study. The PR curve diagram analysis reveals that the Mal-ASSF model displays superior classification performance, as indicated by its closer proximity to the upper left corner of the diagram. This finding is substantiated by the Mal-ASSF model's achieved AUC value of 0.992, which exceeds those of the comparison models, specifically 0.986 for TCNN-LSTM, 0.972 for TextCNN, and 0.985 for CNN-LSTMs.

Table 7 presents the specific comparative outcomes between the Mal-ASSF model and other deep learning models. According to the results presented in Table 7, the Mal-ASSF model outperformed other models with an accuracy rate of 94.49% in accurately classifying malicious code families. Additionally, the Mal-ASSF model demonstrated superior precision rate, recall rate, and F1-score value compared to other models. The observed accuracy rate of TextCNN amounts to merely 8.69%, which suggests that the utilization of myriad convolution kernels is imperative for effective feature extraction from sequential data. The CNN-LSTM and TextCNN-LSTM models exhibited an accuracy rate that surpassed 90%, underscoring the significance of preserving the sequential order relationship.

Table 7. Comparison in the results of the deep learning model.

Model Composition	Accuracy	Precision	Recall	F1
TEXTCNN	88.69%	88.59%	88.69%	0.8864
CNN-LSTM	90.23%	89.64%	90.23%	0.8993
TCNN-LSTM	92.13%	92.17%	92.13%	0.9215
Mal-ASSF	94.49%	94.01%	94.19%	0.9402

4.2.5. Ablation Experiment

In this study, the validation of the efficacy of individual modules within the fusion model is performed via ablation experiments.

In Table 8, the record named "one hot" indicates using the one-hot encoding method to replace the API2Vec module. The findings delineated indicate the API2Vec module has the

most significant effect on the overall accuracy rate, while solely API encoding with one-hot representation yields an accuracy rate of merely 87%. The incorporation of API implicit semantics yields a noteworthy improvement of 2.1% in accuracy, thereby suggesting the presence of additional features in the sequence data that are open to discovery. The enhancement produced by BiLSTM exhibits a marginal impact of only 1.3%, suggesting that the information acquired by the forward sequence alone is capable of capturing a substantial number of features within the time series data. It can be observed that the utilization of attention mechanisms results in an enhancement of the accuracy rate by approximately 2%. The results of the ablation experiments demonstrate that each module exhibits a discernible increase in the accuracy rate. The Mal-ASSF model proposed in this research exhibits superior fusion capability and achieves peak levels of accuracy, precision, recall, and F1 score.

Table 8. Comparison of the results of ablation experiments.

Ablation Part	Accuracy	Precision	Recall	F1-Score
one-hot	87.22%	87.63%	87.22%	0.8721
nosemantic	91.12%	91.21%	91.12%	0.9116
nobilstm	92.75%	92.55%	92.75%	0.9265
no-attention	92.13%	92.17%	92.13%	0.9215
Mal-ASSF	94.49%	94.01%	94.19%	0.9402

5. Conclusions

To address the problem that the dynamic feature set is too large to discover the key hidden information, we propose a dynamic malware detection model, Mal-ASSF, based on the fusion of sequential features and semantic features of API call sequences. In Mal-ASSF, the API functions are represented by dimensionality reduction through word embedding. The sequential behavioral features of different length segments are captured by multiple convolution kernels. The information of the function API is depicted by a triplet of action, type, category, and other information.

Mal-ASSF has advantages in the semantic representation and key sequence recognition abilities of API call sequences. Compared with traditional machine learning methods such as KNN, SVM, DT, RF, and XGB, the detection performance of Mal-ASSF is improved by three to eight percentage points; compared with deep learning models such as TextCNN, CNN + LSTM, and TextCNN + LSTM, the performance of the Mal-ASSF model is also improved by two to five percentage points. However, through the ablation experiment, it can be concluded that the improvement of the BiLSTM module is not particularly obvious. Further studies can be conducted to obtain a better representation of semantic features using pre-trained models.

Author Contributions: Conceptualization, S.Z. and W.Y.; methodology, S.Z. and M.Z.; software, J.W. and M.Z.; validation, M.Z. and J.W.; formal analysis, J.W.; investigation, J.W.; resources, W.Y.; data curation, M.Z.; writing—original draft preparation, S.Z.; writing—review and editing, J.W.; visualization, M.Z.; supervision, W.Y.; project administration, W.Y.; funding acquisition, W.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key Research and Development Program, grant number 2018YFB1003800.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Al-rimy, B.A.S.; Maarof, M.A.; Shaid, S.Z.M. Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. *Comput. Secur.* **2018**, *74*, 144–166. [CrossRef]
2. Mcafee Labs Threats Report. 2021. Available online: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-threats-jun-2021.pdf (accessed on 25 March 2022).
3. VirusTotal File Statistics. 2020. Available online: https://www.virustotal.com/en/statistics (accessed on 20 June 2021).
4. Ye, Y.; Li, T.; Adjeroh, D.; Iyengar, S.S. A Survey on Malware Detection Using Data Mining Techniques. *ACM Comput. Surv.* **2017**, *50*, 1–40. [CrossRef]
5. Mohanta, A.; Saldanha, A. *Malware Analysis and Detection Engineering: A Comprehensive Approach to Detect and Analyze Modern Malware*; Springer: Berlin/Heidelberg, Germany, 2020.
6. Aghakhani, H.; Gritti, F.; Mecca, F.; Lindorfer, M.; Ortolani, S.; Balzarotti, D.; Vigna, G.; Kruegel, C. When Malware is Packin' Heat; Limits of Machine Learning Classifiers Based on Static Analysis Features. In Proceedings of the Network and Distributed Systems Security (NDSS) Symposium 2020, San Diego, CA, USA, 23–26 February 2020.
7. Mantovani, A.; Aonzo, S.; Ugarte-Pedrero, X.; Merlo, A.; Balzarotti, D. Prevalence and Impact of Low-Entropy Packing Schemes in the Malware Ecosystem. In Proceedings of the NDSS, San Diego, CA, USA, 23–26 February 2020.
8. Sahay, S.K.; Sharma, A.; Rathore, H. Evolution of Malware and Its Detection Techniques. In Proceedings of the Information and Communication Technology for Sustainable Development, New Delhi, India, 27–28 February 2020; pp. 139–150.
9. Mehrabi Koushki, M.; AbuAlhaol, I.; Raju, A.D.; Zhou, Y.; Giagone, R.S.; Shengqiang, H. On building machine learning pipelines for Android malware detection: A procedural survey of practices, challenges and opportunities. *Cybersecurity* **2022**, *5*, 16. [CrossRef]
10. Carlin, D.; O'Kane, P.; Sezer, S. A cost analysis of machine learning using dynamic runtime opcodes for malware detection. *Comput. Secur.* **2019**, *85*, 138–155. [CrossRef]
11. Nguyen, T.N.; Ngo, Q.-D.; Nguyen, H.-T.; Nguyen, G.L. An Advanced Computing Approach for IoT-Botnet Detection in Industrial Internet of Things. *IEEE Trans. Ind. Inform.* **2022**, *18*, 8298–8306. [CrossRef]
12. Rosenberg, I.; Shabtai, A.; Rokach, L.; Elovici, Y. Generic Black-Box End-to-End Attack Against State of the Art API Call Based Malware Classifiers. In Proceedings of the Research in Attacks, Intrusions, and Defenses, Crete, Greece, 10–12 September 2018; pp. 490–510.
13. Qiang, W.; Yang, L.; Jin, H. Efficient and Robust Malware Detection Based on Control Flow Traces Using Deep Neural Networks. *Comput. Secur.* **2022**, *122*, 102871. [CrossRef]
14. Rosenberg, I.; Shabtai, A.; Elovici, Y.; Rokach, L. Query-Efficient Black-Box Attack Against Sequence-Based Malware Classifiers. In Proceedings of the Annual Computer Security Applications Conference, Austin, TX, USA, 7–11 December 2020; pp. 611–626.
15. Huang, W.; Stokes, J.W. MtNet: A Multi-Task Neural Network for Dynamic Malware Classification. In Proceedings of the Detection of Intrusions and Malware, and Vulnerability Assessment, San Sebastián, Spain, 7–8 July 2016; pp. 399–418.
16. Wang, Q.; Qian, Q. Malicious code classification based on opcode sequences and textCNN network. *J. Inf. Secur. Appl.* **2022**, *67*, 103151. [CrossRef]
17. Kang, J.; Jang, S.; Li, S.; Jeong, Y.-S.; Sung, Y. Long short-term memory-based Malware classification method for information security. *Comput. Electr. Eng.* **2019**, *77*, 366–375. [CrossRef]
18. Pektaş, A.; Acarman, T. Classification of malware families based on runtime behaviors. *J. Inf. Secur. Appl.* **2017**, *37*, 91–100. [CrossRef]
19. Suaboot, J.; Tari, Z.; Mahmood, A.; Zomaya, A.Y.; Li, W. Sub-curve HMM: A malware detection approach based on partial analysis of API call sequences. *Comput. Secur.* **2020**, *92*, 101773. [CrossRef]
20. Jan, N.; Gwak, J.; Pei, J.; Maqsood, R.; Nasir, A. Analysis of Networks and Digital Systems by Using the Novel Technique Based on Complex Fuzzy Soft Information. *IEEE Trans. Consum. Electron.* **2022**, *69*, 183–193. [CrossRef]
21. Fan, M.; Liu, J.; Luo, X.; Chen, K.; Tian, Z.; Zheng, Q.; Liu, T. Android Malware Familial Classification and Representative Sample Selection via Frequent Subgraph Analysis. *IEEE Trans. Inf. Forensics Secur.* **2018**, *13*, 1890–1905. [CrossRef]
22. Singh, J.; Singh, J. Detection of malicious software by analyzing the behavioral artifacts using machine learning algorithms. *Inf. Softw. Technol.* **2020**, *121*, 106273. [CrossRef]
23. AlAhmadi, B.A.; Martinovic, I. MalClassifier: Malware family classification using network flow sequence behaviour. In Proceedings of the 2018 APWG Symposium on Electronic Crime Research (eCrime), San Diego, CA, USA, 15–17 May 2018; pp. 1–13.
24. Cui, Z.; Xue, F.; Cai, X.; Cao, Y.; Wang, G.-G.; Chen, J. Detection of Malicious Code Variants Based on Deep Learning. *IEEE Trans. Ind. Inform.* **2018**, *14*, 3187–3196. [CrossRef]
25. Vinayakumar, R.; Alazab, M.; Soman, K.P.; Poornachandran, P.; Al-Nemrat, A.; Venkatraman, S. Deep Learning Approach for Intelligent Intrusion Detection System. *IEEE Access* **2019**, *7*, 41525–41550. [CrossRef]
26. Jha, S.; Prashar, D.; Long, H.V.; Taniar, D. Recurrent neural network for detecting malware. *Comput. Secur.* **2020**, *99*, 102037. [CrossRef]
27. Catak, F.O.; Yazı, A.F.; Elezaj, O.; Ahmed, J. Deep learning based Sequential model for malware analysis using Windows exe API Calls. *PeerJ Comput. Sci.* **2020**, *6*, e285. [CrossRef] [PubMed]

28. Abusnaina, A.; Abuhamad, M.; Alasmary, H.; Anwar, A.; Jang, R.; Salem, S.; Nyang, D.; Mohaisen, D. DL-FHMC: Deep Learning-Based Fine-Grained Hierarchical Learning Approach for Robust Malware Classification. *IEEE Trans. Dependable Secur. Comput.* **2022**, *19*, 3432–3447. [CrossRef]
29. Amer, E.; Zelinka, I. A dynamic Windows malware detection and prediction method based on contextual understanding of API call sequence. *Comput. Secur.* **2020**, *92*, 101760. [CrossRef]
30. Zhang, Z.; Qi, P.; Wang, W. Dynamic Malware Analysis with Feature Engineering and Feature Learning. *Proc. AAAI Conf. Artif. Intell.* **2020**, *34*, 1210–1217. [CrossRef]
31. Daeef, A.Y.; Al-Naji, A.; Nahar, A.K.; Chahl, J. Features Engineering to Differentiate between Malware and Legitimate Software. *Appl. Sci.* **2023**, *13*, 1972. [CrossRef]
32. Chen, X.; Hao, Z.; Li, L.; Cui, L.; Zhu, Y.; Ding, Z.; Liu, Y. CruParamer: Learning on Parameter-Augmented API Sequences for Malware Detection. *IEEE Trans. Inf. Forensics Secur.* **2022**, *17*, 788–803. [CrossRef]
33. 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.* **2022**, *122*, 102872. [CrossRef]
34. Balan, G.; GavriluŢ, D.T.; Luchian, H. Using API Calls for Sequence-Pattern Feature Mining-Based Malware Detection. In Proceedings of the Information Security Practice and Experience, Taipei, Taiwan, 23–25 November 2022; pp. 233–251.
35. Nawaz, M.S.; Fournier-Viger, P.; Nawaz, M.Z.; Chen, G.; Wu, Y. MalSPM: Metamorphic malware behavior analysis and classification using sequential pattern mining. *Comput. Secur.* **2022**, *118*, 102741. [CrossRef]
36. Agrawal, R.; Stokes, J.W.; Marinescu, M.; Selvaraj, K. Neural Sequential Malware Detection with Parameters. In Proceedings of the 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Calgary, AB, Canada, 15–20 April 2018; pp. 2656–2660.
37. Alibaba Cloud Malware Detection Based on Behaviors. 2021. Available online: https://tianchi.aliyun.com/competition/entrance/231694/information (accessed on 20 June 2021).