*electronics*

MDPI

*Article*

# Dynamic Malware Detection Using Parameter-Augmented Semantic Chain

**Donghui Zhao [1], Huadong Wang [2], Liang Kou [1,*], Zhannan Li [1] and Jilin Zhang [1,*]**

[1] College of Cyberspace, Hangzhou Dianzi University, Hangzhou 310018, China; zhaodh@hdu.edu.cn (D.Z.); zhannan@hdu.edu.cn (Z.L.)
[2] DBAPPSecurity Co., Ltd., Hangzhou 310051, China; jim.wang@dbappsecurity.com.cn
* Correspondence: kouliang@hdu.edu.cn (L.K.); jilin.zhang@hdu.edu.cn (J.Z.)

**Abstract:** Due to the rapid development and widespread presence of malware, deep-learning-based malware detection methods have become a pivotal approach used by researchers to protect private data. Behavior-based malware detection is effective, but changes in the running environment and malware evolution can alter API calls used for detection. Most existing methods ignore API call parameters while analyzing them separately, which loses important semantic information. Therefore, considering API call parameters and their combinations can improve behavior-based malware detection. To improve the effectiveness of behavior-based malware detection systems, this paper proposes a novel API feature engineering method. The proposed method employs parameter-augmented semantic chains to improve the system's resilience to unknown parameters and elevate the detection rate. The method entails semantically decomposing the API to derive a behavior semantic chain, which provides an initial representation of the behavior exhibited by samples. To further refine the accuracy of the behavior semantic chain in depicting the behavior, the proposed method integrates the parameters utilized by the API into the aforementioned semantic chain. Furthermore, an information compression technique is employed to minimize the loss of critical actions following truncation of API sequences. Finally, a deep learning model consisting of gated CNN, Bi-LSTM, and an attention mechanism is used to extract semantic features embedded within the API sequences and improve the overall detection accuracy. Additionally, we evaluate the proposed method on a competition dataset Datacon2019. Experiments indicate that the proposed method outperforms baselines employing vocabulary-based methods in both robustness to unknown parameters and detection rate.

**Keywords:** privacy protection; malware detection; deep learning; feature hashing; API sequences



**Citation:** Zhao, D.; Wang, H.; Kou, L.; Li, Z.; Zhang, J. Dynamic Malware Detection Using Parameter-Augmented Semantic Chain. *Electronics* **2023**, *12*, 4992. https://doi.org/10.3390/electronics12244992

Academic Editor: Fabio Grandi

Received: 6 November 2023
Revised: 11 December 2023
Accepted: 12 December 2023
Published: 13 December 2023



## 1. Introduction

With the development of information technology, security of cyberspace, which is mostly threatened by malware, has become more important than ever. According to the malware report released by AV-TEST, the amount of malware has been growing at a tremendous rate, which is 50 million on average per year in the past decade [1]. In the face of such an abundance of malicious software, the privacy data of any internet user is under significant threat. Therefore, it is crucial to have effective malware detection approaches to protect this private information and data.

In past decades, the mainstream of malware detection methods proposed by researchers can be divided into two: static analysis and dynamic analysis, respectively. Static analysis mostly focuses on signature-based analysis, which creates a specific signature that is unique enough to represent the whole file by scanning the binary byte streams, such as printable strings and n-gram [2,3]. Static analysis can finish analyzing tasks at a fast rate, but it is easy to be evaded by some malware with specific technologies, such as code obfuscation [4]. Also, it is hard for static analysis to detect new malware, which probably

uses a zero-day attack [5]. On the contrary, dynamic analysis is more robust to malware using evasion technology [6] by monitoring the behavior generated by a target sample in an isolated environment, which is something like IntelligentMonitor proposed by [7]. This captured behavior information often contains API calls, network activities, memory usage, etc. Moreover, all such information can serve as valuable reference material for judging a malware. For example, Herrera-Silva et al. [8] collected the behavior information reports of ransomware and selected features—including file, PID, network, and more—that are of interest for ransomware detection. Moreover, due to such behavior, dynamic analysis exerts a higher detection rate and precision.

The system API call sequence is the most representative among all the captured behavior information, including file manipulation operation, network access, and register key modification, etc. [9]. API sequences consist of API calls and their parameters, in which case text processing methods such as N-grams can be applied to support API sequences analysis, which is commonly used by researchers.

With the rapid rise of deep learning in recent years, researchers have also started to apply deep learning models such as CNN, RNN, and LSTM to the field of malware detection. These deep learning models can effectively mine the feature information in a sequence to train a malware detector with high accuracy. Although many researchers have proposed API-sequence-based malware detection methods [10,11], most of these methods only focus on the API itself and not on the parameters of the API. Methods such as those in [10,12] show a significant improvement in methods that combine APIs and their parameters compared to methods that only consider the APIs themselves. Indeed, [13,14] dealt with both the API and its parameters but did not consider the semantic information hidden in the API itself. Furthermore, [13] uses a rule-based clustering algorithm to deal with the parameters in the API, which also means that it requires a great deal of expertise and is more complicated. Although [15] constructs a semantic chain with the information hidden in the API sequence, it also does not consider the influence of the parameters on the semantics.

Furthermore, the most prevalent approach to handling text in the NLP field is by utilizing a dictionary—initially segmenting the text into tokens and building a dictionary, followed by substituting the tokens in the original text with their respective indices in the dictionary. Under this kind of method of feature engineering, deep learning struggles, in some cases, to handle text that is outside the dictionary. In the case of malware detection, the parameters are determined by the malware developers, meaning that any text could potentially appear in these parameters, especially parameters like url and file paths. A similar situation exists in the field of recommender algorithms, such as the ID, which is usually a combination of numbers and letters of streaming media on YouTube, where such features typically cannot be processed using a standard dictionary and cannot be ignored.

In this paper, a novel API sequence feature extraction method is proposed to solve the problems mentioned above. Inspired by [9,15], we use feature hash to encode the semantic chain of APIs with added parameters. Text can be directly transformed into vectors by hashing algorithms without a middle step by using feature hashing, which means feature hashing is able to handle all the text that can be hashed. Thus, the problem of unknown input from parameters is effectively solved, which allows the model to handle unknown samples effectively. By semantic analysis of the APIs, the actions and objects representing the API operations and operation objects are extracted from the APIs, and the statistics of the actions and objects appearing in the API sequences are counted as supplementary information of the sequences. Secondly, in order to preserve the semantic information to the maximum extent and to reduce the overhead of feature engineering, only the simplest formatting of the parameters is performed. Finally, the obtained semantic chains are encoded using feature hash to generate feature vectors. Due to the complexity of the parameters in the API sequences, the ordinary method of constructing a word list and replacing them would consume a great deal of time and memory, and the model cannot effectively handle unknown inputs. Using feature hash can be a good solution to this

challenge without constructing a word list, and it can also make the model robust to unknown inputs. Finally, we put the obtained feature vectors after encoding into a deep network for training, which consists of gated CNN, Bi-LSTM, and attention.

The main contributions of this paper include

1. In this paper, a new API feature extraction approach is proposed from the perspective of semantic information and the characteristics of API sequences. This approach maximizes the description of the behavior of the samples by referring to both the semantic information of the API itself and its parameters. We semantically decompose the APIs and use the parameters of the APIs as an augmentation of the semantic information to extract a semantic chain containing complete semantic information from the API sequences.
2. This paper refers to the common practice in recommender systems and applies feature hash to behavior-based dynamic malware detection to solve the dynamic input problem caused by API parameters. It enables the malware detection system to handle unknown inputs more efficiently, thus alleviating the problem of an aging detection system to a certain extent.
3. A new API sequence information compression method is proposed. Therefore, this paper borrows the method of dealing with extra-long sequences in NLP, i.e., using key sentences instead of whole paragraphs. The statistical information of the sequence is used as the "key sentence" of the whole sequence, which is used as the complement of the API sequence and solves the problem of losing the key behavior caused by truncating the API sequence. The final experimental results prove that this approach is very effective in dynamic detection.
4. We evaluate the recognition performance of this method on a competition dataset. Comparison experiments with other baseline models are also conducted. The final experimental results show that our method is significantly better than various baseline models.

## 2. Related Work

### *2.1. Static Analysis*

Static analysis was widely used in malware detection in the past. Thanks to its high detection efficiency, static analysis is still active in malware detection. Static analysis methods create signatures by scanning the raw input (such as binary byte streams of software) and judging the target sample by created signatures. Downing et al. [16] proposed DEEPREFLECT, which is a tool that is capable of localizing and identifying malicious components inside a malware. The authors used an unsupervised deep neural network and a semi-supervised cluster analysis to classify the components. Saxe et al. [17] extracted three different types of complementary features, which are context bytes features, PE import features, and string 2-dimension histogram features, respectively. Raff et al. [18] proposed a novel approach for malware detection that involves utilizing the entire raw binary file as a lengthy sequence for identification purposes. This method eliminates the need for conventional reverse engineering techniques, thereby significantly reducing the human effort required in the malware detection process.

Due to the fact that static methods do not require running samples, static methods can be used to analyze samples in a short period of time. However, the disadvantage of static detection methods is also obvious, namely that they cannot effectively identify unknown samples. Also, static detection is susceptible to interference from technical means, such as encryption and code obfuscation.

### *2.2. Dynamic Analysis*

#### 2.2.1. API-Sequence-Based Methods

Researchers focused on API sequences and the associated features early on. Lee et al. [19] introduced two new machine learning techniques that exhibit high sustainability using API calls and obtained the highest accuracy of 97.8% and an F1-score of 98.8%. Ahmadi et al. [20],

on the other hand, focused on frequently repeated parts of the API sequences, and they argued that malware developers often use a number of cyclic operations to perform actions such as decrypting encryption or infection. Ravi et al. [21] used a 4-gram to model the API sequences and classified the samples by comparing the average confidence of all the 4-grams. Cheng et al. [22] also considered APIs and their parameters, and used information retrieval theory to detect the samples. Fang et al. [23] also used feature hash to transform API name, return value, and module name into a fixed-length feature vector and used the machine learning algorithm XGBoost to mine the feature vector. Tian et al. [24] used a hash table to represent the string, i.e., API name and parameters, and used a random forest algorithm to classify the samples. Zhang et al. [25] created a "behavioural chain" based on API names and the order of calls between APIs to aid detection. All the above methods are based on some human-defined rules for analysis and have limited feature mining.

#### 2.2.2. Deep-Learning- and Sequence-Encoding-Based Methods

With the rise of deep learning, researchers have proposed many approaches based on sequence coding using the way sequences are handled in NLP as a reference object. Tran et al. [26] used paragraph vectors to semantically segment API sequences and assigned weights to each API using TF-IDF. TF-IDF is typically used to transform sequences of n-grams divided into word elements into numeric vectors that can be processed by deep learning models. Amer et al. [10] used word embedding to represent each API name as a feature vector containing contextual information. However, both the above approaches only consider the API names without considering the parameters and the semantic information of the APIs themselves.

Agrawal et al. [12] used an n-gram to obtain the N lexical elements with the highest number of occurrences in the parameters and used one-hot to encode the API and the obtained N lexical elements to obtain the feature vector. As for the work of Zhang et al. [9], a comprehensive analysis of the parameters of the API was performed and different encoding methods were used for different parameter types. In addition, the number of occurrences of each type of parameter in the parameters was counted. Chen et al. [13] performed a clustering analysis of the various parameters of the API by formulating rules, from which the API parameters were mined for possible malicious parameter combinations, and replaced the cumbersome parameters in the sequence using labels indicating the degree of maliciousness, and finally encoded the API and parameter labels separately using word embedding. Although [9,13] both analyze API parameters in detail, they do not consider the semantic information of the API itself, and also ignore the global information of the API sequence.

## 3. Methodology

### *3.1. Overview of the Design*

The overview of dynamic malware detection using parameter-augmented semantic chain is shown in Figure 1. At the beginning of our detection process, we need to collect the behavior reports, which are conducted by virtual environment like sandbox, of all the target samples. Then, we analyze the reports and extract parameter-augmented semantic chain from these reports. Eventually, we would convert obtained semantic chain into feature vectors and analyze by a deep learning network.

### *3.2. Feature Vector Generation*

Most API-sequence-based detection methods miss the information implied by the parameters of the API, which, in all likelihood, is the key information to determine whether the sample is malicious or not. For example, for NtWriteFile, the parameters indicate what file the sample will write. If the file to be written is a system file, then the sample is likely to be malicious; however, if the target to perform the write operation is just a file of little use, then the sample is relatively less likely to be malicious. It can be seen that, although it

is the same API, its semantic information in the sequence can vary very much depending on the parameters.

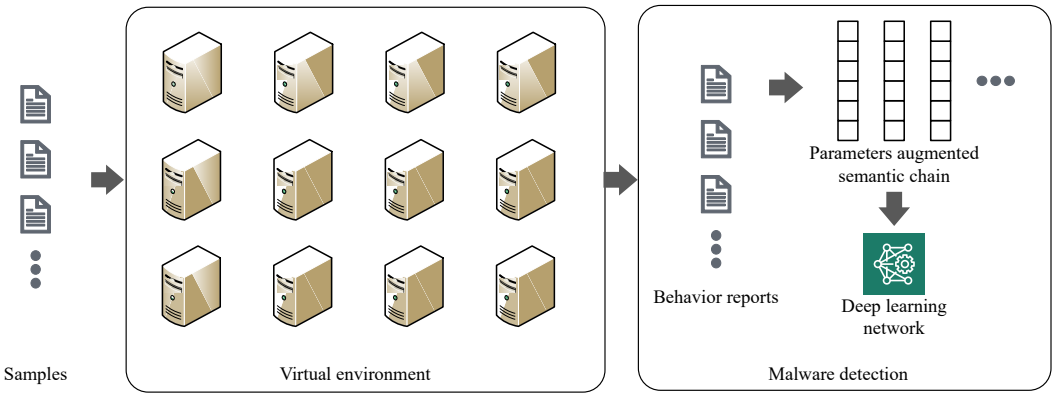


**Figure 1.** System overview.

Likewise, the API itself contains some information. This is because the developers of the operating system define the API in a principle to a specific naming convention, usually semantic-based naming [27]. For example, NtWriteFile contains both write as an action. Thus, the API name contains some semantic information that reflects the behavior of the API and the object it operates on. Based on this principle, we think it is meaningful to semantically segment an API name, to separate API into actions and objects, rather than just simply handle the whole API or split it by words, which is exactly what is conducted in [9]. Therefore, when we combine the API and parameter information together, we can obtain the complete semantic information. Inspired by the work in [9], we use the hash method proposed in [28] to encode the action, object, and parameter of the API separately. The whole process of generating feature vector is shown in Figure 2.

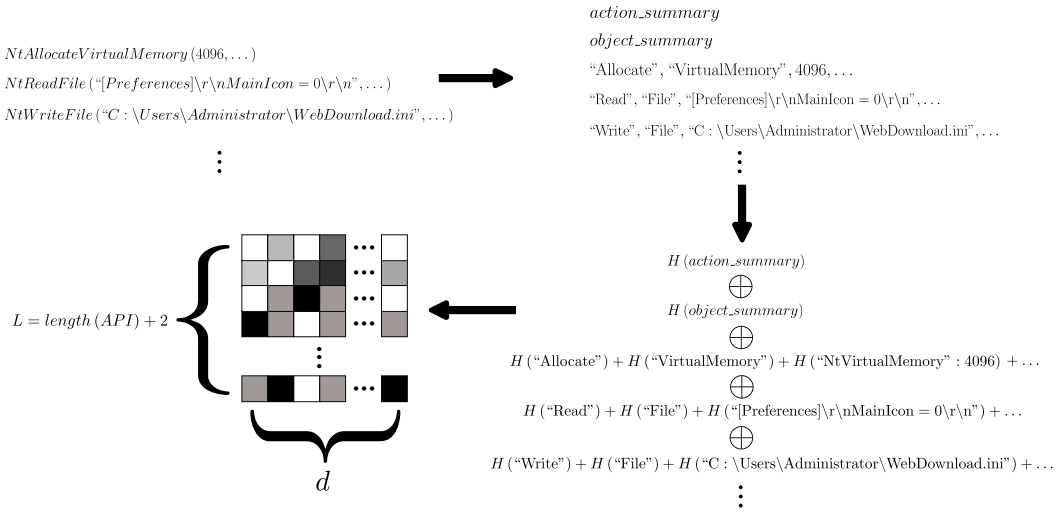


**Figure 2.** API sequence to semantic chain.

### *3.3. Action and Object*

Both action and object are extracted directly from the API. The action dictionary proposed in [15] was consulted and some updates were completed based on it. Moreover, 78 different actions are available, most of which are verbs, such as add, create, etc. The remaining parts of the API can be regarded as objects. "W" only represents a different coding style, and its impact on semantics can be ignored. Table 1 contains almost all the actions in the API.

**Table 1.** Actions.

| Actions | # |
|---|---|
| Bind, Accept, Acquire, Add, Adjust, Allocate, analyze, Assign, check, Close, Compress, Connect, Control, Copy, Crack, Create, Decode, Decompress, Decrypt, Delete, Detect, Download, Draw, Duplicate, Encode, Encryp, Enum, Exec, Exit, Export, Find, First, Free, Gen, Get, Hash, inject, Initialize, Is, Ki, listen, Load, Lookup, Make, Map, Move, Next, Obtain, Open, power, Protect, Put, Query, Queue, Read, Recv, Register, Remove, Resume, Save, Search, Select, Send, Set, shutdown, Sizeof, Socket, Start, Status, suspend, Terminate, Unhook, Uninitialize, Unload, Unmap, unpack, Unprotect, Write | 78 |

After the action and object are extracted, they are transformed into a feature vector of a specific length using the formula in [28]. The process of conventing string to a vector is shown in the middle part in Figure 3. Suppose a random variable $x$ represents a sequence of elements, each of which is a string or a character. m represents the number of dimensions, and the value of the $i$-th dimension is computed by Equation (1).

$$\sigma_i(x) = \sum_{p:f(x_p)=i} \varphi(x_p) \tag{1}$$

where $f$ is the hash function, mapping $x_p$ to a number $m \in \{1, \ldots, M\}$ as a dimensional index. $\varphi$ is another hash function, mapping elements to $\{-1, +1\}$.

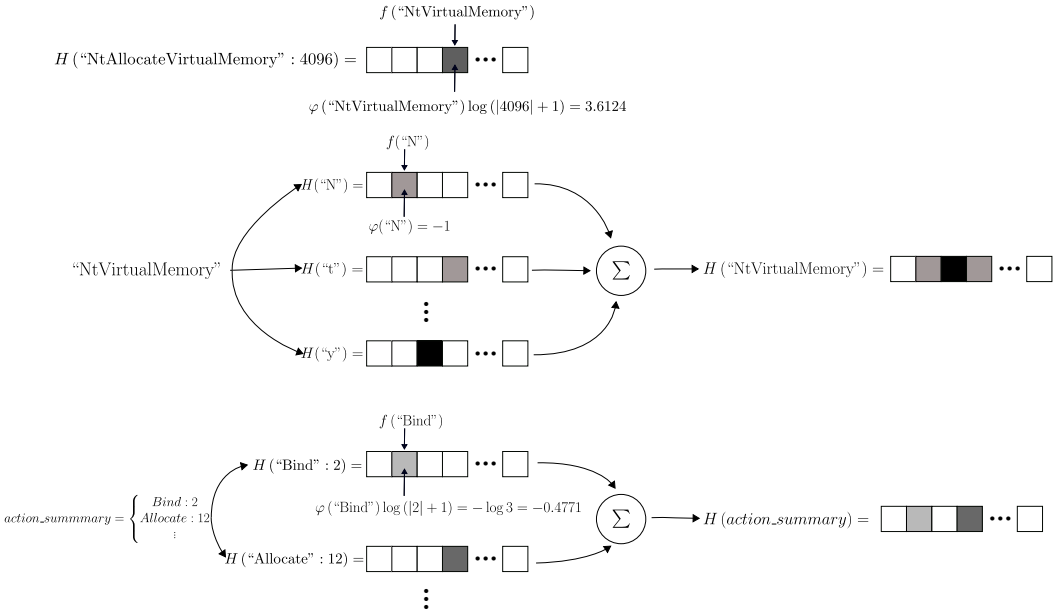


**Figure 3.** Feature hashing.

### *3.4. Parameters*

The parameters in the API sequence also contain a great deal of semantic information. In terms of type, there are only two types of parameters in API sequences, string and integer.

The process of encoding integer type parameters is shown in the uppermost part in Figure 3. For integer type parameters, the semantic information is not available for a single number but needs to be combined with the API name. For example, under the API name Connect, the parameter 22 represents the port number, while under other API names it may represent the file size. Therefore, we need to include the API name information when encoding the integer type parameters, as shown in Equation (2).

$$\sigma_i(x) = \sum_{p:f\left(x_p^{API}\right)=i} \varphi\left(x_p^{API}\right) \log \left|x_p^{value}\right| + 1 \tag{2}$$

We locate the position of the parameter in the feature vector based on the API name, i.e., $f\left(x_p^{API}\right) = i$ , and place the integer value into that position. However, if we directly put the original value into the feature vector, it may cause the value to fluctuate too much and the distribution to be sparse. Therefore, we only take its logarithm and put it into the feature vector. The rest of the symbols in the formula are the same as those in Equation (1).

For string type parameters, we need to conduct some cleaning of the parameters before converting them into feature vectors, considering the complexity of the parameters. First, we remove the hash values from the parameters. Most of these hashes represent a certain area in memory, i.e., virtual addresses, and these virtual addresses change with different devices, so virtual addresses are not such important information for dynamic detection methods based on API sequences [29,30], and hashes are more difficult to handle compared to other parameters. Therefore, the hash values in the parameters are not processed. Also, we filtered out the parameters that belong to API like LoadSystemModulue because we have observed a significant occurrence of such parameters in every single sample, no matter benign or malware. According to our statistics number, the average proportion of such API is expected to reach 40% of the total API length in one sample. More importantly, the values of parameters that belong to API like LoadSystemModule exhibit a remarkably high degree of redundancy. Due to the aforementioned reasons, we are led to believe that these values lack significant reference value, and we choose to filter them out. Finally, we format the chosen parameters, which includes standardizing the capitalization and the delimiters in the paths. After completing the parameter processing, the parameters of string type are also encoded by Equation (1).

### *3.5. Action and Object Summary*

When processing sequences using deep neural networks, it is common to use truncation or complementary methods to make the sequences the same length. If the complementary approach is used, using padding characters to make all the sequence lengths equal to the longest sequence, it will undoubtedly consume very much storage space. Therefore, the truncated sequence approach is more common in NLP problems. However, truncating sequences can result in missing sequence information. For malware, only a few specific behaviors are needed to achieve the final goal, and these behaviors may appear anywhere in the sequence.

In order to reduce the information loss caused by truncating the sequence, we add the global statistics of the sequence represented by each feature vector to the head of the vector. That is, the count statistics of the actions and objects in the sequence. In order to make the added information spliceable with the original sequence, the dimensions of action and object are each half of the total dimension of the sequence, as shown in the Figure 4.

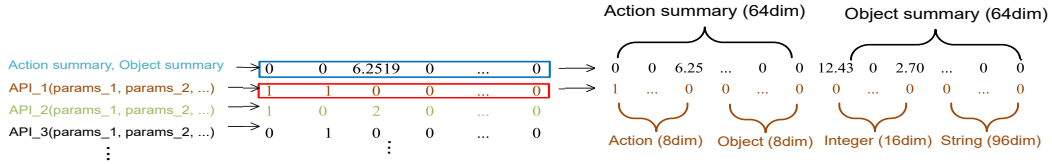


**Figure 4.** Feature engineering.

The feature vectors of statistical information are similarly calculated via Equation (2). The generation of summary vector is shown in the bottom part in Figure 3. The specific position of the number in the feature vector space is located by action and object, i.e., $f\left(x_p^{action}\right) = i$ and $f\left(x_p^{object}\right) = i$.

### *3.6. Deep Learning Network Architecture*

In order to reduce the probability of losing key behaviors due to truncated sequences, it is necessary to retain as much behavioral information in the sequences as possible, i.e., to preserve the length of the sequences as much as possible. This paper chooses to use the

combination of TextCNN and Bi-LSTM, which can process longer text sequences with less computational resources than the pre-trained model. Also, this paper makes some improvements to this combination by adding GatedCNN, which has better attention and text processing; deep learning network architecture can be seen in Figure 5.

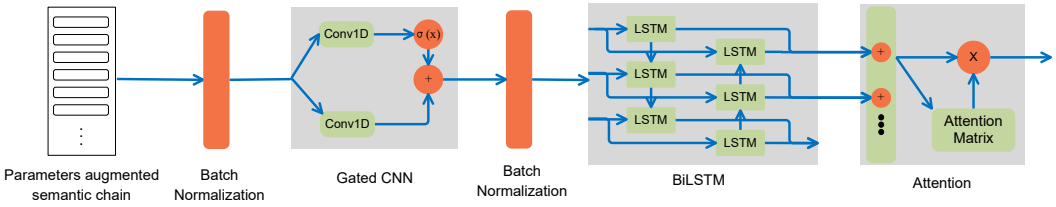


**Figure 5.** Network architecture.

#### 3.6.1. Gated CNN

The input feature vectors will be fed into gated CNN after batch normalization. Gated CNN [31] allows the selection of important and relevant information, making it more competitive with recurrent models on language tasks with fewer computation resources. To be more detailed, the input will be fed into two different convolution layers: let$X_A$ be the output of the first convolution layer, $X_B$ be the output of the second convolution layer, and then the final output of gated CNN will be $X_A \otimes \sigma(X_B)$, where $\otimes$ is an element-wise multiplication operation, $\sigma$ is the sigmoid function, $\sigma(x) = \frac{1}{1+e^{-x}}$. $\sigma(X_B)$ represents the gate unit that controls the information from $X_A$ to the next layers.

#### 3.6.2. Bi-LSTM

After convolutional filters, we apply bidirectional LSTM to capture the hidden patterns in the sequences. Although LSTM can predict future information based on past information, this also means that, in a text sequence, LSTM can only predict the following in conjunction with the information above. This is a more serious drawback when dealing with text data because there is basically no strict temporal order in text, and the semantic environment of the context affects the semantics of the word at the current position. By stacking two LSTMs in different directions, this makes the model save both past information and future information. The result of the i-th word after BiLSTM $h_i$ can be calculated by Equation (3).

$$h_t = [\overrightarrow{h_i} \oplus \overleftarrow{h_i}] \tag{3}$$

where $\overrightarrow{h_i}$ and $\overleftarrow{h_i}$ are the results of the forward and reverse calculations of Bi-LSTM, respectively.

#### 3.6.3. Attention

After learning sequential patterns from Bi-LSTM module, we apply a word-level attention mechanism [32] to learn the part that contributes to the result most. Let $H \in R^{d \times n}$ be the output of the bi-LSTM, which is a matrix of hidden status $h_1, h_2, \ldots, h_n$. $d$ denotes the size of the hidden layers in bi-LSTM, which is, in our case, 200. Furthermore, $n$ denotes the number of words in a sequence. Let $h_{ij}$ be a hidden status in $H$, and the hidden representation $u_{ij}$ of $h_{ij}$ is calculated by Equation (4).

$$u_{ij} = tanh(W_w h_i j + b_w) \tag{4}$$

where $W_w$ and $b_w$ are the weights and bias, respectively. Then, we can measure the importance of the word, a normalized importance weight vector $\alpha$, which would be the similarity of $u_{ij}$ and a context vector $u_w$ that is randomly initialized and learned during training. The importance weight vector of $\alpha_{ij}$ can be calculated by Equation (5).

$$\alpha_{ij} = \frac{e^{u_{ij}^T}}{\sum_j e^{u_{ij}^T u_w}} \tag{5}$$

And the final weighted output $r$ would be calculated by Equation (6).

$$r = H\alpha^T \tag{6}$$

## 4. Experiment

### *4.1. Experimental Environment*

An experimental environment can be generally divided into hardware environment and software environment.

Hardware Environment: The hardware environment of the experimental machine is as follows: CPU is i7-10700 with 8 cores and 32 GB memory; GPU is GeForce RTX 3060. The specific hardware parameters are shown in Table 2.

**Table 2.** Hardware parameter table.

| Hardware | Configuration |
|---|---|
| CPU | i7-10700 |
| Cores | 8 |
| Threads | 16 |
| GPU | GeForce RTX 3060 |
| Memory | 12 GB |
| RAM | 32 GB |
| Disk | 2 TB |

Software Environment: The experimental software environment is mainly configured as follows: the operating system is Windows 10, and the programming language used is Python 3.9. The third-party libraries required for the construction of the text detection model and the text recognition model are shown in Table 3, and the functions of each library are briefly introduced in the table. For specific usage methods, refer to the user manuals of each dependent package.

**Table 3.** Software environment.

| Name | Version | Function |
|---|---|---|
| torch | 1.8.0 | A deep learning framework open-sourced by Facebook, which supports GPU-based tensor computation and automatic gradient calculation. |
| numpy | 1.19.4 | The fundamental package for scientific computing with Python, providing a large number of matrix calculation functions. |
| math | 3.10.10 | Performs various advanced mathematical operations. |
| pandas | 0.25.1 | Used for simplifying large-scale structured data operation and analysis, supporting various matrix operations, data cleaning, and other functions. |
| matplotlib | 3.1.1 | A commonly used plotting library in Python for data visualization and creation of various charts. |
| scikit-learn | 0.21.3 | A third-party module that encapsulates commonly used machine learning methods, used for learning classification, regression, dimensionality reduction, and clustering the four major machine learning algorithms. |

### *4.2. Dataset*

We use the dataset from competition Datacon2019, which contains 69,207 behavior reports of different samples generated by sandbox for training our deep learning network. In those behavior reports, 20,000 of them are from benign samples; the remaining 49,207 reports are produced by malware. We divide the dataset into training set and validation set, 70% for training and 30% for validation. Additionally, we collated 2500 newest samples, which consist of both benign and malware in the real network to evaluate.

### *4.3. Hyperparameters*

In order to retain the behavioral information in the original sequence as much as possible, the maximum length of the API sequence is set to 3000. In addition, the dimensionality of each feature vector is set to 128, i.e., 7 times of 2 to speed up the computation in this paper. The convolutional kernel size in gated CNN is set to 3, and the number of LSTM units in each direction in Bi-LSTM is 100, totaling 200 LSTM units. The detailed parameters are shown in the table.

### *4.4. Metrics*

In this paper, we evaluate this method by calculating the accuracy, precision, recall, and F1-score of the final classification results and compare them with other methods. In addition, the ROC curve and the AUC score are used to compare the methods.

In the process of metrics calculating, we used TP and TN. TP denotes true positive, i.e., the number of samples correctly classified as POSITIVE (correctly classified as malware), and TN denotes true negative, i.e., the number of samples correctly classified as NEGATIVE (correctly classified as benign). FN denotes false negative; i.e., the number of samples incorrectly classified as FN denotes the number of samples incorrectly classified as negative (malicious samples are judged as benign samples), and FP denotes the number of samples incorrectly classified as positive (benign samples are judged as malicious samples).

### *4.5. Baselines*

In this paper, machine learning algorithms such as SVM and MLP are used as baseline models. The feature vectors obtained after semantic chain transformation of API sequences are put into the baseline model for feature mining and examining the indicators such as accuracy to evaluate the model performance in this paper. In addition, a widely used sequence processing method, embedding + TextCNN, is selected as the baseline for comparison in this paper. The results of the experiments can be observed in Table 4; the ROC curve is shown in Figure 6. Obviously, the method proposed in this paper outperforms all the baseline models.

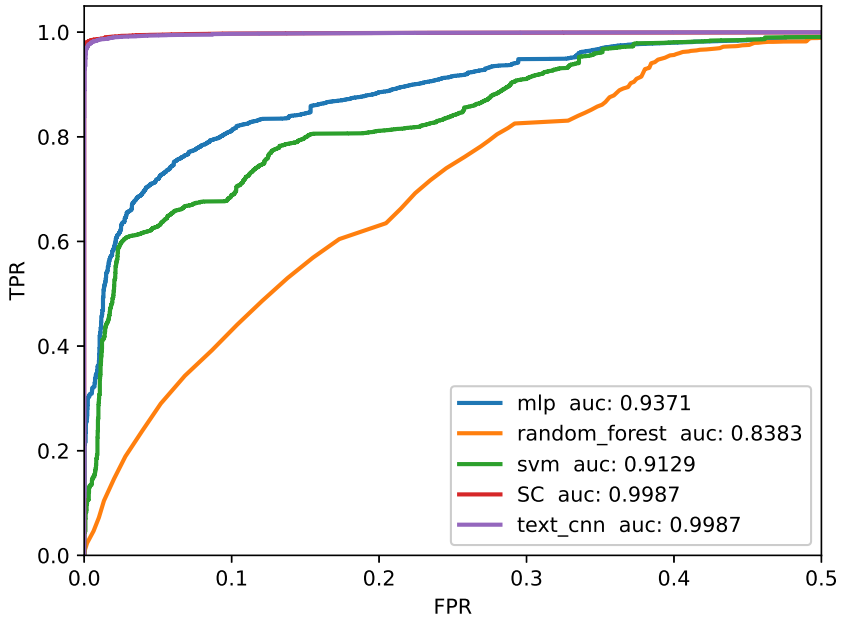


**Figure 6.** Comparison with baselines.

**Table 4.** Comparison with baselines.

| Methods | Accuracy | Presicion | Recall | F1-Score |
|---|---|---|---|---|
| SVM | 86.9190 | 86.7813 | 96.0866 | 91.1972 |
| RandomForest | 84.8239 | 85.3460 | 94.7479 | 89.8015 |
| MLP | 87.0827 | 89.0900 | 93.0815 | 91.0420 |
| Embedding + TextCNN | 98.5599 | 99.6882 | 98.2652 | 98.9716 |
| Proposed method | 99.1925 | 99.5428 | 99.3054 | 99.4264 |

## 5. Discussion

### *5.1. Robustness to Unknown Input*

In order to demonstrate that the proposed method can enhance the robustness of the model to unknown inputs, 2500 up-to-date samples existing in a real network environment are collected and the behavior reports of the samples are obtained by running the samples in a Cuckoo sandbox [33]. Three different methods are used to detect these 2500 samples and determine whether they are malware or not; the results are shown in Table 5.

**Table 5.** Robustness to unknown input.

| Methods | Accuracy | Presicion | Recall | F1-Score |
|---|---|---|---|---|
| Embedding + TextCNN | 75.21 | 78.68 | 76.26 | 84.97 |
| Proposed method (without API parameters) | 74.03 | 79.34 | 81.74 | 83.63 |
| Proposed method | 86.32 | 84.54 | 85.30 | 91.59 |

It is obvious from Table 4 and Figure 6 that the difference is small between the final detection rate of the proposed method and the method using embedding+TextCNN in the same dataset. The difference is small, and the proposed method in this paper has less advantage in detection rate. However, in Table 5, it can be seen that the proposed method has stronger robustness facing an unknown input, and it can obtain more information about the sample behavior from the unknown input, thus improving the detection rate of the model for unknown samples. Meanwhile, the experimental results also show that the detection rate of the model has a very significant improvement after adding the parameters. It is proved that the information carried by the parameters can greatly improve the recognition rate of unknown samples.

### *5.2. Influence of Summary*

To verify whether the supplementary information used in the method has a positive effect on the truncated sequences, an additional set of experiments is set up in this paper. In this set of experiments, different lengths are truncated from the API sequences. Considering that the maximum length that the pre-trained model BERT can handle is 512, three different lengths of 100, 300, and 500 are set in this paper. In addition, an experimental group with sequence length of 3000, which cannot be handled by the general model, is also set in this paper. For each sequence length, two experiments are conducted in this paper, i.e., with and without the addition of supplementary information. The final F1-score is shown in Table 6, and the ROC curve is shown in Figure 7. As is shown in Table 6, the F1-score obtained by sequences with summary is significantly better than those without summary, which means summary information is indeed useful for complementing the truncated chain information. Furthermore, with an increase in the length of sequences, the results that belong to sequences without summary maintain a trend of decreasing, which means that the detection performance would reduce with longer sequences without complementary information. On the other hand, the results conducted by sequences with summary showed a curved trend. Initially, a decrease occurs as the sequence length increases, but an increase begins after reaching a certain length. The reason, according to our analysis, can be that it is more easy for a deep learning network to fit when the sequence length is short, and,

when the length is not so short, the model's degree of fit will decrease for a while. After that, as the sequence length is further increased, the summary information will play a more prominent role.

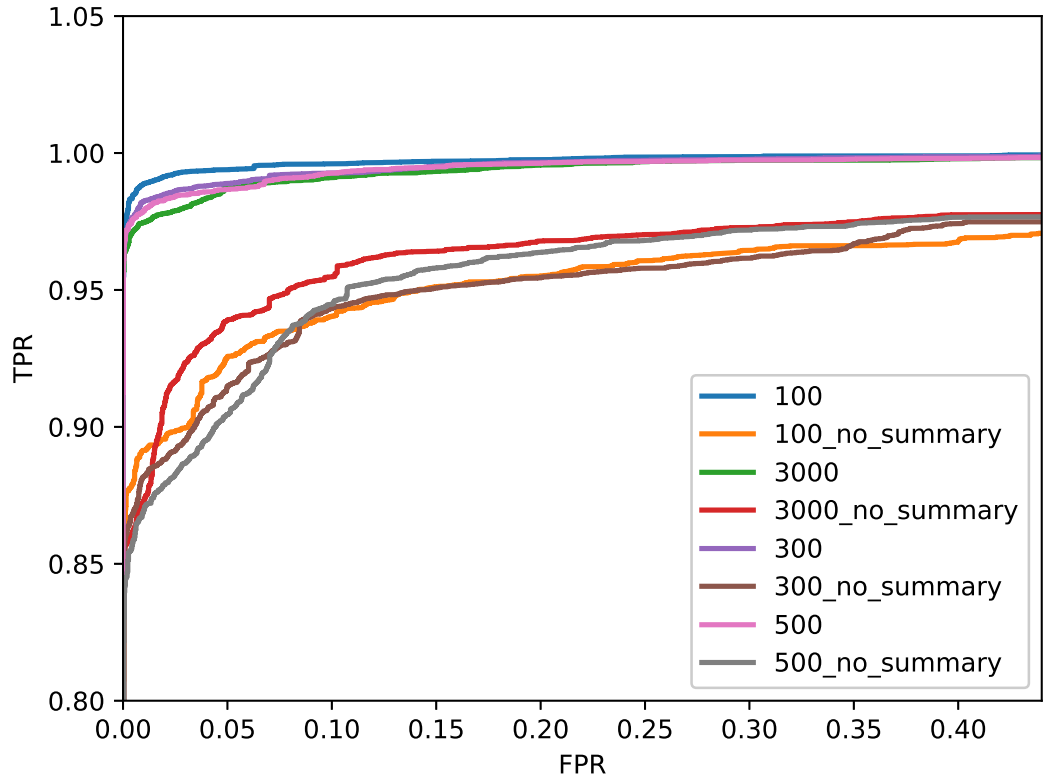


**Figure 7.** Influence of summary.

**Table 6.** Influence of summary.

| Length of API | With Summary | Without Summary |
|---|---|---|
| 100 | 99.20 | 92.79 |
| 300 | 98.90 | 92.35 |
| 500 | 98.19 | 91.53 |
| 1000 | 98.21 | 91.12 |
| 3000 | 98.26 | 90.26 |

### *5.3. Limitations and Future Work*

Although the method proposed in this paper is obviously better than the baseline model, the method currently has many shortcomings. First, the APIs studied in this paper are basically from the Windows environment, so the method is currently only applicable to malware detection in the Windows environment. If there is a sample that will only perform malicious behavior under Linux, the method cannot perform accurate analysis of the sample. In addition, the text does not fundamentally address the problem of concept drift, although it uses feature hash to enhance the robustness of the model to unknown inputs. If the behavior pattern of a sample changes, the model still needs to be retrained to adapt it to the new behavior pattern.

In future work, we plan to investigate the second deficiency mentioned above in more depth, i.e., to more deeply investigate the concept drift that occurs in deep learning models on new data. The problem of gradual aging of the model on new datasets is mitigated by taking certain optimization measures on the deep learning model.

## 6. Conclusions

In this paper, we propose a novel manner for API sequences encoding. We obtain semantic chains by deconstructing the API and employing the parameters of the API to augment the semantic information. Moreover, the use of feature hash to solve the dynamic input problem arising from the processing of parameters enhances the robustness of the model in the face of unknown inputs. In addition, we mitigate the effect of sequence truncation by incorporating sequence statistics. Finally, we use deep neural networks to mine the features in the API sequences and complete the detection. In addition, we also design experiments to demonstrate that adding the overall information of the sequence as a complement to the truncated sequence can effectively solve the problem of missing key information.

**Author Contributions:** Conceptualization, L.K.; Methodology, D.Z.; Validation, H.W.; Data curation, Z.L.; Writing—original draft, J.Z. All authors have read and agreed to the published version of the manuscript.

**Funding:** This paper has been supported by the Key Technology Research and Development Program of Zhejiang Province under Grant 2022C01125 and the General Research Program of the Department of Education under Grant Y202044517.

**Data Availability Statement:** Data are contained within the article.

**Conflicts of Interest:** Huadong Wang was employed by the company DBAPPSecurity Co., Ltd. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

## References

1. AV-TEST. AV-TEST Report. 2022. Available online: https://www.av-test.org/en/statistics/malware/ (accessed on 1 December 2022).
2. Santos, I.; Penya, Y.K.; Devesa, J.; Bringas, P.G. N-grams-based file signatures for malware detection. In *International Conference on Enterprise Information Systems*; SCITEPRESS: Setúbal, Portugal, 2009; Volume 9, pp. 317–320.
3. Griffin, K.; Schneider, S.; Hu, X.; Chiueh, T.C. Automatic generation of string signatures for malware detection. In Proceedings of the International Workshop on Recent Advances in Intrusion Detection, Saint-Malo, France, 23–25 September 2009; Springer: Berlin/Heidelberg, Germany, 2009; pp. 101–120.
4. You, I.; Yim, K. Malware obfuscation techniques: A brief survey. In Proceedings of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications, Fukuoka, Japan, 4–6 November 2010; pp. 297–300.
5. Bilge, L.; Dumitraş, T. Before we knew it: An empirical study of zero-day attacks in the real world. In Proceedings of the 2012 ACM conference on Computer and Communications Security, Raleigh, NC, USA, 16–18 October 2012; pp. 833–844.
6. Damodaran, A.; Troia, F.D.; Visaggio, C.A.; Austin, T.H.; Stamp, M. A comparison of static, dynamic, and hybrid analysis for malware detection. *J. Comput. Virol. Hacking Tech.* **2017**, *13*, 1–12. [CrossRef]
7. Thantharate, P. IntelligentMonitor: Empowering DevOps Environments with Advanced Monitoring and Observability. In Proceedings of the 2023 International Conference on Information Technology (ICIT), Amman, Jordan, 9–10 August 2023; pp. 800–805.
8. Herrera-Silva, J.A.; Hernández-Álvarez, M. Dynamic feature dataset for ransomware detection using machine learning algorithms. *Sensors* **2023**, *23*, 1053. [CrossRef] [PubMed]
9. Zhang, Z.; Qi, P.; Wang, W. Dynamic malware analysis with feature engineering and feature learning. *AAAI Conf. Artif. Intell.* **2020**, *34*, 1210–1217. [CrossRef]
10. 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]
11. 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]
12. 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.
13. 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]
14. Salehi, Z.; Sami, A.; Ghiasi, M. MAAR: Robust features to detect malicious activity based on API calls, their arguments and return values. *Eng. Appl. Artif. Intell.* **2017**, *59*, 93–102. [CrossRef]

15. 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.* **2022**, *116*, 102686. [CrossRef]
16. Downing, E.; Mirsky, Y.; Park, K.; Lee, W. DeepReflect: Discovering Malicious Functionality through Binary Reconstruction. In Proceedings of the USENIX Security Symposium, Online, 11–13 August 2021; pp. 3469–3486.
17. Saxe, J.; Berlin, K. Deep neural network based malware detection using two dimensional binary program features. In Proceedings of the 2015 10th International Conference on Malicious and Unwanted Software (MALWARE), Fajardo, PR, USA, 20–22 October 2015; pp. 11–20.
18. Raff, E.; Barker, J.; Sylvester, J.; Brandon, R.; Catanzaro, B.; Nicholas, C. Malware detection by eating a whole exe. *arXiv* **2017**, arXiv:1710.09435.
19. Lee, H.; Cho, S.j.; Han, H.; Cho, W.; Suh, K. Enhancing Sustainability in Machine Learning-based Android Malware Detection using API calls. In Proceedings of the 2022 IEEE Fifth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), Laguna Hills, CA, USA, 19–21 September 2022; pp. 131–134.
20. Ahmadi, M.; Sami, A.; Rahimi, H.; Yadegari, B. Malware detection by behavioural sequential patterns. *Comput. Fraud. Secur.* **2013**, *2013*, 11–19. [CrossRef]
21. Ravi, C.; Manoharan, R. Malware detection using windows api sequence and machine learning. *Int. J. Comput. Appl.* **2012**, *43*, 12–16. [CrossRef]
22. Cheng, J.Y.C.; Tsai, T.S.; Yang, C.S. An information retrieval approach for malware classification based on Windows API calls. In Proceedings of the 2013 International Conference on Machine Learning and Cybernetics, Tianjin, China, 14–17 July 2013; Volume 4, pp. 1678–1683.
23. Fang, Y.; Yu, B.; Tang, Y.; Liu, L.; Lu, Z.; Wang, Y.; Yang, Q. A new malware classification approach based on malware dynamic analysis. In Proceedings of the Australasian Conference on Information Security and Privacy, Auckland, New Zealand, 3–5 July 2017; Springer: Berlin/Heidelberg, Germany, 2017; pp. 173–189.
24. Tian, R.; Islam, R.; Batten, L.; Versteeg, S. Differentiating malware from cleanware using behavioural analysis. In Proceedings of the 2010 5th International Conference on Malicious and Unwanted Software, Nancy, France, 19–20 October 2010; pp. 23–30.
25. Zhang, H.; Zhang, W.; Lv, Z.; Sangaiah, A.K.; Huang, T.; Chilamkurti, N. MALDC: A depth detection method for malware based on behavior chains. *World Wide Web* **2020**, *23*, 991–1010. [CrossRef]
26. Tran, T.K.; Sato, H. NLP-based approaches for malware classification from API sequences. In Proceedings of the 2017 21st Asia Pacific Symposium on Intelligent and Evolutionary Systems (IES), Hanoi, Vietnam, 15–17 November 2017; pp. 101–105.
27. Hart, J.M. *Win32 Systems Programming*; Addison-Wesley Longman Publishing Co., Inc.: Reading, MA, USA, 1997.
28. Weinberger, K.; Dasgupta, A.; Langford, J.; Smola, A.; Attenberg, J. Feature hashing for large scale multitask learning. In Proceedings of the 26th Annual International Conference on Machine Learning, Montreal, QC, Canada, 14–18 June 2009; pp. 1113–1120.
29. Jindal, C.; Salls, C.; Aghakhani, H.; Long, K.; Kruegel, C.; Vigna, G. Neurlux: Dynamic malware analysis without feature engineering. In Proceedings of the 35th Annual Computer Security Applications Conference, San Juan, PR, USA, 9–13 December 2019; pp. 444–455.
30. Avllazagaj, E.; Zhu, Z.; Bilge, L.; Balzarotti, D.; Dumitraș, T. When Malware Changed Its Mind: An Empirical Study of Variable Program Behaviors in the Real World. In Proceedings of the 30th USENIX Security Symposium (USENIX Security 21), Virtual, 11–13 August 2021; pp. 3487–3504.
31. Dauphin, Y.N.; Fan, A.; Auli, M.; Grangier, D. Language modeling with gated convolutional networks. In Proceedings of the International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 933–941.
32. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems 30, Proceedings of the Annual Conference on Neural Information Processing Systems 2017, Long Beach, CA, USA, 4–9 December 2017*; Curran Associates, Inc.: Red Hook, NY, USA, 2017.
33. Foundation, C. Cuckoo Sandbox. 2019. Available online: https://cuckoosandbox.org/ (accessed on 1 January 2019).