telecom

MDPI

Article

Lightweight and Robust Malware Detection Using Dictionaries of API Calls

Ammar Yahya Daeef [1], Ali Al-Naji [2,3,*] and Javaan Chahl [3]

[1] Technical Institute for Administration, Middle Technical University, Baghdad 10074, Iraq; ammaryahyadaeef@mtu.edu.iq
[2] Electrical Engineering Technical College, Middle Technical University, Baghdad 10022, Iraq
[3] School of Engineering, University of South Australia, Mawson Lakes, SA 5095, Australia; javaan.chahl@unisa.edu.au
* Correspondence: ali_al_naji@mtu.edu.iq

Abstract: Malware in today's business world has become a powerful tool used by cyber attackers. It has become more advanced, spreading quickly and causing significant harm. Modern malware is particularly dangerous because it can go undetected, making it difficult to investigate and stop in real time. For businesses, it is vital to ensure that the computer systems are free from malware. To effectively address this problem, the most responsive solution is to operate in real time at the system's edge. Although machine learning and deep learning have given promising performance for malware detection, the significant challenge is the required processing power and resources for implementation at the system's edge. Therefore, it is important to prioritize a lightweight approach at the system's edge. Equally important, the robustness of the model against the concept drift at the system's edge is crucial to detecting the evolved zero-day malware attacks. Application programming interface (API) calls emerge as the most promising candidate to provide such a solution. However, it is quite challenging to create API call features to achieve a lightweight implementation, high malware detection rate, robustness, and fast execution. This study seeks to investigate and analyze the reuse rate of API calls in both malware and goodware, shedding light on the limitations of API call dictionaries for each class using different datasets. By leveraging these dictionaries, a statistical classifier (STC) is introduced to detect malware samples. Furthermore, the study delves into the investigation of model drift in the STC model, employing entirely distinct datasets for training and testing purposes. The results show the outstanding performance of the STC model in accurately detecting malware, achieving a recall value of one, and exhibiting robustness against model drift. Furthermore, the proposed STC model shows comparable performance to deep learning algorithms, which makes it a strong competitor for performing real-time inference on edge devices.

Keywords: API call sequence; statistical classifier; model drift; malware detection

check for updates

Citation: Daeef, A.Y.; Al-Naji, A.; Chahl, J. Lightweight and Robust Malware Detection Using Dictionaries of API Calls. *Telecom* **2023**, *4*, 746–757. https://doi.org/10.3390/telecom4040034

Academic Editor: Christos Volos

Received: 16 July 2023
Revised: 9 September 2023
Accepted: 16 October 2023
Published: 2 November 2023

1. Introduction

The proliferation of communication networks and the Internet has created a paradigm shift in how organizations manage their operations. This great influence of technology, along with the rapid digitization of various spheres of life [1] and the emergence of cloud and edge computing [2,3], has ushered in a new era of communication with a huge number of interconnected devices [4]. Unfortunately, this interconnectedness has also given rise to frightening and unprecedented levels of cyber attacker activity. The ongoing spread of new and sophisticated cyber-attacks targeting critical infrastructure, government agencies, and the financial sector poses significant challenges at both the individual and societal levels. These sophisticated cyber attacks rely heavily on malicious software, commonly referred to as malware [5]. Malware serves diverse purposes, including identity theft, financial theft, disruptive activities, cyber espionage, and illegal extraction of sensitive data, driven by various personal or political motives [6].

Malware is classified into different categories based on harmful behavior such as Trojan, virus, worm, and ransomware [7–9]. The behavior of each type is characterized by the spreading method and the impact on the targeted machine; for instance, the worms have the ability to independently spread into other networks without the need for file attachment or user action. Contradictorily, the virus spreads by attaching itself to files on the host, and user interaction is required to function. Malware harms the targeted system in different ways to break the framework of IT security CIA (confidentiality, integrity, and availability) [10]. Some malware, such as keyloggers, leads to the leakage of sensitive information which compromises the confidentiality term, while worms can affect the availability of the system. Also, some malware can be created to modify or delete data on the system which compromises the integrity of data.

The fast development of cyber threats has witnessed the rise of more complicated and harmful forms of malware such as metamorphic and polymorphic malicious code [11]. Polymorphic malware changes the binary form every time it targets a new system. The change affects the code structure while keeping the same behavior. A common method used to achieve such transformation is packing using encryption and compression to transfer the binary. In the same context, metamorphic malware takes the same idea of polymorphic malware a step further by transforming the binary and behavior with each new infection. These techniques make the detection of traditional signature-based solutions a challenging process.

According to a report by Cybercrime Magazine [12], only one malware type (ransomware) had a devastating impact in 2021, causing a global financial loss of USD 20 billion. It is estimated that this number will grow exponentially, reaching US$265 billion by 2031. Alongside cutting-edge strategies employed by attackers, the proliferation of malware is also escalating rapidly. Kaspersky's detection models identified an average of 400,000 new attacks being distributed daily during 2022, which is an increase of 5 percent compared to 2021 [13]. This incessant generation of malware poses significant risks to enterprises, exposing them to a wide range of sophisticated attacks. Consequently, integrating smart security technologies into the network infrastructure becomes imperative for organizations in order to effectively address the diverse range of malware attacks and ensure the protection of data and systems.

Malware often exhibits signature characteristics that expose the malicious action; the leading antivirus applications heavily rely on such signatures for detection. However, this approach falls short in identifying newly emerged malware [14]. Attackers further undermine the reliability of such solutions by employing elusive methods such as encryption and packing [15]. Despite the ingenious ideas employed by attackers, malware execution and malicious activities still depend on the presence of a host's operating system [16]. Therefore, malware seeks to exploit the Windows API call service as a means to carry out malicious actions. By taking advantage of the resources and functions provided by the operating system, malware can manipulate and interact with critical system components, enabling unauthorized operations and potentially compromising the security of the target system. As a result of this interaction, harmful behaviors are exhibited that can be leveraged to detect the presence of malware. As stated in [17], the use of the system API call cannot be hidden by malware, allowing for effective identification and detection of malicious activity. Despite the effective benefits of employing machine learning in detecting malware, particularly zero-day malware, by using data extracted from the execution phase, such as API calls, the process of gathering precise API call features to reliably differentiate between malware and goodware continues to present a formidable challenge. The accuracy provided by API call features has been emphasized in most research in this area [18], rather than prioritizing the speed of solutions. Additionally, earlier studies have focused on complex feature sets or the utilization of neural networks, while simpler techniques remain relatively unexplored. Equally important, in a dynamic world, nothing remains static especially when talking about malware as the characteristics evolved over time. This effect changes the distribution of data used to train the detection models. Such a significant effect makes the detection models less effective in detecting zero-day malware.

Cybersecurity tools can be employed at different layers of the enterprise IT infrastructure. However, by implementing malware detection at the edge, it becomes possible to implement malware protection technologies directly at the point where data is generated. While this method offers the advantage of real-time operation, its effectiveness is restricted by resource availability, making execution time a critical factor to consider. To develop edge systems that accurately identify potentially harmful situations within the constraints of execution time and complexity, this study aims to investigate the following research questions:

- What is a straightforward representation of API call features that can yield a highly effective malware detection model with fast execution?
- How does model drift impact the accuracy of such a model?

The main contributions of this research can be summarized in the following way:

- An analysis of API call behavior to uncover the reuse rate trends in both malware and goodware.
- Utilization of distinct datasets for training and testing purposes.
- Introduction of a straightforward feature representation method using API call dictionaries.
- A recommendation that the statistical classifier (STC) serves as a simple yet robust classifier that can effectively combat model drift. Furthermore, it offers a solution with less complexity, rendering it highly suitable for edge environments.

The subsequent sections of this paper are structured as follows: Section 2 discusses the malware analysis methods; Section 3 presents the related works; the used datasets are elaborated in Section 4; API call dictionaries are described in Section 5; Section 6 explains the STC model; the experimental results are presented in Section 7; Section 8 provides the comparison and limitation of this work; and finally, conclusions and future works are outlined in Section 9.

2. Static and Dynamic Methods for Malware Analysis

Cybersecurity techniques have witnessed many advances and developments to protect the systems. Despite such evolution, malware constitutes the most challenging threads in cyberspace [19]. Analysis of malware uses many techniques such as network and software analysis in order to draw a comprehensive understanding of malware behavior and how the harmful component has evolved over time. Malware can be analyzed statically or dynamically. Static analysis is used to collect different static indicators such as the data extracted from the header of the executable file [20]. Although this method does not need a controlled environment to run the malware to extract precious insights into the potential threat. However, the sophisticated malware can evade detection using various obfuscating techniques.

Conversely, to overcome the limitations of static methods, malware is analyzed dynamically by executing the file in an isolated environment to observe the functionality, behavior, and harmful actions. This is achieved by recording images of memory during the run-time. Opcodes [21], API calls [22], network data, and registry [23] are some examples of dynamic data that can be extracted and monitored. Since such a technique monitors the interaction of malware with the operating system, many effective detection models have been developed using dynamic analysis.

3. Related Works

Dynamic detection methods are impervious to obfuscation techniques of static detection like encryption and packing. This resistance has garnered significant attention from the malware detection community, leading to a heightened interest in dynamic detection approaches. In dynamic detection, the use of API calls is considered a valuable technique to represent malware behavior. The extraction process of API call sequences requires executing the malware in an isolated environment in order to monitor and record the API calls made by the malware [24]. The API call reveals the actions of malware such as changing the system registry, downloading infected files, extracting sensitive information, and other possible malicious activities. These API call sequences are used with different machine learning techniques to detect malware. The study referenced in [25] used Cuckoo Sandbox to capture several API calls generated by different types of malware and according to [26], the order of API calls can be thought of as a collection of transaction data, where each API call with the associated arguments constitutes a collection of elements. Consequently, identifying frequent itemsets within the sequence of API calls can unveil behavioral patterns. In this context, API call sequences and their frequency were utilized by Hansen et al. [27] to identify and classify malware. The use of a random forest (RF) classifier yielded encouraging predictive performance. The study presented in [22] introduced a process of two stages. Extracting API call frequency features using the Markov model is the first step followed by utilizing these features to train different machine learning models. The study by Daeef et al. [28] used visualization methods and the Jaccard index to investigate which groups of API calls are present in each malware family, with the goal of revealing hidden patterns of malicious behavior. Then, the frequency of each API call is used as a feature with RF providing the best results for malware family classification. Moreover, Daeef et al. [29] focused on utilizing the API calls' frequency within the initial 100 sequences. The outcomes were highly encouraging, indicating that RF exhibited comparable performance to other models such as long short-term memory (LSTM) and deep graph convolutional neural networks (DGCNNs). The authors in [30] employed the provided dataset from [31] to train a diverse range of deep learning and machine learning methods. The results indicated that traditional machine learning classifiers demonstrated superior performance while demanding less training time. Additionally, other investigations [32,33] focused on converting API call sequences into term frequency-inverse document frequency (TF-IDF) feature representation and subsequently evaluated multiple machine learning classifiers. Despite the efficiency of the machine learning method on the tested set, these approaches represent API calls as a bag of words, limiting their applicability to scenarios requiring full-length analysis.

Deep learning has been shown to be very effective in processing sequence data over time, especially in the field of natural language processing. In their paper [34], the authors used the recurrent neural networks (RNN) model to classify different malware families. They used long sequences of API calls to classify different types of malware. Furthermore, researchers in [35] utilize RNNs along with features extracted from API requests to differentiate malware in binary detection scenarios. A deep neural network (DNN) is presented in [36] to evaluate different malware datasets. Despite the promising analysis made, however, the methodology is not sufficient in representing the API call dependencies. The concept of transforming API calls into a graph structure was introduced by Oliveira et al. [31]. To achieve this, a secure sandbox environment is used to execute the malware and legitimate goodware to extract the sequences of API calls. These sequences are subsequently utilized to generate a behavioral graph. For detection purposes, the DGCNN is employed as the model. In a similar vein, researchers [37] have introduced a technique to represent the behavior of malware by converting API calls into images, by adhering to predefined criteria for color mapping. Then, these images' features are then classified using a convolutional neural network (CNN). Capturing the contextual relationship among the API call sequences using word embedding is presented in [38]. API calls are clustered according to the contextual similarity to generate a malware behavioral graph and a Markov chain technique is used for the detection process. Zhang et al. [39] propose a feature extraction process with a deep neural network. Feature representation employs the trick of hashing to encode the API call's category, parameters, and name. Multiple gated CNNs are used to transform the features of API calls and the output is fed into LSTM to learn the correlation relationship existing among the API calls.

In summary, the accuracy provided by API call features has been emphasized in most research in this area, rather than prioritizing the speed of solutions. Additionally, earlier studies have focused on complex feature sets which include an abundance of unnecessary information resulting in inefficient malware detection. The diversity of API call types, high-dimension features, and long sequences of API calls constitute a big challenge to creating lightweight detection models to be implemented in a real-time environment. Also, the utilization of neural networks makes things worse in this context due to the resource requirements. It is not logical to dive into complex solutions while simpler techniques remain relatively unexplored. Equally important, in a dynamic world, nothing remains static especially when talking about malware as the characteristics evolved over time. This effect changes the distribution of data used to train the detection models. Such a significant effect makes the detection models less effective in detecting zero-day malware. The concept of drift is another important issue not tackled in previous studies.

4. Malware and Goodware Dataset

In order to test any classification model in terms of classification performance and the effect of model decay, different datasets should be used. On this basis, this research uses available public datasets for this purpose. The first dataset (dataset1) was taken from [40] as this dataset aims to provide a foundation for continued development and improvement within the scholarly community. This dataset contains 1079 goodware and 42,797 malicious samples. Each sample was executed using the Cuckoo Sandbox program to generate the report of dynamic analysis. After this step, the API calls were extracted and processed to generate the equivalent ordered values. Each record in the dataset consists of the first one hundred consecutive, non-repeated API requests besides the MD5 hash of that sample. Clearly, the sample rate of this dataset is imbalanced, with the malware being 97% of the entire samples in the dataset. Such an imbalance could impair performance during the testing stage. To address this issue and to obtain more goodware samples, another dataset (dataset2) is collected from Prateek Lalwani's [41]. The dataset comprises 138,047 samples divided into two categories with 41,323 for goodware and 96,724 malware files collected using virusshare.com. This dataset provides the MD5 hash of each record. Model drift describes a change in the data's underlying distribution on which the model was originally trained resulting in a performance decline on novel unseen data. To test the impact caused by model drift, the third dataset (dataset3) [16] is employed. The dataset contains 7107 samples of different malware types with no goodware and each entry in the dataset has an arbitrary-length API call extracted by executing the samples in a sandbox. Figure 1 depicts the size of each dataset.

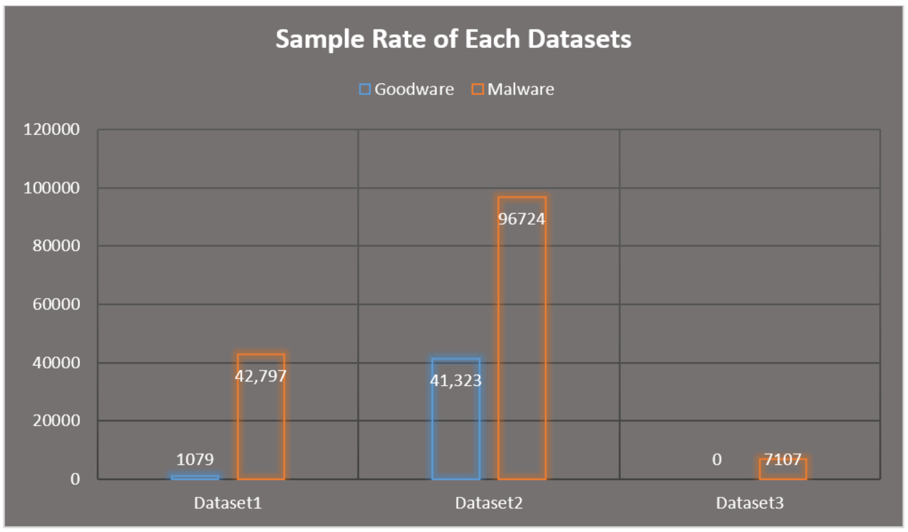
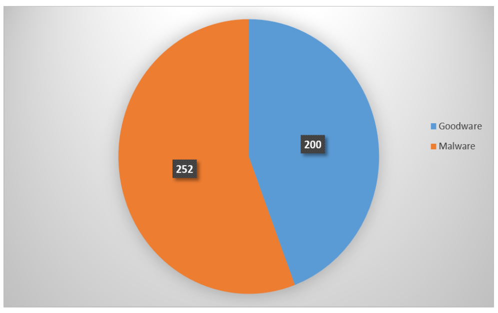


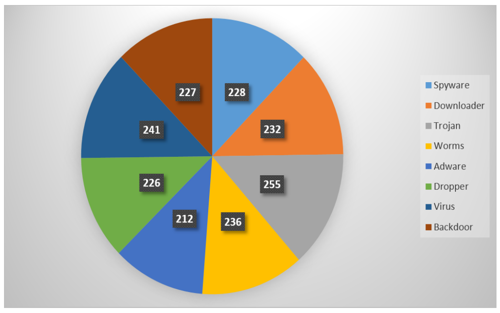
Figure 1. The datasets of malware and goodware samples.

5. The Dictionary of API Calls

Attackers receive benefits from previous attacks to develop new samples of malware with few changes. In this sense, we can infer that the API calls that exist in the malware category are affiliated with a limited dictionary and are frequently utilized in other samples. Investigating the dictionary of API calls used by malware and goodware could provide an opportunity to build a reliable classifier based on API calls, once a sufficient number of reliable and relevant examples are analyzed. Starting from this basis, initial statistics are crucial to understanding the trend of API calls in each category. Python scripts are created to automate the process of extracting the unique API calls for both malware and goodware. The results show that the number of unique API calls in dataset1 is 264 distributed over both categories with 252 and 200 API calls for malware and goodware, respectively, as shown in Figure 2a. Although goodware constitutes a minority in dataset1, 200 API calls were found, which means most of the API calls are shared by malware and goodware in dataset1. This observation is very interesting and shows the limited number of API calls found in both classes which supports our assumption that the malware dictionary is limited, as attackers reuse the same tricks and techniques with few updates. This is supported by an examination of dataset3, which includes 278 distinct API calls shared by all malware families with rates that are extremely close to one another, as shown in Figure 2b.



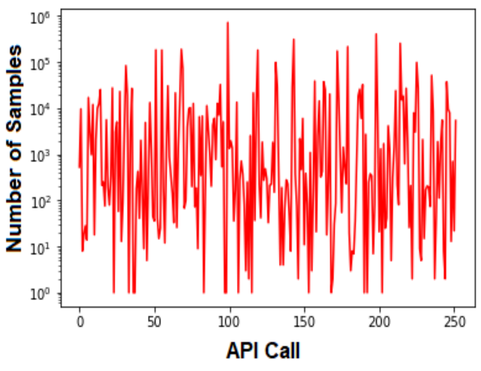
(**a**) Dataset1



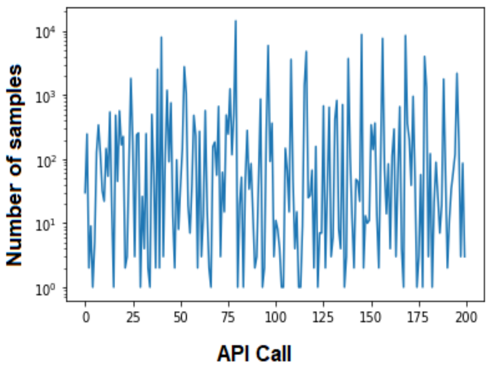
(**b**) Dataset3

Figure 2. The distribution of API calls in dataset1 and dataset3.

The reusability of API calls reflects the trend of attackers' thinking. Figure 3a clearly depicts the higher reuse rate of API calls within the malware which can be understood as a reuse of previous tricks, although the goodware reuses the API calls as shown in Figure 3b. However, the reuse rate is low compared with the case of malware which again confirms our assumption.



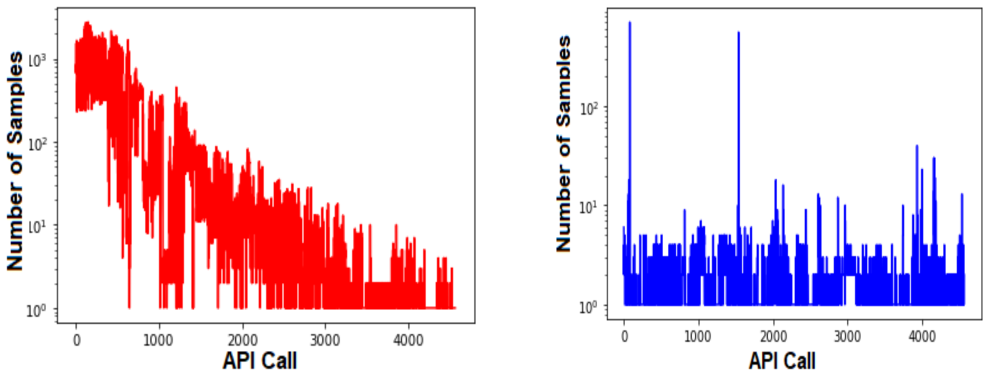
(**a**) Malware



(**b**) Goodware

Figure 3. The reuse rate of API calls in dataset1.

The small number of goodware samples in dataset1 might make the aforementioned conclusions less strong. Also, this depends on the fact that only 100 sequences of API calls can hide a lot of useful information. Therefore, 3135 malware samples were randomly taken from dataset1, and the goodware samples of dataset1 were combined with goodware samples taken from dataset2 to complete a total of 3135 samples to generate a balanced dataset. In order to obtain the complete sequences of API calls, the MD5 hashes of the samples are provided to VirusTotal [42] through the API interface, resulting in the generation of a JSON report. Following this process, the API calls are extracted and analyzed. Once again, the analysis results are consistent with our previous assumption. The number of unique API calls extracted from malware was 4567. This number is very limited compared with the 34,945 API calls achieved from goodware. Furthermore, drawing the reuse rate of the full API calls for both malware and goodware, it can be observed that the reuse rate is still higher in malware as shown in Figure 4. This confirms the limitation API call dictionary for malware class. This shows that our suggested API calls analysis could offer superior discriminatory qualities to be a foundation for a reliable classification model. The next section presents the construction process of a classification model that only depends on an API calls dictionary.



(**a**) Malware (**b**) Goodware

Figure 4. The reuse rate of full-length API calls.

6. The Statistical Classifier STC

The classifier's construction starts by supervising the learning process. In this process, pre-classified samples of malware and goodware are fed for extracting the API calls and recording the frequency of each one. At the end of this process, two dictionaries are created one for malware and the other for goodware.

After completing the learning phase, the classifier can be used to process the unclassified samples. It takes these samples as inputs and produces a binary classification of either malware or goodware. To achieve this, the classifier extracts the API calls from the samples and finds out the frequency of each API call with its associated class. A logarithm of each API call frequency is aggregated and then normalized by the number of API calls in that samplem as shown in Equation (1), to calculate the malicious rate. Ultimately, the class of the sample under test is determined according to the maximum value. A visual representation of workflow and the details of this process are depicted in Algorithm 1 and Figure 5.

$$Malicious_Rate = \sum_{1}^{N} Log(\frac{API_Call_Frequancy}{N}) \tag{1}$$

where N is the number of API calls in the sample under test.

Algorithm 1: Creation of dictionaries and classifier testing

```
Input  : Malware API calls X, goodware API calls Y
1  foreach $x \in X$ do
2      foreach $API_call \in x$ do
3          $API_call_frequencyM$ = ($OccurrencesCount$)
4      end
5      $CreateMalwareDictinary$
6  end
7  foreach $y \in Y$ do
8      foreach $API_call \in x$ do
9          $API_call_frequencyL$ = ($OccurrencesCount$)
10     end
11     $CreateGoodwareDictinary$
12 end
13 $X_{Train}, X_{Test}, Y_{Train}, Y_{Test} \longleftarrow$ Split rate (X,Y) for training
Output: Binary classifier
```

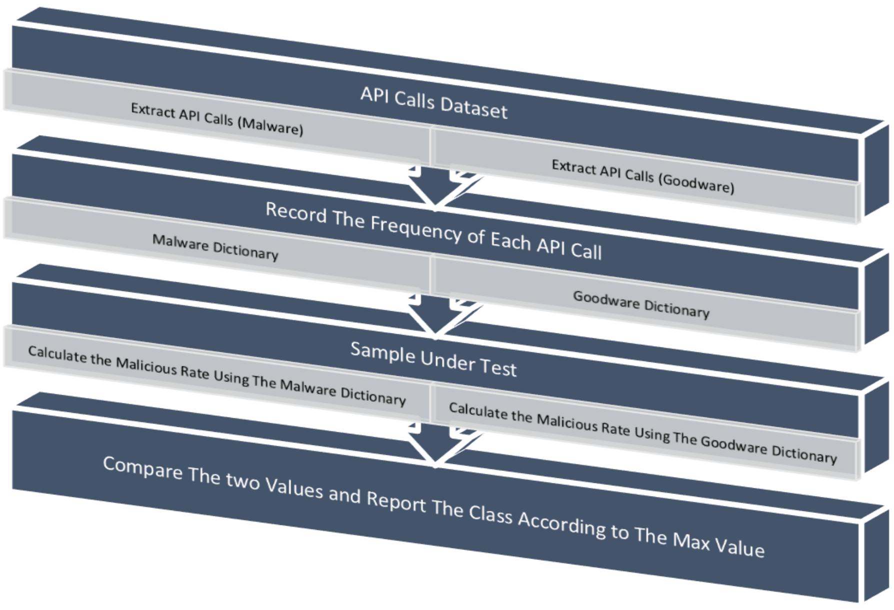

API Calls Dataset
Extract API Calls (Malware)
Extract API Calls (Goodware)
Record The Frequency of Each API Call
Malware Dictionary
Goodware Dictionary
Sample Under Test
Calculate the Malicious Rate Using The Malware Dictionary
Calculate the Malicious Rate Using The Goodware Dictionary
Compare The two Values and Report The Class According to The Max Value

Figure 5. Workflow of the proposed API calls analysis.

7. The Experimental Outcomes of the Statistical Model STC

The results of this section highlight the effectiveness of using API call-based techniques to distinguish between malware and goodware. The goal of this research was to develop an accurate method for identifying potential malware based on the sequence of API calls made during its execution. To conduct the study, a diverse dataset comprising both malware and goodware software samples was collected. There is an unbalanced number of samples in dataset1 utilized in this study. When training the STC model with imbalanced datasets, it can lead to misleading accuracy results. In this study, to mitigate accuracy-related concerns, evaluation metrics such as F1 score, recall, and precision are utilized. Furthermore, the STC model undergoes testing in two scenarios: the first test uses a balanced number of malware and goodware, and an imbalanced number of samples is employed in the second test. The following paragraphs provide a comprehensive overview of the findings and outcomes of this study.

7.1. Experiment 1: Imbalanced Class Distribution

In this experimental setting, the STC model was built and tested using an original unbalanced dataset of 42,797 malware and 1079 goodware samples. However, promising results were achieved in this experiment, as shown in Table 1, especially for recall value. Despite this, the results clearly indicated that the presence of a class imbalance in the dataset had a significant impact on the performance of STC. The accuracy of the model was significantly affected by the class distribution, with a tendency to favor the majority class because of its higher prevalence. Thus, the models showed high accuracy in correctly predicting the majority class (all malware samples were classified correctly) but struggled with the minority class (misclassified all goodware samples), resulting in imbalanced classification results. Also, depending only on the first 100 sequences can hide important API calls which differentiate the goodware from malware.

Table 1. STC performance results.

SCT	
Precision	0.97
Recall	1.0
F1 score	0.98
Accuracy	0.97

7.2. Experiment 2: Balanced Class Distribution

In this test, a balanced dataset was created by ensuring an equal number of instances (3135) for malware and goodware. As stated before, the full-length API call sequences were extracted first, then the dictionaries of each class were built. For this test, 100 random samples are taken for each class during the test process. The results in Table 2 demonstrate that the performance of the STC model provides 0.97 recall as just three samples out of 100 are misclassified as goodware. Also, the detection of the goodware is improved in this test as 62 samples are correctly classified with an FP rate of 38. These results indicate that the STC model provides a high detection rate for malware in experiments 1 and 2.

Table 2. STC performance results.

SCT	
Precision	0.72
Recall	0.97
F1 score	0.80
Accuracy	0.80

7.3. Experiment 3: Drift of STC Model

In the field of machine learning, the usual practice is to train models using a specific dataset that represents the target problem domain. However, over time, the distribution of data may change due to various factors such as new trends, behavior shifts, or novel attack techniques in the case of security-related applications. These changes can lead to a mismatch between the training data and the real-world data that the model encounters in action, which leads to performance degradation. The STC model is exposed to the same phenomenon because it depends entirely on the API calls which may change over time.

This test focuses on examining the effect of model drift by training the STC model with dataset1 and evaluating its performance using dataset3, which consists exclusively of malware without any samples of goodware. To the best of our knowledge, both datasets were collected at different times in 2018 and 2019; therefore, the collection time of these datasets provides reasonable information about the trend of attackers during these years.

As dataset3 comprises malware only, evaluation scales of true positive (TP) and false negative (FN) are used to accurately assess the ability of the model to detect malware. Using dataset1, the full-length API call sequences are extracted to create the dictionaries of goodware and malware.

The results of this analysis are particularly interesting. The STC shows an impressive performance by achieving a TP rate of 7106, which indicates that almost all malware samples were correctly classified. Only one sample was misclassified as goodware, resulting in an impressively low FN rate. These findings highlight the model's robustness and effectiveness in detecting malware which is reinforced by the fact that attackers often rely on similar techniques with minor modifications, resulting in a relatively limited dictionary of malware characteristics. Overall, the results of this experiment show the substantial capability of the model against the drift of the model. This knowledge enables the development of robust and adaptive models that can accurately classify malware while limiting false negatives. The results of the study contribute to enhancing cybersecurity strategies to create time-independent, real-time models in the field of malware detection.

8. Comparisons and Limitations

To draw a comparative analysis with state-of-the-art works in the field, the traditional classifier CatBoost [30] and deep learning classifiers (LSTM and DGCNN) [31] are selected for the purpose. The selection is based on the success of these techniques in creating malware detection models using the sequences of API calls. To make fair comparisons, the same dataset and evaluation metrics are used. Hence, recall, F1 score, and precision are used to discuss the comparison results.

As shown in Table 3, the proposed STC model performs very closely to deep learning algorithms. Deep learning model-based API calls could be built using thousands of sequences. Such a large number of API call sequences can be limited to a specific number. However, in both cases, it is important to note that these solutions often demand substantial computational power and resources. The simple representation of API calls in STC overcomes the curse of dimensionality problem that exists in representing API calls as a bag of words such as those found in CatBoost.

The proposed STC model shows high performance for malware detection using the training and testing datasets and successfully passed the testing of model drift. However, the lack of testing of the model on a daily basis with evolved malware samples is considered a limitation of this work.

Table 3. Comparison with cutting-edge approaches.

	SCT	**CatBoost [30]**	**Model 2 (DGCNN) [31]**	**LSTM [31]**
Precision	0.97	0.96	0.99	0.99
Recall	1.0	0.84	0.99	0.99
F1 score	0.98	0.89	0.99	0.99

9. Conclusions and Future Direction

The objective of this study was to detect malicious software by utilizing malware API calls as classification features. First, the approach involved extracting the initial part of the API calls of length 100. By adopting this method, the study aimed to identify malicious behavior at the earliest possible stage. Nonetheless, this approach conceals a wealth of valuable information, particularly for goodware. To uncover comprehensive insights into both malware and goodware, extracting full-length sequences of API calls proves to be highly beneficial. Furthermore, the reuse rate of API calls offers significant insights into the utilization of identical combinations by malware, thereby uncovering trends followed by malware creators. It is worth noting that the dictionary of API calls used by malware is comparatively limited in comparison to goodware, as attackers frequently resort to employing previously successful tactics.

While there has been considerable success in utilizing time series data of API call sequences with neural network models for malware detection, it is important to note that these solutions often demand substantial computational power and resources.

STC demonstrates shorter training time and lower intrinsic complexity compared to other approaches. The experimental results exhibited a superior performance of STC in detecting malware, indicating its potential for real-time inference on edge devices. Additionally, it has been demonstrated that STC exhibits robustness against model drift.

For future research, it would be intriguing to train STC using a dataset and assess its performance on a daily basis with collected malware samples. Furthermore, as an extension of this study, the STC results can be utilized as a feature in combination with other static or dynamic features to minimize false positives, false negatives, and enhance overall detection accuracy.

Author Contributions: Conceptualization, A.Y.D. and A.A.-N.; Data curation, A.Y.D.; Formal analysis, A.Y.D.; Funding acquisition, A.A.-N. and J.C.; Investigation, A.Y.D.; Methodology, A.Y.D. and A.A.-N.; Project administration, A.A.-N. and J.C.; Resources, A.Y.D.; Software, A.Y.D.; Validation, A.Y.D.; Writing—original draft, A.Y.D.; Writing—review & editing, A.A.-N. and J.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: All data were presented in the main text.

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

References

1. Gobble, M.M. Digitalization, digitization, and innovation. *Res.-Technol. Manag.* **2018**, *61*, 56–59. [CrossRef]
2. Jamsa, K. *Cloud Computing*; Jones & Bartlett Learning: Burlington, MA, USA, 2022.
3. Hartmann, M.; Hashmi, U.S.; Imran, A. Edge computing in smart health care systems: Review, challenges, and research directions. *Trans. Emerg. Telecommun. Technol.* **2022**, *33*, e3710. [CrossRef]
4. Sahani, A.; Sushree, B.B.P. The Emerging Role of the Internet of Things (Iot) in the Biomedical Industry. In *The Role of the Internet of Things (Iot) in Biomedical Engineering*; Apple Academic Press: Palm Bay, FL, USA, 2022; pp. 129–156.
5. Conti, M.; Dargahi, T.; Dehghantanha, A. *Cyber Threat Intelligence: Challenges and Opportunities*; Springer: Berlin/Heidelberg, Germany, 2018.
6. Gandhi, R.; Sharma, A.; Mahoney, W.; Sousan, W.; Zhu, Q.; Laplante, P. Dimensions of cyber-attacks: Cultural, social, economic, and political. *IEEE Technol. Soc. Mag.* **2011**, *30*, 28–38. [CrossRef]
7. Huang, Z.; Wang, Q.; Chen, Y.; Jiang, X. A survey on machine learning against hardware trojan attacks: Recent advances and challenges. *IEEE Access* **2020**, *8*, 10796–10826. [CrossRef]
8. Kim, J.Y.; Cho, S.B. Obfuscated malware detection using deep generative model based on global/local features. *Comput. Secur.* **2022**, *112*, 102501. [CrossRef]
9. Ball, R. Computer Viruses, Computer Worms, and the Self-Replication of Programs. In *Viruses in all Dimensions: How an Information Code Controls Viruses, Software and Microorganisms*; Springer: Berlin/Heidelberg, Germany, 2023; pp. 73–85.
10. Greubel, A.; Andres, D.; Hennecke, M. Analyzing Reporting on Ransomware Incidents: A Case Study. *Soc. Sci.* **2023**, *12*, 265. [CrossRef]
11. Deng, X.; Mirkovic, J. Polymorphic malware behavior through network trace analysis. In Proceedings of the 2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS), Bangalore, India, 4–8 January 2022; pp. 138–146.
12. Braue, D. Global Ransomware Damage Costs Predicted to Exceed $265 Billion by 2031. Available online: https://rb.gy/premy (accessed on 17 June 2022).
13. Kaspersky. Cybercriminals Attack Users with 400,000 New Malicious Files Daily—That Is 5% More Than in 2021. Available online: https://rb.gy/xwak5 (accessed on 17 June 2022).
14. Akhtar, M.S.; Feng, T. Malware Analysis and Detection Using Machine Learning Algorithms. *Symmetry* **2022**, *14*, 2304. [CrossRef]
15. Kimmell, J.C.; Abdelsalam, M.; Gupta, M. Analyzing machine learning approaches for online malware detection in cloud. In Proceedings of the 2021 IEEE International Conference on Smart Computing (SMARTCOMP), Irvine, CA, USA, 23–27 August 2021; pp. 189–196.
16. Catak, F.O.; Yazı, A.F. A benchmark API call dataset for windows PE malware classification. *arXiv* **2019**, arXiv:1905.01999.
17. Wagener, G.; State, R.; Dulaunoy, A. Malware behaviour analysis. *J. Comput. Virol.* **2008**, *4*, 279–287. [CrossRef]
18. Aboaoja, F.A.; Zainal, A.; Ghaleb, F.A.; Al-rimy, B.A.S.; Eisa, T.A.E.; Elnour, A.A.H. Malware detection issues, challenges, and future directions: A survey. *Appl. Sci.* **2022**, *12*, 8482. [CrossRef]

19. Ucci, D.; Aniello, L.; Baldoni, R. Survey of machine learning techniques for malware analysis. *Comput. Secur.* **2019**, *81*, 123–147. [CrossRef]
20. Naz, S.; Singh, D.K. Review of machine learning methods for windows malware detection. In Proceedings of the 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kanpur, India, 6–8 July 2019; pp. 1–6.
21. Banin, S.; Shalaginov, A.; Franke, K. Memory Access Patterns for Malware Detection. Available online: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2455297/memoryaccesspatterns.pdf?sequence=1 (accessed on 30 October 2023).
22. Hwang, J.; Kim, J.; Lee, S.; Kim, K. Two-stage ransomware detection using dynamic analysis and machine learning techniques. *Wirel. Pers. Commun.* **2020**, *112*, 2597–2609. [CrossRef]
23. Belaoued, M.; Boukellal, A.; Koalal, M.A.; Derhab, A.; Mazouzi, S.; Khan, F.A. Combined dynamic multi-feature and rule-based behavior for accurate malware detection. *Int. J. Distrib. Sens. Netw.* **2019**, *15*, 1550147719889907. [CrossRef]
24. Aboaoja, F.A.; Zainal, A.; Ali, A.M.; Ghaleb, F.A.; Alsolami, F.J.; Rassam, M.A. Dynamic Extraction of Initial Behavior for Evasive Malware Detection. *Mathematics* **2023**, *11*, 416. [CrossRef]
25. Yazi, A.F.; Çatak, F.Ö.; Gül, E. Classification of methamorphic malware with deep learning (LSTM). In Proceedings of the 2019 27th Signal Processing and Communications Applications Conference (SIU), Sivas, Turkey, 24–26 April 2019; pp. 1–4.
26. Qiao, Y.; Yang, Y.; Ji, L.; He, J. Analyzing malware by abstracting the frequent itemsets in API call sequences. In Proceedings of the 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, Melbourne, VIC, Australia, 16–18 July 2013; pp. 265–270.
27. Hansen, S.S.; Larsen, T.M.T.; Stevanovic, M.; Pedersen, J.M. An approach for detection and family classification of malware based on behavioral analysis. In Proceedings of the 2016 International Conference on Computing, Networking and Communications (ICNC), Kauai, HI, USA, 15–18 February 2016; pp. 1–5.
28. Daeef, A.Y.; Al-Naji, A.; Chahl, J. Features Engineering for Malware Family Classification Based API Call. *Computers* **2022**, *11*, 160. [CrossRef]
29. 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]
30. Cannarile, A.; Dentamaro, V.; Galantucci, S.; Iannacone, A.; Impedovo, D.; Pirlo, G. Comparing deep learning and shallow learning techniques for api calls malware prediction: A study. *Appl. Sci.* **2022**, *12*, 1645. [CrossRef]
31. Oliveira, A.; Sassi, R. Behavioral malware detection using deep graph convolutional neural networks. *Techlixiv* 2019, *preprint*. [CrossRef]
32. Schofield, M.; Alicioglu, G.; Binaco, R.; Turner, P.; Thatcher, C.; Lam, A.; Sun, B. Convolutional neural network for malware classification based on API call sequence. In Proceedings of the 8th International Conference on Artificial Intelligence and Applications (AIAP 2021), Zurich, Switzerland, 23–24 January 2021.
33. Ali, M.; Shiaeles, S.; Bendiab, G.; Ghita, B. MALGRA: Machine learning and N-gram malware feature extraction and detection system. *Electronics* **2020**, *9*, 1777. [CrossRef]
34. Li, C.; Zheng, J. API call-based malware classification using recurrent neural networks. *J. Cyber Secur. Mobil.* **2021**, 617–640. [CrossRef]
35. Eskandari, M.; Khorshidpur, Z.; Hashemi, S. To incorporate sequential dynamic features in malware detection engines. In Proceedings of the 2012 European Intelligence and Security Informatics Conference, Odense, Denmark, 22–24 August 2012; pp. 46–52.
36. Vinayakumar, R.; Alazab, M.; Soman, K.; Poornachandran, P.; Al-Nemrat, A.; Venkatraman, S. Deep learning approach for intelligent intrusion detection system. *IEEE Access* **2019**, *7*, 41525–41550. [CrossRef]
37. Tang, M.; Qian, Q. Dynamic API call sequence visualisation for malware classification. *IET Inf. Secur.* **2019**, *13*, 367–377. [CrossRef]
38. 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]
39. Zhang, Z.; Qi, P.; Wang, W. Dynamic malware analysis with feature engineering and feature learning. In Proceedings of the AAAI Conference on Artificial Intelligence, New York, NY, USA, 7–12 February 2020; Volume 34, pp. 1210–1217.
40. Oliveira, A. Malware Analysis Datasets: API Call Sequences. Available online: https://ieee-dataport.org/open-access/malware-analysis-datasets-api-call-sequences (accessed on 10 June 2023).
41. Lalwani, P. MalwareData. Available online: https://github.com/saurabh48782/Malware_Classification (accessed on 10 June 2023).
42. VirusTotal. VirusTotal API v3 Overview. Available online: https://developers.virustotal.com/reference/overview (accessed on 10 June 2023).