computers

MDPI

Article

Features Engineering for Malware Family Classification Based API Call

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

[1] Technical Institute for Administration, Middle Technical University, Baghdad 10010, Iraq
[2] Electrical Engineering Technical College, Middle Technical University, Baghdad 10022, Iraq
[3] School of Engineering, University of South Australia, Mawson Lakes, SA 5095, Australia
* Correspondence: ali_al_naji@mtu.edu.iq

Abstract: Malware is used to carry out malicious operations on networks and computer systems. Consequently, malware classification is crucial for preventing malicious attacks. Application programming interfaces (APIs) are ideal candidates for characterizing malware behavior. However, the primary challenge is to produce API call features for classification algorithms to achieve high classification accuracy. To achieve this aim, this work employed the Jaccard similarity and visualization analysis to find the hidden patterns created by various malware API calls. Traditional machine learning classifiers, i.e., random forest (RF), support vector machine (SVM), and k-nearest neighborhood (KNN), were used in this research as alternatives to existing neural networks, which use millions of length API call sequences. The benchmark dataset used in this study contains 7107 samples of API call sequences (labeled to eight different malware families). The results showed that RF with the proposed API call features outperformed the LSTM (long short-term memory) and gated recurrent unit (GRU)-based methods against overall evaluation metrics.

Keywords: malware classification; Jaccard similarity; API call sequence

Citation: Daeef, A.Y.; Al-Naji, A.; Chahl, J. Features Engineering for Malware Family Classification Based API Call. *Computers* **2022**, *11*, 160. https://doi.org/10.3390/computers11110160

Academic Editor: Paolo Bellavista

Received: 9 October 2022
Accepted: 7 November 2022
Published: 11 November 2022

1. Introduction

As of May 2022, more than 9 million new malware instances have been released, as stated in a report published by AV-TEST [1]; there are a total of 1363.92 million known malware instances currently active in the environment, requiring significant and ongoing technological development to counter new attacks. Furthermore, attackers are constantly improving their skills in writing malicious programs to evade existing protection mechanisms. Conversely, security software developers are updating their methods and techniques to block such zero-day attacks. This competitive "game" has led to the production of metamorphic and polymorphic malware, which are more advanced than regular malicious software. Metamorphic malware programs can change their functions and structures with each iteration; polymorphic malware involves composing new malware based on previous malware with new features. These capabilities make sensing and classifying specimens complex processes [2].

Despite an attacker's tricks, malicious programs need the host operating system and application services to complete the malicious activity [3]. Hence, the malware requests the API service of the Windows operating system to perform the malicious action. Such service requests generate malicious behaviors employed for detection and classification purposes. As stated by [4], the malware cannot obfuscate or hide the use of the system's API call.

A large number of malicious programs are classified into families that share common characteristics. Malware family identification is crucial to researchers because it helps to expedite the detection and mitigation procedures of incident responders [5,6].

The process of labeling malware into families is not easy because labeling depends on the outputs of the antivirus programs, which are not always identical, e.g., VirusTotal [3] provides the analysis results from several antivirus engines with malware labeling. As

malware in the same family shares common attributes, capturing the API call sharing and sequences could provide a strong indication of the probable family of the malware.

There is little research on classifying malicious programs into families. Therefore, this research analyzes the API calls and sequences carried out by malware in the Windows operating system. The main objective of this research is how API calls (as features) can classify malware into families accurately. Moreover, we examine the usefulness of visual techniques in data analysis and combine computationally sophisticated algorithms with human intuition [7]. Finally, we suggest an algorithm to extract and represent attributes to improve the performances of existing tools for classifying malware into families. The proposed classification system is based on the bag-of-words (bow) model and machine learning.

2. Techniques for Malware Analysis

Malware analysis describes how to investigate a malicious file, in order to understand its behavior and examine the various parts of harmful software. Malware analysis aids in determining and reducing the damage caused by attackers. Moreover, malware analysis assists security experts and incident responders in covert "indications" that ought to be prevented [8]. It is possible to conduct a malware analysis in a static, dynamic, or mixed way.

2.1. Static Analysis

For simple static analysis, the malware does not need to be run. Instead, static analysis searches the file for clues to malicious intent. Identifying libraries, malicious infrastructures, or packaged files may be valuable. Static analyses have been used to identify many malware variants [9–12]. The benefits of a static analysis include its speed and not requiring a controlled environment to run harmful code. However, obfuscation techniques, such as metamorphic and polymorphic malware, which conceal the harmful payload and make it unrecognizable, hinder the benefits of this technique [13,14]. Static features are not as helpful for malware variants that are characterized by frequent changes in dangerous structures, codes, functions, or even by completely rewriting themselves.

2.2. Dynamic Analysis

In contrast, dynamic analysis does not require reverse engineering of malware. This method relies on executing malware in an isolated environment to record the file behavior, such as API calls, memory writing, registry access, and other activities. Sandbox technology provides a secure environment for suspicious file analysis. Since it monitors how malware interacts with the operating system in a quarantined environment, it helped in developing an effective defense mechanism. This technique is considered the most effective for polymorphic and metamorphic classification. In this context, many dynamic analysis techniques are proposed [15–17] to detect malware files.

3. Related Works

The amount of malware spreading has increased significantly; as a reaction from the security community, several studies have been conducted to automatically detect and analyze malware. As attackers design malware for many harmful purposes, malware poses different behaviors on the attacked machine, which results in the existence of many malware families. Malware family identification is crucial to researchers because it helps to expedite the detection and mitigation procedures of incident responders [5,6].

Machine learning solutions have great success in the detection and classification of malware families [18]. Therefore, researchers have employed various machine-learning techniques with different static and dynamic features for malware classification [19–22]. Malware producers have created a variety of malware obfuscation techniques, including encryption, packing, and API call insertion to avoid detection [23]. The dynamic detection method is unaffected by static obfuscation techniques, such as encryption and packing, because they often do not alter the behavior of malware. For this reason, dynamic detection

has received a lot of attention in the field of malware detection. However, for API call-based solutions, the obfuscation method using rearrangement and insertion in the sequences of API calls may result in the failure of the detection system. For instance, authors of a black-box attack [24] focus on generating attacks against API call-based models. Aggressive patterns utilize both static features and API calls. Research shows that such attacks are effective against a variety of classification models.

In order to lessen the effects of obfuscation methods on malware detection, the authors in [25] used Cuckoo Sandbox to extract the sequence of API calls from different malware families to provide a public dataset and employ the deep learning model to classify each malware family. Hansen et al. [26] and Qiao et al. [27] proposed a malware identification system with features related to API calls. In order to classify malware families, the first author translated dynamic analysis findings from the Cuckoo Sandbox investigation into API call sequences. In contrast to Hansen's findings, the authors of [27] classified malware based on the API call sequences and their frequencies. The Cuckoo Sandbox and CWSandbox were used to acquire the API call sequences. Typically, the API requests take the form of sequences with different lengths, making feature extraction from these sequences a complex process. Therefore, researchers thought about employing deep learning models, such as RNNs (recurrent neural networks) using API call sequences. RNN is highly efficient at processing time series sequences, especially in the natural language processing field. Li et al. [28] presented a classification model for malware families using the RNN model. Long API call sequences are used as classification features for variants of malware. The binary classification presented by Eskandari et al. [22] distinguished malware using RNNs and features extracted from API calls. A deep learning and visualization method for classifying malware was proposed by Tang et al. [29]. Dynamic analysis was used to retrieve API calls, after which, feature images that depicted malware behavior in accordance with color mapping guidelines were created. CNN (convolutional neural network) was then used to categorize the feature images. A single-layer LSTM model for binary classification was used by Yazi et al. [25] to offer malware family recognition. Additionally, Catak et al. [30] used API calls to categorize malware families, they suggested a single-layer LSTM and a two-layer LSTM in their work. In particular, they initially suggested an LSTM to obtain API sequences and that these features be used as input to a CNN for binary classification. Some researchers mapped API call sequences as TF-IDF vectors of features, such as in [31].

The API calls and their sequences constitute the basis for the above-mentioned family classification methods. However, extracting features from API calls to differentiate malware families is still challenging. Most works have tackled malware classification as a binary problem and focused on the accuracy provided by API call features. However, using the visualization method to extract the hidden patterns and correlation of API calls among different malware families was not highlighted in previous works. Based on the aforementioned findings, this paper uses visualization techniques to uncover the hidden patterns of API call sequences and provide a deep understanding of the sequence correlations of API calls among different malware families. Moreover, we provide a classification algorithm to classify different malware families.

4. Windows API Call Dataset

The publicly available dataset of malware API calls found in [3] was used in this research. The total number of samples was 7107, categorized into 8 families as depicted in Figure 1. The eight families were Trojan, worm, downloader, spyware, virus, backdoor, adware, and dropper with no goodware samples. The dataset contains malware labels and an arbitrary length of API calls for each record. These API calls are collected by running windows malware in an isolated environment using a Sandbox. Python scripts are written to extract some important statistics for initially understanding the dataset. The API call sequence length ranges from ten to thousands and the average length of sequences per class is presented in Table 1. Spyware had the longest average API call sequence, which can be justified due to the purpose of spyware in recording user internet movement as

well as other activity data. As the total number of unique API calls found in the dataset is 278, finding the distribution of overall malware families indicates the API call correlation. Table 2 depicts the number of API calls each malware family shares; Trojan shares the most API calls. Extraction of the most frequently utilized API calls from each malware family may provide insight into the thought processes of attackers. Figure 2 shows the top 30 most frequently used API calls found in each family. When extracting the shared API calls among all malware families from the top 30, most are related to network connection activities for attackers. These shared API calls are listed in Table 3.

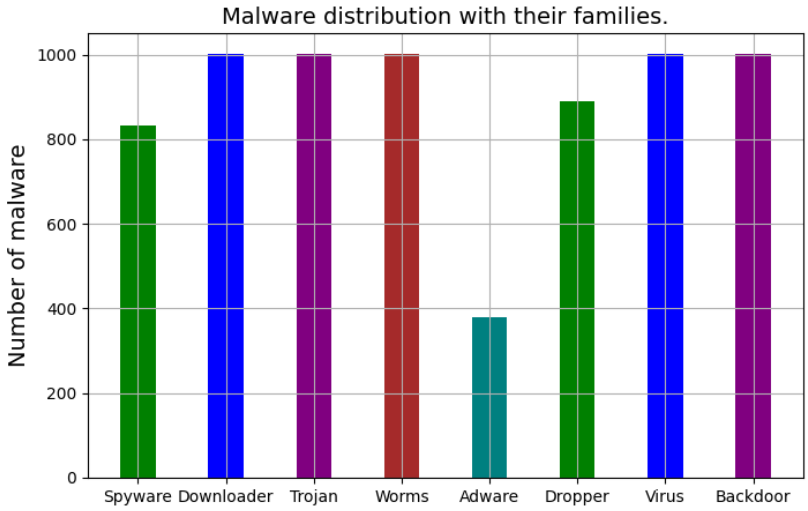


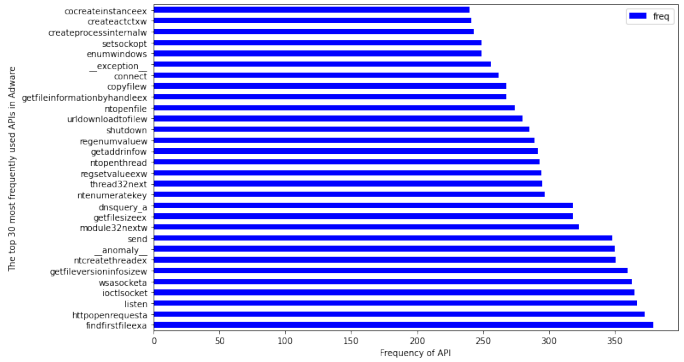
Figure 1. Number of samples in each malware family.

Table 1. API sequence length (average).

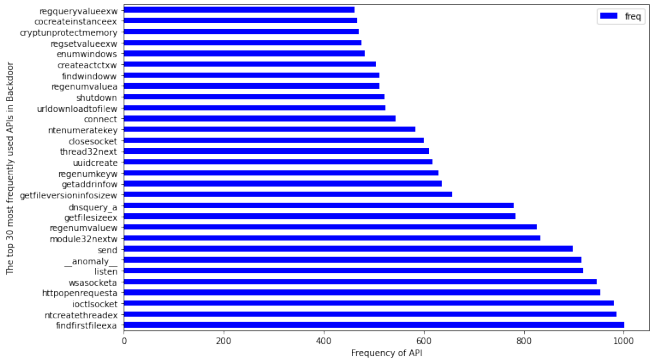
Family	Average of Sequence Length
Spyware	46,951
Downloader	6522
Trojan	13,818
Worms	33,614
Adware	6867
Dropper	16,008
Virus	18,369
Backdoor	11,293

Table 2. Number of API calls found in each family.

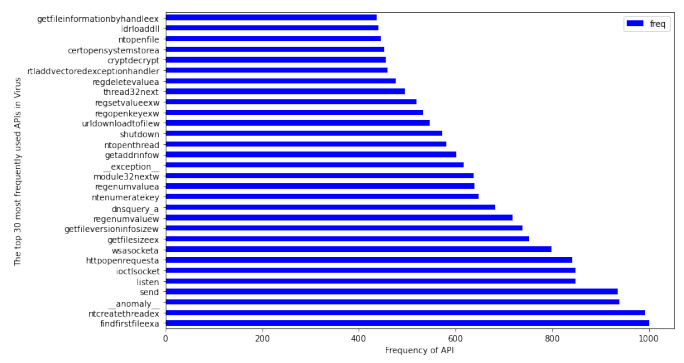
Family	Number of API
Spyware	228
Downloader	232
Trojan	255
Worms	236
Adware	212
Dropper	226
Virus	241
Backdoor	227



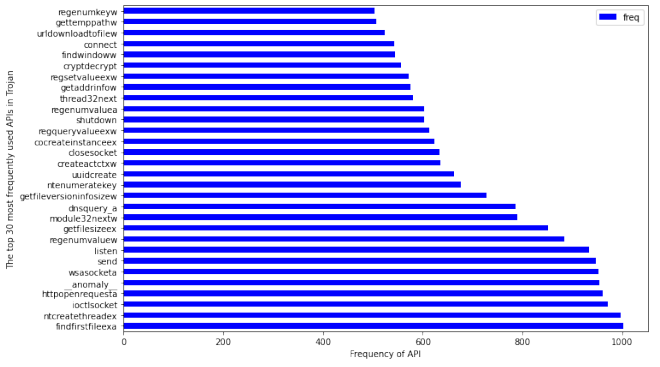
(**a**)



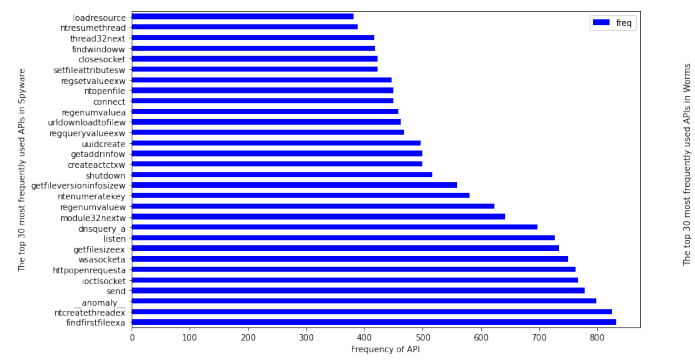
(**b**)



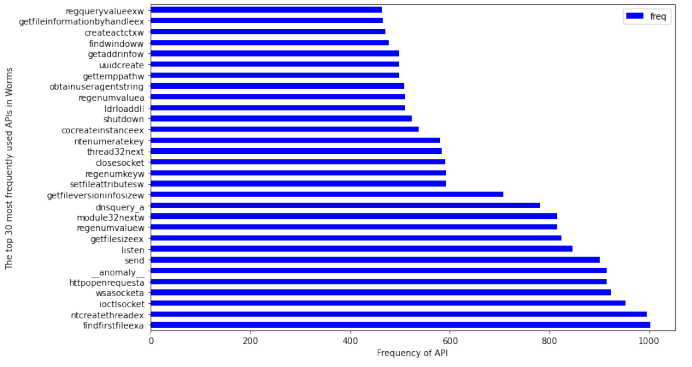
(**c**)



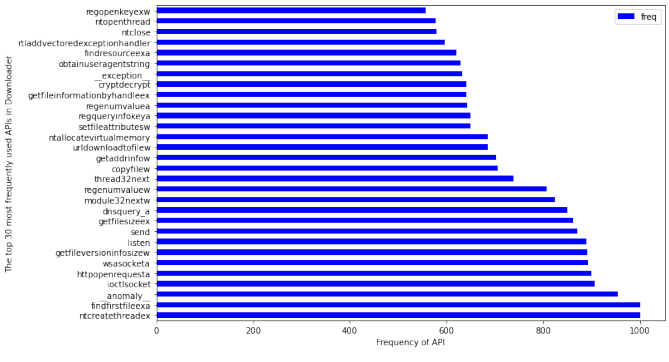
(**d**)



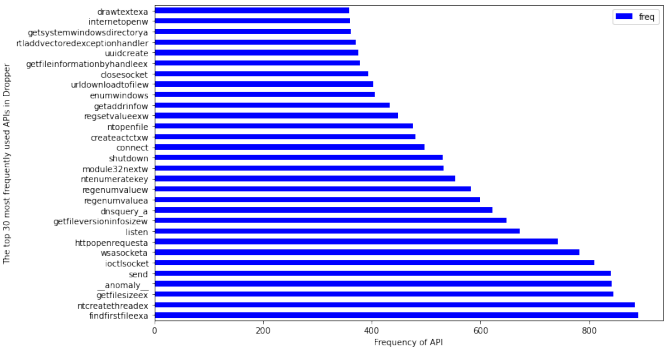
(**e**)



(**f**)



(**g**)



(**h**)

Figure 2. The top 30 most frequently used API calls in each malware family. (**a**) Adware; (**b**) Backdoor; (**c**) Virus; (**d**) Trojan; (**e**) Spyware; (**f**) Worms; (**g**) Downloader; (**h**) Dropper.

Table 3. Shared API calls among the top 30.

Shared API Calls	
listen	getfileversioninfosizew
ntcreatethreadex	regenumvaluew
httpopenrequesta	dnsquery_a
findfirstfileexa	module32nextw
ioctlsocket	getfilesizeex
send	getaddrinfow
__anomaly__	wsasocketa

The unique API call occurrence in each family can provide good features to recognize the malware class in this context; Table 4 describes each unique API call with the number of samples in which they occur. According to the above statistics, malware families share most of the 278 API calls, which makes class separation a challenging job. Therefore, more statistical analyses and visualizations are required to explore possible hidden patterns.

Table 4. Unique API in each family.

Family	Unique API	Description	No. Samples Occurrence
Spyware	rtlcreateuserthread	The malware calls this API by passing the address of the load library to this API call to make the remote process run the DLL on behalf of the malicious code.	7
Downloader	createdirectoryexw	Creates a new directory	2
	getusernameexa	Retrieves the user's name	1
Trojan	wsasend	Sends data on a connected socket	1
	rtldecompressfragment	Decompress part of a compressed buffer	1
	rtlcompressbuffer	For easy file compression implementation	1
	cryptgenkey	Decrypt the malware's configurations	1
	ntdeletefile	Delete specific file	1
	cryptdecodeobjectex	Decode the structure of the encryption algorithm	1
Worms	recvfrom	To transfer data using UDP connections.	1
Adware	-	-	-
Dropper	-	-	-
Virus	certopensystemstorea	The local system's certificates can be accessed using this function.	1
	netgetjoininformation	Function returns data on the specified computer's join status	1
	cryptunprotectdata	Retrieve and decode the saved logins	1
	findwindowexw	Finds a window whose class name and window name match the given strings.	1
	setinformationjobobject	Restrictions for a job object	1
	findfirstfileexa	Directory search for a file or sub-folder with the supplied name and properties.	1
Backdoor	-	-	-

5. Jaccard Similarity

Visualizing the shared API calls found in each malware family provides the common characteristics of that family. Visualization techniques are used to answer the important question: to what extent do the malware samples of specific families use the same combinations of API calls? Network visualization of samples sharing API calls is used for this purpose. It is possible to compare sets of unordered collections of elements using the Jaccard similarity metric [32]. Although different techniques for similarity are found in the scientific community, such as Euclidean L2, L1 distance, cosine distance, and others, the Jaccard similarity index is still the most widely used technique. The Jaccard similarity

between two sets is the shared features divided by the total number of the features. Considering extraction of the unique API calls of malware A and B, the intersection of A and B provides the features that are shared by both malware. Figure 3 illustrates this by depicting three pairs of malware attributes that were taken from three different malware samples. Each case shows the features that are unique to the two sets, the features that both sets share, and the Jaccard value that results from the supplied malware examples and features.

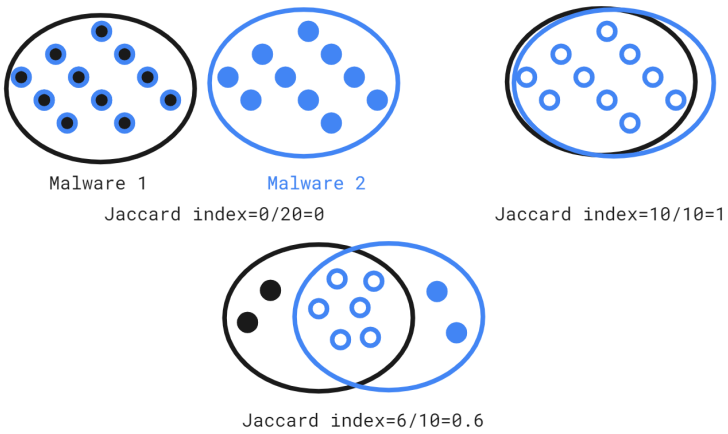


Figure 3. An illustration of the concept underlying the Jaccard index in graphic form.

The similarity index is 1 if two malware samples share the same API combination, and 0 for a completely dissimilar combination. This can be expressed in Equation (1).

The process depicted in Figure 4 was used to visualize the shared API calls based on the Jaccard similarity. Experimentally, the first 500 samples of each malware family were used (except adware was 300). The distinctive API calls were collected for the malware to calculate the similarity score between all malware pair samples belonging to a given family. Following the extraction process, repeatedly comparing the attributes of each pair of malware samples by using the Jaccard index and the API call-sharing graph was then constructed. To achieve this, the cutoff point for how many API calls the two samples should share was chosen; this study used 0.5 as a threshold value as shown in Algorithm 1. A malware pair was linked for visualization if the Jaccard value of that pair was higher than the threshold. The Python library NetworkX [33] was used to generate the network dot file and the open-source network visualization tool Graphviz [34] was used to visualize the dot file.

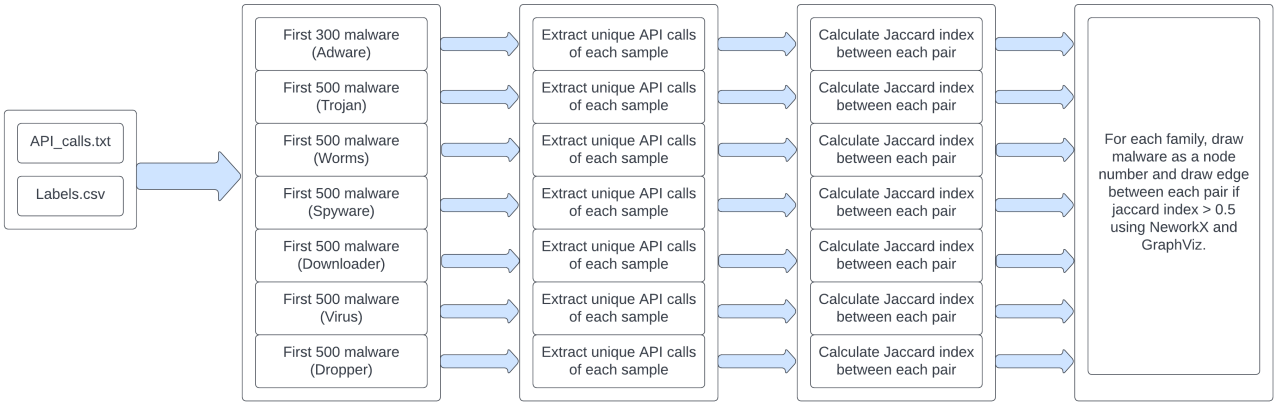


Figure 4. API call visualization workflow using the Jaccard similarity index.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \quad (1)$$

Algorithm 1: Creating Networkx file.

Input : Unique API call sets X

foreach $x \in X$ **do**
  foreach $x_{n+1} \in X$ **do**
    Intersection = len($x_n \cap x_{n+1}$)
    Union = len($x_n \cup x_{n+1}$)
    Similarity index = $Intersection/Union$
    if *Similarity index* > 0.5 **then**
      Create graph edge
    end
  end
end

Output: Networkx dot file

NetworkX is a Python library used to build and manipulate complex networks. It is used to examine complex networks represented as node and edge-based graphs. A node denotes an element, while an edge links two nodes to show their connections as shown in Figure 5, where the nodes are labeled from A to E. To visualize API calls, the graph's nodes are numbered according to the sequence of the malware sample. For example, the first 500 malware samples of the Dropper family were drawn as nodes numbered from 1 to 500. The node names and the edge information according to the Jaccard index were stored as a dot file containing graph description language as represented in Figure 6. By creating the dot file, Graphviz was used to process it to generate PNG representation.

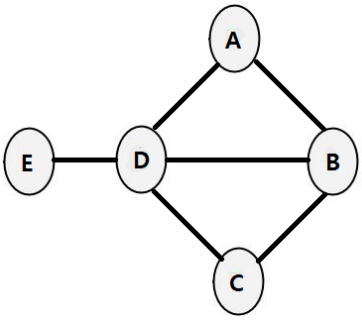


Figure 5. Nodes and edges in Networkx.

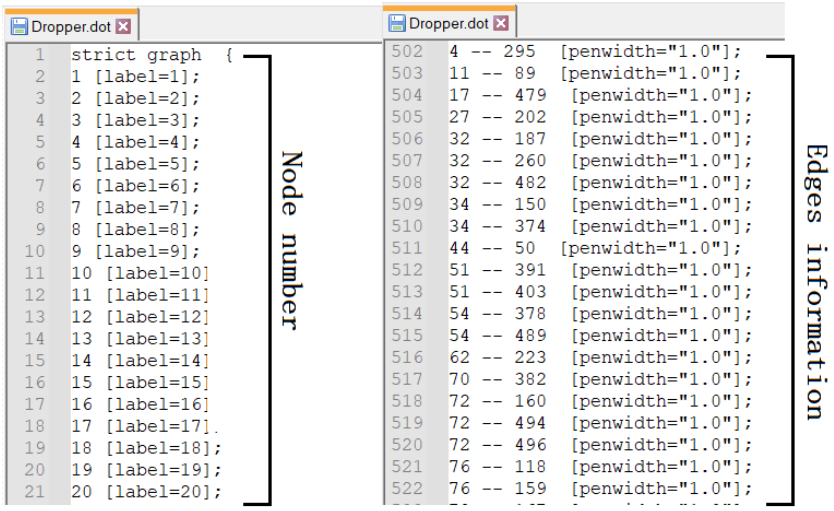
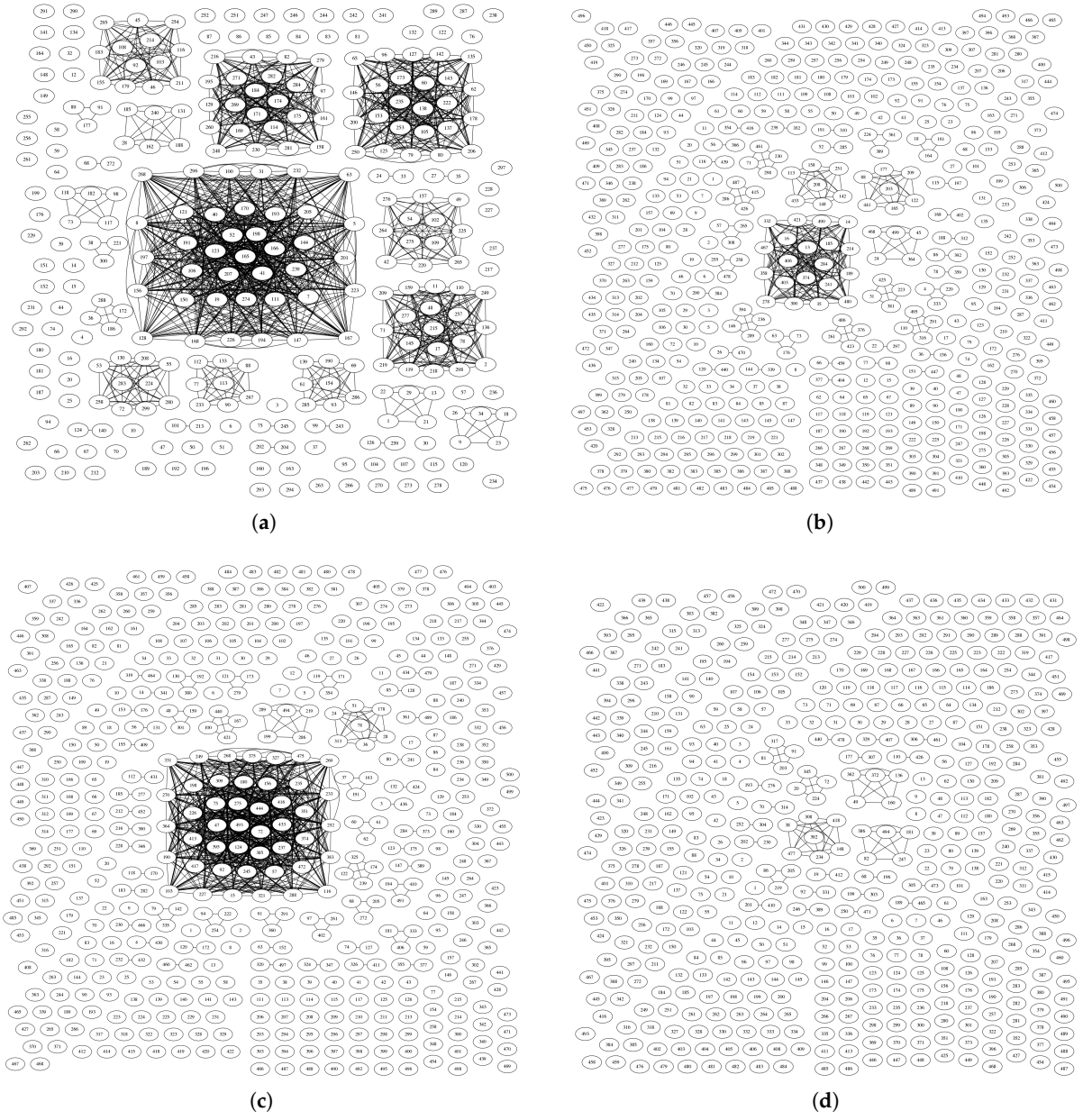


Figure 6. The content of the Networkx dot file.

6. Network Visualization of Jaccard Similarity

The resulting networks of API call sharing for each family are shown in Figure 7. It is very clear that sharing relationships were very weak within the same family, and this means that most of the malware samples used different combinations of API calls, which weakened their reliability in classifying malware families. It also weakened the reliability of the API call sequence because it gave inaccurate results. This can be justified for a number of reasons, e.g., malware family labeling in the dataset may be prone to error. Moreover, most malware creators insert additional API call sequences to mask the malicious behavior of malware.



(**a**) (**b**)

(**c**) (**d**)

Figure 7. *Cont*.

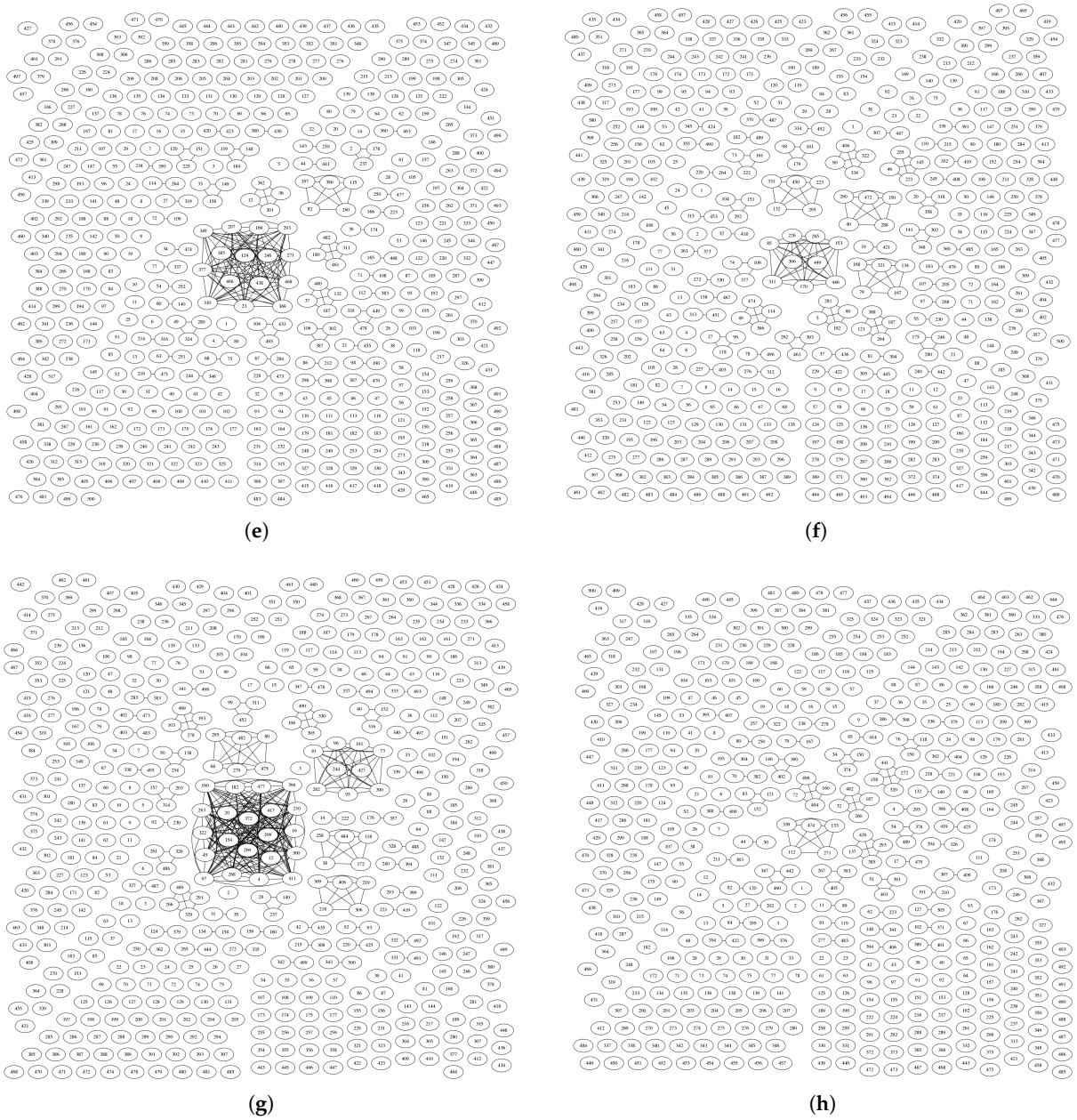


Figure 7. API call similarity graph for each malware family. (**a**) Adware; (**b**) Backdoor; (**c**) Virus; (**d**) Trojan; (**e**) Spyware; (**f**) Worms; (**g**) Downloader; (**h**) Dropper.

One interesting observation about the similarity graph is that some malware samples are tightly linked, which means they belong to the same attack group or the same compilation tools. The adware graph shows a more connected group compared with other families due to its very specific purpose. Depending on the visualization results, using API calls, such as a bag-of-words (with 0 and 1), or employing the sequences of these API calls with deep learning leads to low accuracy classification results. As all malware families share most of the 278 API calls that exist in a dataset, using the frequency of occurrence of all API calls can provide a simple and more accurate classification solution.

7. API Call Frequency-Based Classification

The malware dataset [3] was created by executing the malware samples in the Cuckoo Sandbox, which is a wildly used malware analysis environment. This tool monitors the behavior of each malware and generates reports, including dynamic and static analysis

results, such as API calls and hash values. Then, the hash value of each sample is fed to VirusTotal, which analyses each sample with different antivirus tools. As different antivirus engines produce different labels for the same sample, the class of the malware is determined by considering the majority agreement among the results of antivirus engines. The labeled dataset was finally produced by matching the malware classes and the sequences of API calls.

In order to convert the raw data into a format suitable for classification algorithms, Python code was used to extract all 278 unique API calls that existed in the dataset. These API calls were arranged as a bag-of-words; for each malware sample, the frequency of each API call was recorded. Therefore, the feature vector length was 278 and each feature was the frequency of the corresponding API call in that sample. Algorithm 2 depicts the process of feature extraction. The histogram of the feature distribution is visualized in Figure 8. This histogram shows many zero values for the API calls in the extracted features. Adware presents the best histogram distribution.

Algorithm 2: Feature extraction and classifier learning.

Input : Dataset of malware API calls X, family class Y

foreach $x \in X$ **do**
  foreach $API_call \in x$ **do**
    $API_call_frequency = (Number_Of_Occurrences)$
  end
  $Create feature vector$
end
$X_{Train}, X_{Test}, Y_{Train}, Y_{Test} \longleftarrow$ Split rate (X,Y) 0.8 for training

Output: Malware class classifier

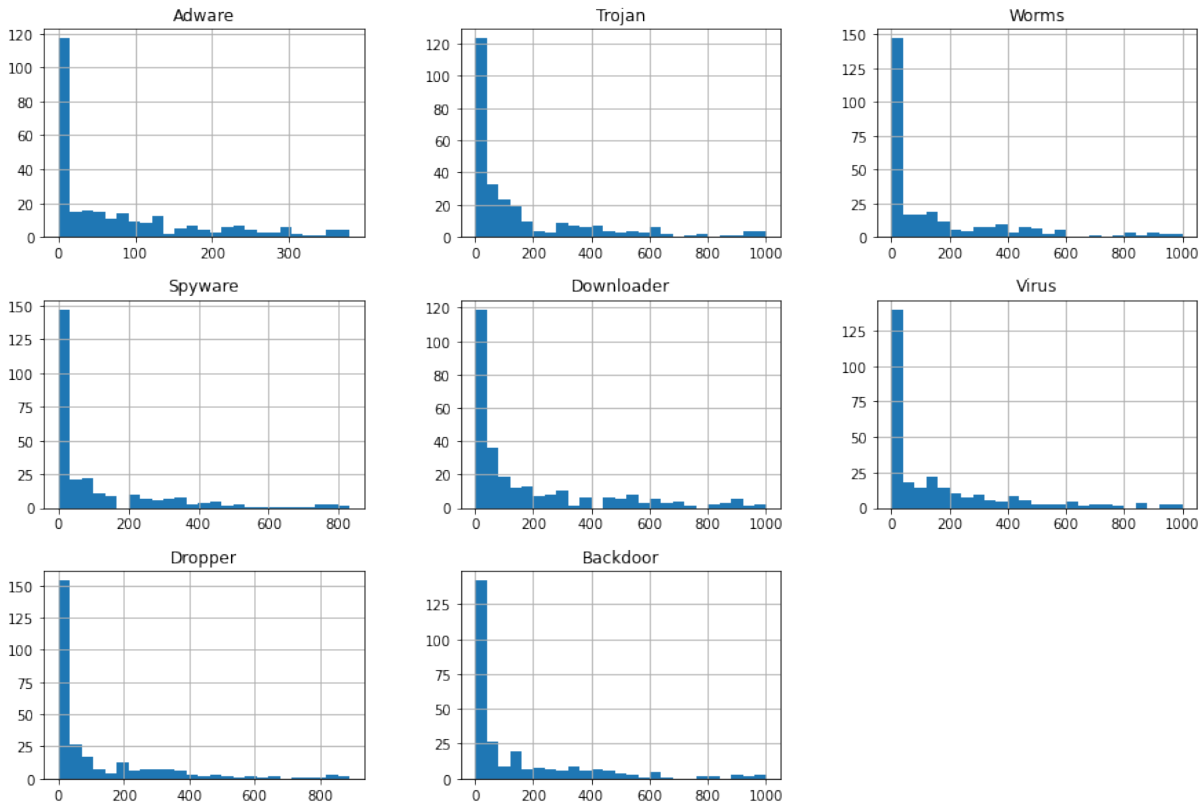


Figure 8. Histogram distribution of the extracted features.

The testing classifiers were chosen based on their widespread use and the success provided by them in the field of malware classification. Therefore, random forest (RF), support vector machine (SVM), and k-nearest neighborhood (KNN) were used in this research. The Python machine learning library scikit-learn was used with the default parameters to implement the classification process.

8. Experimental Results

The benchmark dataset was imbalanced in some malware families, such as Adware and Spyware. Hence, accuracy evaluation was not enough to identify the best classifier and make fair comparisons with other research [28,30]; the same evaluation metrics, precision, recall, and F1 score were used to present the results.

By examining the results shown in Table 5, RF and SVM show trade-offs in some evaluation values but RF provides the best classification performance compared to all of the evaluation metrics, and kNN is the worst. The results are based on the visualization analysis, where the highest evaluation metrics are noted for the Adware class and the worst classification results are observed for Spyware and Trojan. Referring to Figure 7, the lowest API call sharing was drawn for Trojan followed by Spyware. This interprets the low classification results for these families. The comparisons with the state-of-the-art works that employ deep learning methods, such as LSTM and GRU, are shown in Table 6. On the one hand, the proposed method in this work using RF outperforms the deep learning techniques in terms of the higher evaluation metrics for all malware families, and on the other hand, it provides a simple traditional malware classifier compared with deep learning techniques, which require more processing resources and longer times during the learning processes.

Table 5. Classifiers performance results.

	Spyware	Downloader	Trojan	Worms	Adware	Dropper	Virus	Backdoor
RF								
Precision	0.49	0.77	**0.43**	**0.66**	**0.92**	**0.55**	**0.87**	**0.62**
Recall	**0.46**	**0.74**	0.40	**0.66**	**0.87**	**0.71**	**0.77**	**0.66**
F1 score	**0.48**	**0.75**	**0.42**	**0.66**	**0.89**	**0.62**	**0.82**	**0.64**
SVM								
Precision	**0.58**	**0.83**	0.19	0.62	0.91	0.50	0.74	**0.64**
Recall	0.27	0.39	**0.79**	0.26	0.55	0.43	0.41	0.27
F1 score	0.37	0.53	0.31	0.36	0.69	0.46	0.53	0.38
kNN								
Precision	0.43	0.63	0.41	0.47	0.71	0.37	0.71	0.44
Recall	0.39	0.63	0.31	0.37	0.82	0.49	0.61	0.60
F1 score	0.41	0.63	0.36	0.41	0.76	0.42	0.66	0.51

Table 6. Comparison with state-of-the-art works.

	Spyware	Downloader	Trojan	Worms	Adware	Dropper	Virus	Backdoor
RF								
Precision	**0.49**	**0.77**	**0.43**	**0.66**	**0.92**	0.55	**0.87**	**0.62**
Recall	**0.46**	**0.74**	**0.40**	**0.66**	0.87	**0.71**	**0.77**	**0.66**
F1 score	**0.48**	**0.75**	**0.42**	**0.66**	**0.89**	**0.62**	**0.82**	**0.64**
LSTM (2 layers [30])								
Precision	0.23	0.64	0.25	0.39	0.61	0.27	0.56	0.35
Recall	0.19	0.61	0.14	0.20	0.75	0.41	0.68	0.46
F1 score	0.21	0.62	0.18	0.26	0.67	0.33	0.61	0.40
LSTM (Case 2 [28])								
Precision	0.38	0.71	0.30	0.45	0.77	**0.62**	0.72	0.55
Recall	0.32	0.72	0.38	0.52	**0.91**	0.51	0.76	0.51
F1 score	0.35	0.71	0.33	0.48	0.83	0.56	0.74	0.53
GRU (Case 2 [28])								
Precision	0.36	064	0.40	0.60	0.74	0.61	0.68	0.40
Recall	0.38	**0.74**	0.32	0.52	0.90	0.46	0.76	0.65
F1 score	0.37	069	0.36	0.56	0.81	0.53	0.72	0.50

9. Research Limitations

The training dataset must be sufficiently large in order to draw reliable findings from a study, such as malware family classification. How accurate the results are will depend on the sample size used. More meaningful relationships in the data can be found from the larger dataset. Therefore, the limited number of malware samples for each family is considered a limitation of this study. Additionally, due to processing resource limitations, the Jaccard similarity visualization was only possible for the first 500 samples.

10. Conclusions and Future Direction

This work utilized malware API calls as classification features to recognize different families of malicious software. The network analysis of shared API calls provided valuable information about the malware connectivity within the same family. This connectivity showed limited API call sharing among the samples of the same malware family, meaning the instances used different combinations of API calls. This can occur for many reasons, e.g., dataset labeling being prone to errors, attackers inserting API calls to hide the real purpose of malware, and the possibility of detecting the analysis environment (Sandbox) by advanced malware blocking the actual malicious behavior. This conclusion reduces the effectiveness of using API calls as time-series features. Time-series-based malware classification solutions use high-computational cost-neural network algorithms. API call frequencies, as features with RF classifiers, show effective and simple solutions for multiclass malware classification compared with existing complex methods.

Employing the proposed method for binary classification by adding legitimate samples would be an interesting future work direction. Another way to utilize the results of this work is by improving the visual analysis to comprise more dynamic and static attributes achieved from the new facilities supported by the VirusTotal service.

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 and 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.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

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

References

1. Institute, A.T. Malware Statistics and Trends Report: AV TEST. 2022. Available online: https://www.av-test.org/en/statistics/malware/ (accessed on 19 July 2022).
2. Al-Hashmi, A.A.; Ghaleb, F.A.; Al-Marghilani, A.; Yahya, A.E.; Ebad, S.A.; Saqib, M.; Darem, A.A. Deep-Ensemble and Multifaceted Behavioral Malware Variant Detection Model. *IEEE Access* **2022**, *10*, 42762–42777. [CrossRef]
3. Catak, F.O.; Yazı, A.F. A benchmark API call dataset for windows PE malware classification. *arXiv* **2019**, arXiv:1905.01999.
4. Oliveira, A.; Sassi, R. Behavioral malware detection using deep graph convolutional neural networks. *TechRxiv* 2019, *preprint*. [CrossRef]
5. VMRay. Sans Webcast Recap: Practical Malware Family Identification for Incident Responders. 2021. Available online: https://www.vmray.com/cyber-security-blog/practical-malware-family-identification-sans-webcast-recap (accessed on 10 July 2022).
6. Sebastián, M.; Rivera, R.; Kotzias, P.; Caballero, J. Avclass: A tool for massive malware labeling. In Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses, Paris, France, 19–21 September 2016 ; pp. 230–253.
7. Heer, J.; Bostock, M.; Ogievetsky, V. A tour through the visualization zoo. *Commun. ACM* **2010**, *53*, 59–67. [CrossRef]
8. Srivastava, V.; Sharma, R. Malware Discernment Using Machine Learning. In *Transforming Management with AI, Big-Data, and IoT*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 215–232.
9. Liu, X.; Du, X.; Lei, Q.; Liu, K. Multifamily classification of Android malware with a fuzzy strategy to resist polymorphic familial variants. *IEEE Access* **2020**, *8*, 156900–156914. [CrossRef]
10. Kakisim, A.G.; Nar, M.; Sogukpinar, I. Metamorphic malware identification using engine-specific patterns based on co-opcode graphs. *Comput. Stand. Interfaces* **2020**, *71*, 103443. [CrossRef]
11. Bayazit, E.C.; Sahingoz, O.K.; Dogan, B. A Deep Learning Based Android Malware Detection System with Static Analysis. In Proceedings of the 2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), Ankara, Turkey, 9–11 June 2022; pp. 1–6.
12. Liu, X.; Lin, Y.; Li, H.; Zhang, J. A novel method for malware detection on ML-based visualization technique. *Comput. Secur.* **2020**, *89*, 101682. [CrossRef]
13. Cui, Z.; Xue, F.; Cai, X.; Cao, Y.; Wang, G.g.; Chen, J. Detection of malicious code variants based on deep learning. *IEEE Trans. Ind. Inform.* **2018**, *14*, 3187–3196. [CrossRef]
14. Zhang, J.; Qin, Z.; Yin, H.; Ou, L.; Zhang, K. A feature-hybrid malware variants detection using CNN based opcode embedding and BPNN based API embedding. *Comput. Secur.* **2019**, *84*, 376–392. [CrossRef]
15. Qiang, W.; Yang, L.; Jin, H. Efficient and Robust Malware Detection Based on Control Flow Traces Using Deep Neural Networks. *Comput. Secur.* **2022**, *122*, 102871. [CrossRef]
16. Palša, J.; Ádám, N.; Hurtuk, J.; Chovancová, E.; Madoš, B.; Chovanec, M.; Kocan, S. MLMD—A Malware-Detecting Antivirus Tool Based on the XGBoost Machine Learning Algorithm. *Appl. Sci.* **2022**, *12*, 6672. [CrossRef]
17. Usman, N.; Usman, S.; Khan, F.; Jan, M.A.; Sajid, A.; Alazab, M.; Watters, P. Intelligent dynamic malware detection using machine learning in IP reputation for forensics data analytics. *Future Gener. Comput. Syst.* **2021**, *118*, 124–141. [CrossRef]
18. Bahtiyar, Ş.; Yaman, M.B.; Altınığne, C.Y. A multi-dimensional machine learning approach to predict advanced malware. *Comput. Netw.* **2019**, *160*, 118–129. [CrossRef]
19. Han, W.; Xue, J.; Wang, Y.; Huang, L.; Kong, Z.; Mao, L. MalDAE: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. *Comput. Secur.* **2019**, *83*, 208–233. [CrossRef]
20. Xiaofeng, L.; Xiao, Z.; Fangshuo, J.; Shengwei, Y.; Jing, S. ASSCA: API based sequence and statistics features combined malware detection architecture. *Procedia Comput. Sci.* **2018**, *129*, 248–256. [CrossRef]
21. Rhode, M.; Burnap, P.; Jones, K. Early-stage malware prediction using recurrent neural networks. *Comput. Secur.* **2018**, *77*, 578–594. [CrossRef]
22. 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.
23. Lu, F.; Cai, Z.; Lin, Z.; Bao, Y.; Tang, M. Research on the Construction of Malware Variant Datasets and Their Detection Method. *Appl. Sci.* **2022**, *12*, 7546. [CrossRef]
24. Rosenberg, I.; Shabtai, A.; Rokach, L.; Elovici, Y. Generic black-box end-to-end attack against state of the art API call based malware classifiers. In Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses, Crete, Greece, 10–12 September 2018; pp. 490–510.
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. 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.
27. 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, Australia, 16–18 July 2013; pp. 265–270.
28. Li, C.; Zheng, J. API call-based malware classification using recurrent neural networks. *J. Cyber Secur. Mobil.* **2021**, *10*, 617–640. [CrossRef]
29. Tang, M.; Qian, Q. Dynamic API call sequence visualisation for malware classification. *IET Inf. Secur.* **2019**, *13*, 367–377. [CrossRef]
30. 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]
31. 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 Proceedings of the 8th International Conference on Artificial Intelligence and Applications (AIAP 2021), EL-Oued, Algeria, 28–30 September 2021.
32. Rogel-Salazar, J. *Data Science and Analytics with Python*; Chapman and Hall/CRC: London, UK, 2018.
33. Networkx. NetworkX Network Analysis in Python. 2022. Available online: https://networkx.org/ (accessed on 20 July 2022).
34. Graphviz. What Is Graphviz? 2022. Available online: https://graphviz.org/ (accessed on 20 July 2022).