Received 15 July 2023, accepted 10 September 2023, date of publication 25 September 2023, date of current version 29 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3319093

RESEARCH ARTICLE

A Malware Detection Approach Based on Feature Engineering and Behavior Analysis

MANUEL TORRES, RAFAEL ÁLVAREZ, AND MIGUEL CAZORLA, (Senior Member, IEEE)
Department of Computer Science and AI, University of Alicante, 03690 Alicante, Spain

Corresponding author: Manuel Torres (mtm41@alu.ua.es)

This work was supported by the ''Methodology for EmotionAware Education Based on Artificial Intelligence'' (Program PROMETEO 2022—CIPROM/2021/017, Conselleria de Innovación, Universidades, Ciencia y Sociedad Digital de la Generalitat Valenciana, Spain).

ABSTRACT Cybercriminals are constantly developing new techniques to circumvent the security measures implemented by experts and researchers, so malware is able to evolve very rapidly. In addition, detecting malware across multiple systems is a challenging problem because each computing environment has its own unique characteristics. Traditional techniques, such as signature-based malware detection, have become less effective and have largely been replaced by more modern approaches, including machine learning and robust cross-platform behavior-based threat detection. Researchers employ these techniques across a variety of data sources, including network traffic, binaries, and behavioral data, to extract relevant features and feed them to models for accurate prediction. The aim of this research is to provide a novel dataset comprised of a substantial number of high-quality samples based on software behavior. Due to the lack of a standard representational format for malware behavior in current research, we also present an innovative method for representing malware behavior by converting API calls into 2D images, which builds on previous work. Additionally, we propose and describe the implementation of a new machine learning model based on binary classification (malware or benign software) using the previously mentioned novel dataset as its data source, thereby establishing an evaluation baseline. We have conducted extensive experimentation, validating the proposed model with both our novel dataset and real-world data. In terms of metrics, our proposed model outperforms a well-known model that is also based on behavior analysis and has a similar architecture.

INDEX TERMS Convolutional neural networks, dataset, machine learning, malware.

I. INTRODUCTION

Malware-related security breaches are one of the main sources of business economic losses associated with IT, coexisting with others that were more common a few years ago, such as human error or industrial espionage. In 2019, 90% of the 5,500 companies interviewed by Kaspersky [1] reported losses of more than $500,000 due to malware-related security incidents. These losses can result from a loss of access to key business information that threatens the reputation of the company almost irreparably and prevents normal logistical and business operations.

In fact, the number of organizations compromised by at least one successful attack has increased by 25% since 2014 (see [2]), leading us to the conclusion that the malware problem is growing. The variability of malware behavior is a significant problem, as malware can rapidly mutate, incorporating elements of other malware or employing innovative strategies that the security community has not yet identified. Furthermore, Alazab et al. [3] have identified the use of code obfuscation, primarily metamorphic and polymorphic techniques (with or without encryption), as one of the greatest challenges in malware detection, effectively rendering signature-based malware detection ineffective. According to researchers such as Yuan et al. [4], antivirus software employing this type of detection has a low probability of correctly identifying obfuscated malware as malicious software, between 25% and 50%.

Software development evolves over time, utilizing new patterns, frameworks, and tools that result in more diverse

The associate editor coordinating the review of this manuscript and approving it for publication was Yassine Maleh.

and difficult-to-classify behaviors, thereby enabling more sophisticated malware that can exploit this variability. Simple systems that only check for certain features must be replaced with more intelligent methods that can provide a higher level of accuracy when determining whether or not software is actually harmful. In fact, given the growing success of machine learning (ML) methods applied to a wide range of tasks and their unique characteristics, automated malware detection presents an excellent opportunity; consequently, we will focus on the application of ML-based methods to malware detection in this paper.

There are several approaches within the scope of ML-based malware detection:

- Network traffic analysis. Numerous publications discuss classification techniques for network traffic; the most pertinent methods include traffic matching with static rules and pattern recognition based on particular characteristics [5]. The latter is more flexible when it comes to resolving concerns associated with the usage of sophisticated evasion countermeasures that are not reliant on predefined-rule matching; however, other issues remain, such as the use of encryption or selecting the optimal features for best results. In addition, modern malware may not generate traffic for extended periods of time until the required information is available; in fact, this information is typically exfiltrated and concealed within common protocols, such as DNS payloads or specific HTTPS request headers. As published by Asaf Nadler et al. [6], there is research that aims to detect data exfiltration by searching for specific characteristics within packet payloads or performing entropy analysis. Unfortunately, these methods are less useful in general malware detection environments and are therefore outside the scope of this work.
- Binary analysis. The majority of these techniques concentrate on extracting executable characteristics through static analysis, taking into consideration PE header text, entropy, OPCode sequence, and imported DLLs in order to construct feature vectors and establish the type of program [7]. One of the biggest problems with this strategy is that malware creators employ countermeasures to hide the real binary code. Usually, they do this by using obfuscation techniques to hide the actual machine code in an executable or DLL, either completely or partially.
- Behavior analysis. In the vast majority of published studies, the most crucial aspect is capturing the sequence of API calls executed by the analyzed program. In contrast, other aspects, such as operating system or network events, are generally considered to be of less significance [8]. Dynamic analysis is required to collect this information and is performed using third-party software running concurrently with the malware on the same system in order to inject itself into the suspicious process and extract the relevant data. This is difficult to accomplish due to the fact that most malware today has anti-debugging and anti-virtual machine (anti-VM) mechanisms.

It should be noted that, despite the fact that malware traffic and binary code analysis provide very interesting and certainly significant information for malware classification, there are too many drawbacks with these methods, in addition to the fact that extracting this information may require dynamic malware analysis. Therefore, the API call data obtained during execution gives us a clear picture of the software's true purpose, whether it is editing a registry key to achieve persistence (RegSetValue), connecting to an external domain to download malware (ConnectEx), or encrypting key files (CryptEncrypt). By analyzing these calls, we are able to identify behavioral patterns and determine whether or not the software in question is malicious.

Different ML techniques, such as deep learning based on decision trees, graphs, support vector machines (SVM), and even convolutional neural networks (CNN), are used within these categories.

The main goal of this study is to create a novel dataset comprised of high-quality samples, providing valuable resources for future malware detection research. Additionally, we propose an innovative method for representing malware behavior, specifically designed to serve as input for a Convolutional Neural Network (CNN).

Our hypothesis is that leveraging this novel representational format will result in superior malware detection when compared to current methods.

Consequently, our research has been primarily motivated by the need to answer certain questions whose solutions involve creating a high-quality dataset devoid of insufficient samples, utilizing the features extracted from the API calls as software behavior patterns, representing these features as 2D images, analyzing the performance of a CNN model to find behavior patterns in 2D images, and determining the optimal model architecture to maximize accuracy.

This work builds upon and extends our previous research on malware detection (see [9]). In this regard, the dataset has been significantly improved, and a novel feature engineering technique has been implemented. In addition to the aforementioned dataset, a new CNN machine learning model has been introduced, and a thorough comparison of results with existing models has been conducted as well.

To conclude this section, we would like to highlight the primary motivation for our research, which is to improve malware detection using machine learning. Our work encompasses several contributions, including the development of a novel dataset containing behavior analysis for thousands of both malware and benign software samples. This dataset contains a variety of elements, including API calls, arguments, and processes, among others. In addition, we present a novel Feature Engineering method to convert API call data from sandbox reports into 2D images. The primary characteristics extracted from each call are its name, category, arguments, and frequency. These images serve as input for a CNN model that we have implemented to improve the accuracy of malware detection. This model will automatically extract features from the input images. By combining these approaches, we aim to advance the field of malware detection and contribute to the overall security of computer systems.

The remainder of the paper is structured as follows. In Section II, we examine existing research that employs ML techniques to detect malware behavior. Next, in Section III, we propose a novel dataset and a CNN model with the potential to accurately classify samples as malware or benign software. In Section IV, we detail our experiments and compare the results obtained with our proposed dataset to those of previous publications. Finally, in Section V, we provide some conclusions and recommendations for future research.

II. RELATED WORK

In this section, we examine recent publications in the field of behavior-based malware detection, focusing primarily on the use of operating system API calls.

In 2010, Trinius et al. [10] published a novel method for representing software API calls as a meta-language known as MIST (Malware Instruction Set). They intended to use this method to establish a standard format for the representation of malware analysis, as a large number of datasets related to malware detection were emerging at the time. This is particularly compelling because it has the potential to significantly improve research and collaboration in this field by providing a way to represent malware that is sufficiently abstract to describe both current and future malware.

This meta-language was initially constructed using a hexadecimal trace with twenty different categories and four different levels of priority (operation category, affected instance, instance specification, and arguments). Fig. 1 depicts a software sample executing a successful system call with the intention of loading the NTDLL library, as well as other descriptive data, including library size and memory addresses.

In order to convert between sandbox-provided behavior reports and MIST, features such as category, operation, file size, memory address, or any other involving a finite number of possibilities are directly converted to hexadecimal. For instance, categories are encoded using two hexadecimal digits, allowing for up to 256 categories. Other attributes that cannot be encoded directly (paths, etc.) are converted to a fixed-length value with a hash function.

As stated previously, dynamic analysis data supplied by a third party is required to represent the behavior of malware. In this instance, the CWSandbox platform was employed following the common approach for software behavior analysis, as depicted in Fig. 2.

Beginning with the work of Trinius et al. [10], other researchers have developed their own representation methods. In 2015, Fan et al. [11] used a collection of API calls containing specific DLLs (that is, user32, kernel32, advapi, ntdll, ws2, and wininet), obtained 1024 samples (773 malware and 251 benign) and developed an ML model based on a naive Bayes classifier, a decision tree (J48) and SVM architectures that managed to achieve an accuracy level of 95.3%. In this instance, a representation format based on feature engineering was selected, in which the frequency of calls to each API call was included in addition to the call name.

In these studies, a custom representational format was utilized to capture the behavior of their respective samples. However, Trinius et al. adopted an architecture with a primary level encompassing category and operation for the system call, followed by multiple subsequent levels representing each argument block. On the other hand, Chun-I Fan et al. employed a Feature Engineering method based solely on API call names and their frequencies.

In contrast to these approaches, our proposed method utilizes a tree-level format that includes the category and API call name consistently. However, the inclusion of API call arguments or additional relevant data in subsequent levels is dependent on the operation in question. This distinction is essential as certain data within this field, such as random file names or memory addresses, could introduce noise into our dataset. A detailed explanation of our proposed method can be found in Section III-A.

It should be noted that employing only a few features may not guarantee a complete capture of malware behavior, as there are many more relevant DLLs that are not being analyzed. Baldangombo et al. [12] collected a total of 236756 malware programs, in which the shell32, wsock32, oleaut32, and msvbvm50 DLLs had a higher frequency than some of those selected by Chun-I Fan et al. In addition, it is possible to perform the same action through multiple DLLs. For instance, shell32 can run another DLL using the Control_RunDLL API call and kernel32's LoadLibraryA API call.

In 2018, Masabo et al. [13] utilized a dataset supplied by Marco Amilli with a MIST-based format, although one that was considerably more direct and focused on rapid generation. It employs 22 features that are not related to signatures, and the values assigned to them are the first eight characters of the MD5 hash for each value of the feature (that is, the name of the function used in an API call, which could be LoadLibraryA). In a dataset containing 2957 samples, they were able to classify two types of malware (Crypto and Zeus) with an accuracy of 97%. However, since they are using the partial result of a broken hash algorithm, such as MD5, this method could lead to a collision issue.

In contrast to other studies, Rabadi and Teo [14] examine the significance of API call arguments as opposed to only assessing the name and frequency of API calls. They develop two ways to represent software features through feature engineering: the first technique treats each API call and its arguments as a single feature, while the second method handles them as individual features. These features are converted into a binary vector and hashed with the MurmurHash3 algorithm before being fed into an XGBoost model, which performs at 96.7% accuracy on a test dataset with 2972 samples (divided into 1418 malicious and 1554 benign software

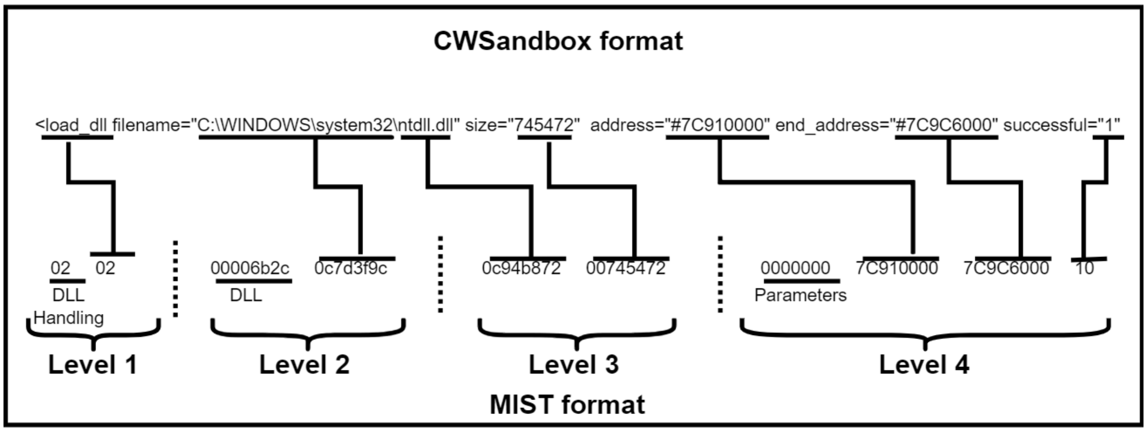
CWSandbox format
<load_dll filename="C:\WINDOWS\system32\ntdll.dll" size="745472" address="#7C910000" end_address="#7C9C6000" successful="1"
02 02
DLL Handling
00006b2c 0c7d3f9c
DLL
0c94b872 00745472
0000000 7C910000 7C9C6000 10
Parameters
Level 1
Level 2
Level 3
Level 4
MIST format

FIGURE 1. Format conversion from CWSandbox to MIST.

samples). Consequently, this strategy suggests that it may be a good idea to make features less specific or, in other words, to eliminate additional parameters that do not alter the purpose of the call and hinder the identification of similar API calls and behavior patterns.

As shown in the study by Rabadi anf Teo, we acknowledge the potential value of arguments in determining software behavior and true intentions in our proposed method. However, it is important to note that the extraction of arguments should be performed carefully to ensure that all relevant information is included. Furthermore, whereas Dima Rabadi et al. and other authors use traditional machine learning models, we have chosen a CNN architecture in which the model selects features from the provided images. This method enables automatic feature extraction, improving the model's ability to learn and capture intricate patterns.

In 2020, Ficco [15] employed a detection approach using a combination of five detectors to identify malware families in the Android operating system. This paper discusses various approaches, such as API call frequency and network traffic analysis, but these features depend on information that can change over time due to updates or obfuscation, particularly in the case of network traffic and memory analysis. Besides, this method does not consider the use of an abstract representational method that can reveal software objectives during system interactions, not taking advantage of the insights gained from MIST [10]. Furthermore, the author does not utilize important information about each API call, such as the arguments that can significantly alter the meaning of certain calls. In our proposed method, described in Section III, we introduce a feature engineering technique that combines category, name, and relevant arguments (if applicable) for each call. These components are encoded into separate channels of an RGB image, which is then analyzed by a CNN model. Lastly, it is important to note that our proposal focuses on Windows-based samples rather than Android samples, as we believe this represents a more diverse environment.

As shown in Fig. 2, the research trends within the field of behavior-based malware detection follow a pattern centered on the collection of API calls executed by malicious and benign samples using third-party tools that are typically sandboxes such as CWSandbox or Cuckoo Sandbox. These tools generate reports for each sample from which a sequence of API calls and other useful parameters such as arguments, frequencies, and timestamps can be extracted. These can be further processed in order to apply feature engineering techniques to compress relevant information, remove irrelevant features, and make the format compatible with any sample representation. When the final features of the samples are obtained, an ML-based model is typically applied as a classifier, which can then be used to identify specific classes of malware (clustering) or simply to differentiate malicious from benign software.

There appears to be a severe lack of high-quality datasets in the field of malware, which is another existing issue. On the other hand, as Gamage et al. note in their survey [16], there are a few datasets that are still used by the scientific community despite being outdated, such as the well-known KD99 and NLS-KDD datasets. There are datasets about malware-generated traffic, binaries, and static features, but it is hard to find high-quality datasets that capture malware behavior and include API calls. The rapid evolution of malware makes it difficult to compile standard datasets. This can lead researchers to generate implicitly biased datasets that are coupled to their own unique representation system.

Our proposed dataset consists of current samples analyzed by the security community in a publicly accessible online sandbox. The selection process for including these samples in our dataset is not subjective, other than the exclusion of non-adequate samples. Despite the fact that we convert API call traces into 2D images, it is important to note that the dataset contains the full report, which includes signatures, static analysis results, network events, etc.

III. PROPOSAL

This section is divided into two subsections. The first one describes the methodology used to generate a novel dataset from API calls made by both benign and malicious samples: we describe the collection phase, highlighting the checks that must be performed to generate the dataset, as well as the process of converting the base reports into 2D images, and conclude by comparing our dataset to others previously published in other papers. In the second section, we describe the technical details of the proposed machine learning model that uses our dataset.

A. PROPOSED DATASET

We can assume that if we can capture the behavior of a software sample in an easily understandable format and also enable pattern recognition through the use of deep learning models, we will be able to successfully classify malicious and benign software or, at the very least, establish groups of similar behavior. The following guidelines should be followed when creating a format:

- In order for it to recognize patterns, it must be classified into various types of actions that pursue the same end goal. Due to the variable nature of the arguments, the frequent use of obfuscation, and the vast number of distinct but equivalent ways in which the same procedure can be executed in an operating system, it would be nearly impossible to repeat individual actions.
- These categories need to be sufficiently robust and abstract to be unaffected by minor differences in malware behavior, thus preventing deception or evading detection.

Zhang et al. develop a dataset and a deep learning model that is capable of classifying software samples as malicious or benign (see [17]). The dataset is formatted in JSON and is based on basic Cuckoo Sandbox reports that represent the behavior of the sample regardless of its nature (executable, DLL, driver, etc.). These software samples are executed within a virtual machine containing an agent that logs and transmits action-related data to a remote host. Finally, for each execution of the software sample, these actions are reflected in a single report, and the virtual machine is reset to its initial state, allowing another suspicious piece of software to be executed independently of previous executions. This behavior is depicted in Fig. 2.

Initially, we attempted to use the same methodology as Zhang et al. [17], that is, generating reports on the behavior of malware uploaded by us to a private Cuckoo Sandbox environment. However, we decided that it would be more appropriate to obtain these reports from a Cuckoo Sandbox online environment (https://cuckoo.cert.ee/), a public platform where anyone can upload software samples to be analyzed. In this way, the samples used in the dataset will not be subjective, although they will obviously be more focused on suspicious software given that users who upload software to this platform will typically be suspicious of its intent. This platform has a higher number of reports and more variation than the Zhaoqui et al. dataset. It also has samples that are based on Linux.

The reports stored on this platform were downloaded using a simple script; however, it was unclear whether the downloaded report corresponded to a malicious or innocuous software sample; as a result, a second script was written to check the rating given by Cuckoo Sandbox (on a scale of 0 to 10) and determine the nature of the software based on the following criteria:

- Benign software (with a rating of 0 to 3). This software is not malware.
- Uncertain software (with a rating of 3 to 7). It is unclear whether this software is malware or not.
- Malicious software (with a rating of 7 to 10). It is certain that this software is malware.

To confirm the malicious nature of each sample, its signature was also validated against the databases of multiple antivirus companies. In order to ensure that the dataset contains as little noise as possible and that the samples used are correctly classified, uncertain samples were omitted from the final dataset and only benign and malicious software samples were considered. In addition, samples that contained very little information were removed, either because they could not be executed successfully due to a lack of dependencies or for other reasons outside the scope of this research.

It was possible to obtain a dataset[1] containing 45236 samples (malware and benign) with an average duration of 6.4 minutes per sample on both Windows 7 and Linux. Regarding the format of the report, it contains very relevant information regarding the execution of the software sample and its identification:

- The *info* and *target* sections contain fundamental information such as the start and end timestamps as well as the operating system on which it was executed, together with the type of software (i.e., exe, dll, etc.). In addition, the *network* section provides a summary of the connection attempts.
- The static analysis of the software sample is represented by the *metadata*, *strings*, and *signatures* sections. This involves comparing hashes and extracted strings from the binary code with antivirus databases.
- The *behavior* section describes each action taken by the processes in the sample. API calls are grouped by the scope of the operation (file, crypto, network, etc.).

We performed a series of transformations on the Cuckoo Sandbox source reports to convert them into 2D images in order to obtain the benefits of CNN models for our dataset and establish a standard method of representation. As shown in Fig. 3, after obtaining the reports, we determined whether the file is malicious or benign, as well as which operating system it was executed on. Nonetheless, a detailed procedure is required to evaluate the sample quality:

[1]This dataset is freely available from the following repository: https://github.com/mtm41/dataset

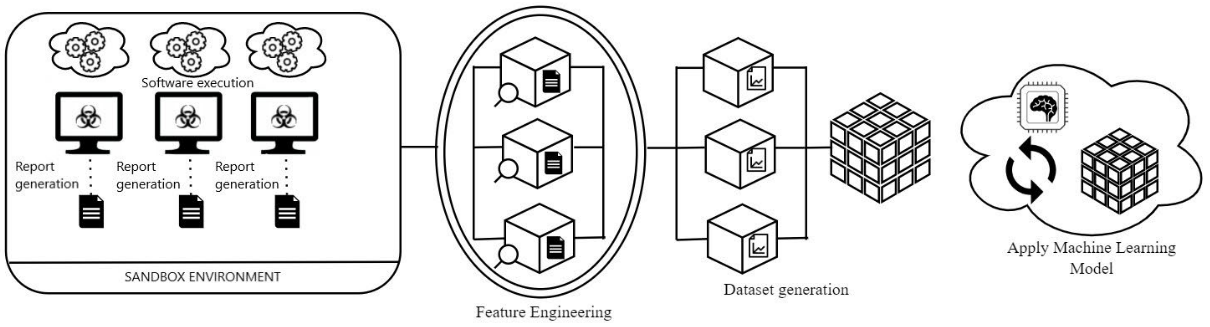

FIGURE 2. Behavior-based malware detection workflow.

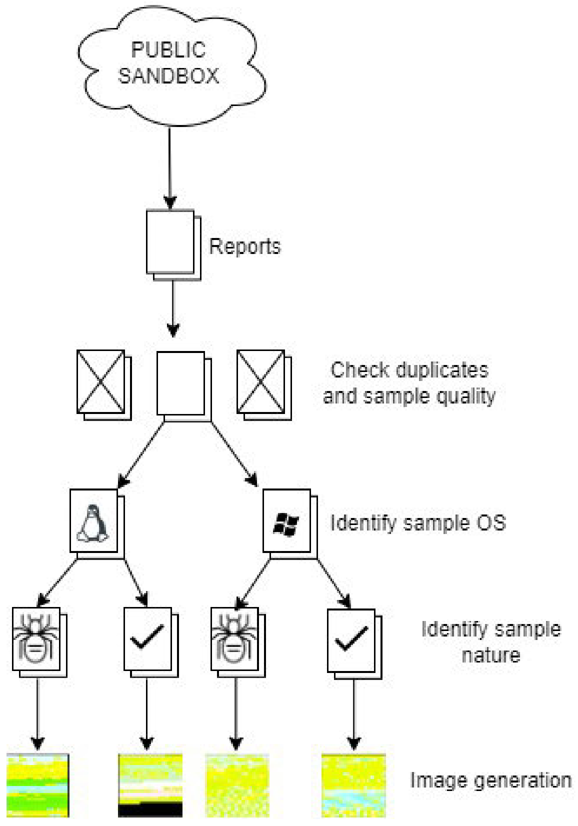

FIGURE 3. Dataset generation.

- Verify that there are no duplicate reports on the dataset; more specifically, that we do not have the same report with the same API calls multiple times in the entire dataset, which is distinct from having multiple reports for the same sample with different API calls. The latter is permitted in our dataset because it may provide alternative execution paths for the same piece of software, a factor that must be taken into account when performing dynamic analysis based on API calls.
- Control the number of reports with a low number of API calls. That is, for instance, in the case of malware samples that were not able to run malicious API calls, the hash of the sample will probably raise a positive when compared with the malware database, as well as the Cuckoo Sandbox score, since a part of it relies on the signature. Therefore, we would have an inconsistency since the API calls captured in that sample do not represent malicious behavior, and thus that sample cannot be categorized as malware. On the other hand, we need some benign samples with fewer API calls because our model should be able to recognize them in future predictions, but these samples must be correctly labeled as benign to begin with.

Finally, reports have to be transformed into 2D images while attempting to introduce as much automation, customization, and abstraction as possible. The RGB representation was chosen as the image format because it has three 8-bit channels in which we can add data, allowing us to represent more behaviors if needed. Once the boundaries are clearly defined, we divide these 24 bits into three parts:

- Category of the operation. This section consists of 8 bits, and most of the categories defined by Cuckoo Sandbox in their reports are included because they are sufficiently abstract for our needs (i.e., *File* for an operation that involves writing to or reading from the file system, *Registry* for operations involving registry keys, etc.). It should be noted that Cuckoo Sandbox only has 14 categories, but they are equally distributed among the possible 2^8 different combinations available, avoiding any possible color imbalance and allowing for future additions since the category of the operation is the most abstract data available and could be key to extracting a behavioral pattern from malware API calls.
- API call. This section has another 8 bits assigned, allowing for the representation of 2^8 API calls in a single category. A global object is employed per each category identified in our dataset through a reflection process, and thus each API call is handled in an isolated method. Other relevant information, such as frequency, is also stored in the global category object. Even though API call names are represented using the entire channel, a higher number is assigned to those that are utilized more frequently by malware (values close to 255). This is consistent with the notion that a darker image (with lower values in each channel) corresponds to a benign

sample, while brighter hues may indicate a suspicious one.

- Arguments used for the API call. This section covers an additional 8 bits, so a total of 2^8 arguments per API call in each category can be represented. Although arguments are a significant source of information regarding intent, they are not only extremely heterogeneous, but their presence or absence can significantly alter the meaning of an API call, making them difficult to represent. In addition, we must consider the fact that some arguments do not contribute any useful information to our dataset but rather add noise. In light of this, it must be determined for each set of API calls if its arguments or another feature should be considered. For instance, in the case of API calls that handle registry keys or files, the name of the involved key, directory, or file type is encoded directly, and a hash function is employed if the information is relevant but too large to be encoded. In this third channel, the frequency of each API call is also encoded.

Each API call is represented by a single RGB pixel, so the image dimensions are based on the existing median number of API calls in our data set. For the current iteration, we have chosen to employ 32×32 pixel images, which corresponds to a median of 1024 API calls. This decision is crucial, as models require the same dimension for each sample and images must be large enough to efficiently encode all necessary data.

As can be seen, the method of converting API call data from Cuckoo Sandbox to a 2D image is comparable to MIST, as our method was heavily influenced by it. We retain the category of the executed API call and use the API call as the operation, but we eliminate as much dynamic data as possible, such as file size or memory addresses. However, file paths and file names are still used, but we try to make them as abstract as possible. For instance, we don't use the file name in the *arguments* section because it could vary greatly depending on the implementation, and they are typically also hashed; instead, we take the file type or extension.

From our perspective, these features may not be a valid option in many instances, and we should allow the ML model to determine the relevant features for classifying samples into one or more categories. When discussing Windows systems, it makes sense to let the ML model decide, as it is possible for malware samples to frequently use API calls related to registry keys or encryption. Due to the diversity of this environment, this pattern is not particularly precise, as many benign samples could be identified as false positives.

Furthermore, development trends change on a regular basis, making new ways of performing operations available through the use of new patterns or frameworks; this results in a situation in which features can hardly be defined in advance. However, other features, such as the permissions of the user running the process, may be suitable candidates for the *argument* section of our implementation.

It should be noted that the rapid technological advances in computing platforms and malware attack strategies may necessitate constant model retraining to maintain classification accuracy, which could lead to an untenable situation. Our work attempts to effectively address this significant challenge by introducing a representation format that emphasizes abstract behavioral characteristics rather than platform-specific API calls or operating system functions that are likely to be subject to future modification. This representation format aims to create a long-term, robust, and sustainable model.

In Table 1, we can view a comparison between our proposed dataset and previously published datasets. First, it is important to note that the research community does not have access to a large number of datasets that focus on API calls. Moreover, many of them are based on a specific format and some of them already include feature engineering methods, as seen in the Zhang et al. dataset [17]; therefore, the experimentation cannot be replicated in its entirety because unprocessed source samples are not provided.

Similarly, Y. Liu et al. employ a natural language processing algorithm called *word2vec* in their dataset [18], a strategy closely related to the objective pursued by feature engineering algorithms and resulting in the same issue. However, there are datasets that provide the original API call name without obfuscation, such as the dataset proposed by Catak et al. [19], but the lack of other relevant information about the API call, such as the arguments, lowers the overall data quality. We can also say that datasets collected more than ten years ago are likely to be less useful because malware evolves rapidly, and API calls used by malware more than a decade ago might not be used today, especially if these datasets only consider API call names and not their true purpose.

Our dataset includes a collection of software execution reports containing a significant number of malware and benign samples, as well as the entire sampling workflow, from the source report generated by the sandbox to 2D image conversion considering multiple API call indicators in addition to the function name. Along with Windows samples, this dataset also includes 29,175 Linux samples. This allows for additional experimentation on both operating systems, improving robustness. The distribution of different malware sample types is detailed in Table 2.

B. PROPOSED MODEL

Before defining the final CNN model, we experimented with other well-known architectures, such as Inception V3 [20] and Resnet [21], both of which produced results that could be easily improved upon and were inferior to our proposed architecture. As explained in Section III-A, it should be noted that our images typically have small dimensions because they are calculated based on the number of API calls in our dataset. Consequently, the increased complexity of these architectures may have a negative impact on the results. In addition, Inception v3 has a minimum image size of 75×75, which is larger than the size of the images we use. As suggested by Luke et al. [22], we must resize the dataset to this minimum size before we can run the model, which may result in a loss of

TABLE 1. Datasets distribution.

Dataset	Malware samples	Benign samples	Total	Year
F. Hamed et. al. Dataset	416	100	516	2009
Malicious Datasets CsMining Group	320	68	388	2014
Y. Liu et. al Dataset	13518	7860	21408	2019
Z. Zhang et. al. Dataset	27074	30712	57786	2020
Catak et. al. Dataset	7107	0	7107	2021
Proposed Dataset	25936	19300	45236	2022

TABLE 2. Sample type distribution.

Type	Samples
Trojan	25032
Adware	367
Backdoor	6309
DDoS	3814
Exploit	129
HackTool	83
Ransomware	2223
Potentially Unwanted App. (PUA)	405
Password Stealer	377
Virus	2693
Worm	54
Unclassified	3687

feature quality and accuracy. In their experiments, they used architectures such as Inception and Resnet.

In the case of Inception v3, we added a global spatial average pooling layer to prevent over fitting, and a softmax activation function is also used. For the optimizer, we defined a learning rate of 0.0001 with a momentum of 0.9. On the other hand, we used the 34-layer version of the Resnet architecture with the same optimizations as in Inception. As shown in Table 3, Inception and Resnet achieve worse results than our proposed model; however, the accuracy is still greater than 90% despite the fact that loss does not decrease as epochs pass, rendering the model more unstable.

As can be seen in Fig. 4, the chosen architecture employs a straightforward strategy in which six convolutional 2D layers are applied with 32, 64, 128, 256, 512, and 1024 filters, each with a dimension of 2×2. Furthermore, a *ReLU* activation function and a *he_uniform* layer weight initializer are used. Two *MaxPooling2D* functions with a pool size of (2, 2) are applied between the first and sixth layers to downscale the output of the layers. A *Flatten* function is then used to obtain features, followed by a *Dropout* layer with a 0.6 rate. Finally, a fully connected network of 4096 nodes is employed, the activation function for this layer is also ReLU, although a Sigmoid is used in the last Dense layer with the aim of reducing the output to two nodes. A gradient descent optimizer with a learning rate of 0.0001 and momentum of 0.9 is employed to compile the model.

This proposed architecture does not have an excessive number of layers and maintains simplicity as its primary characteristic, which is closely related to the concept of shallow networks. In this instance, the use of a shallow network with low complexity makes the model faster in terms of the inference time required for classification, which is extremely important in real-time environments where malware detection must be as efficient as possible. On the other hand, shallow networks have been shown to perform better than more complex architectures when low-resolution images are used (see [23]).

IV. RESULTS

In this section, the performance of our proposed model is compared to that of the model proposed by Zhan et al. [17], which exhibits the best performance among those examined. The following metrics are used to measure the performance of the evaluated models:

- A receiver operating characteristic (ROC) curve depicts the diagnostic capabilities of a binary classifier system relative to the discrimination threshold, analyzing the relationship between the true positive rate (TPR) and the false positive rate (FPR) at a variety of thresholds.
- The area under the curve (AUC) is a measure of a classifier's ability to distinguish between classes, serving as a summary of the ROC curve and corresponding to its area. The greater the AUC, the better the model can distinguish between positive and negative classes.
- Accuracy (ACC) is a straightforward validation metric that measures the proportion of correct classifications. It can be defined as

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}}$$

  where TP (true positives) represents the number of malware samples correctly classified, TN (true negatives) represents the number of benign samples correctly classified, FP (false positives) represents the number of benign samples misclassified as malware, and FN (false negatives) represents the number of malware samples misclassified as benign.

Training time for our detectors is dependent on the configuration of the hardware employed. Our experiments were performed on a computer with an AMD Ryzen 5 3600 processor (3.6 GHz, 6 cores, 12 threads) and a Nvidia GeForce RTX 2060 graphics card (6 GB of GDDR6 RAM) capable of 6.451 TFLOPS in FP32 (float) precision. Total training time, including the 4-K-Fold validation procedure, was measured at 386 minutes.

First, the model provided by Zhang et al. was trained using the training section of their dataset, resulting in a unique H5 file for each of the four K-folds generated during the validation phase. Each H5 file contains a model that has been evaluated using distinct data. This is a crucial aspect since we

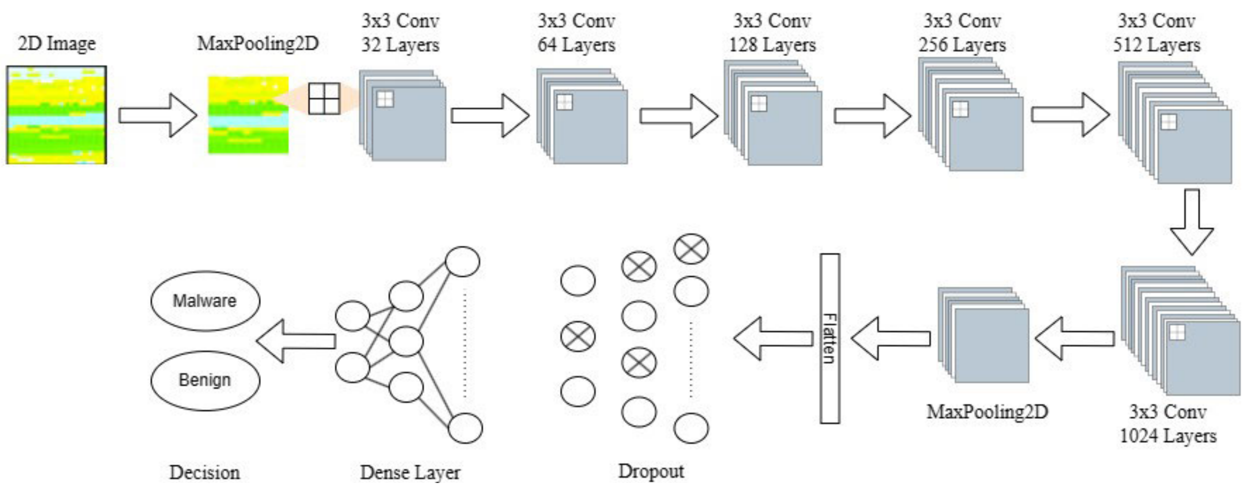

FIGURE 4. Model architecture representation.

TABLE 3. Model performance comparison.

Training

	Accuracy (%)	Loss	Precision (%)	Recall (%)
InceptionV3	95.80 ± 1.86	0.084 ± 015	95.80 ± 1.86	90.96 ± 1.86
Resnet34	94.23 ± 14.26	0.044 ± 0.025	93.94 ± 2.57	91.15 ± 2.28
Proposed model	96.53 ± 0.41	0.099 ± 0.110	96.55 ± 0.45	96.55 ± 0.37
Zhaoqui Zhang et al.	64.30 ± 0.04	0.650 ± 0.001	51.57 ± 0.12	50.00 ± 0.13

Validation

	Accuracy (%)	Loss	Precision (%)	Recall (%)
InceptionV3	92.78 ± 8.09	0.28 ± 0.317	88.46 ± 17.59	80.86 ± 16.87
Resnet34	92.37 ± 12.63	0.3385 ± 0.465	92.10 ± 19.51	84.10 ± 16.62
Proposed model	95.64 ± 1.23	0.146 ± 0.457	95.63 ± 1.15	95.65 ± 1.29
Zhaoqui Zhang et al.	64.41 ± 0.08	0.650 ± 0.001	52.33 ± 0.22	50.32 ± 0.19

Testing

	Accuracy (%)	Loss	Precision (%)	Recall (%)
InceptionV3	90.42 ± 4.39	0.422 ± 0.238	91.79 ± 5.59	81.91 ± 4.79
Resnet34	88.24 ± 5.99	0.6 ± 0.55	92.13 ± 2.05	74.70 ± 5.2
Proposed model	92.10 ± 1.29	0.242 ± 0.290	92.99 ± 3.20	93.21 ± 1.46
Zhaoqui Zhang et al.	47.36 ± 5.23	0.750 ± 0.620	53.12 ± 0.35	50.42 ± 3.46

must utilize each created model and the selected test dataset to determine an acceptable deviation value. The lower the precision variance between K-folds, the more accurate the model will be.

Fig. 5 displays that the model achieves statistics similar to those reported by Zhang et al. [17], with a very low FPR and a high AUC value. Both results are relevant when compared, since the AUC indicates that the model will assign a greater maliciousness score to a malware sample than a benign sample with a chance of 98%, corresponding to a low false positive rate (see [24]).

On the other hand, we observe a recall value that is not particularly high, indicating that there is still room for improvement in the model's ability to identify malware samples. As shown in Table 1, the accuracy of their model is enhanced by the inclusion of roughly twice as many benign samples as malicious samples in their dataset.

In fact, at first glance, we might believe that adding more malware samples to their dataset or improving their quality would improve the results. However, as shown in Table 3, when we trained their model with our dataset, its performance was worse, which may be a result of the lack of sample variation in their dataset, as it is very likely that our dataset exploits sample features that were not present in their dataset.

In Fig. 5, recall values can vary between K-fold executions by 3% to 16%, indicating a classification imbalance. Considering the results obtained with our proposed dataset, we observe a significant decrease in performance statistics. We obtain an extremely low value for TPR, which is nowhere near the value of approximately 73% that can be obtained with their dataset, whilst FPR maintains a good value (around 0.1%).

Furthermore, recall values are clearly worse when we apply their model to our proposed dataset, as shown in the bottom

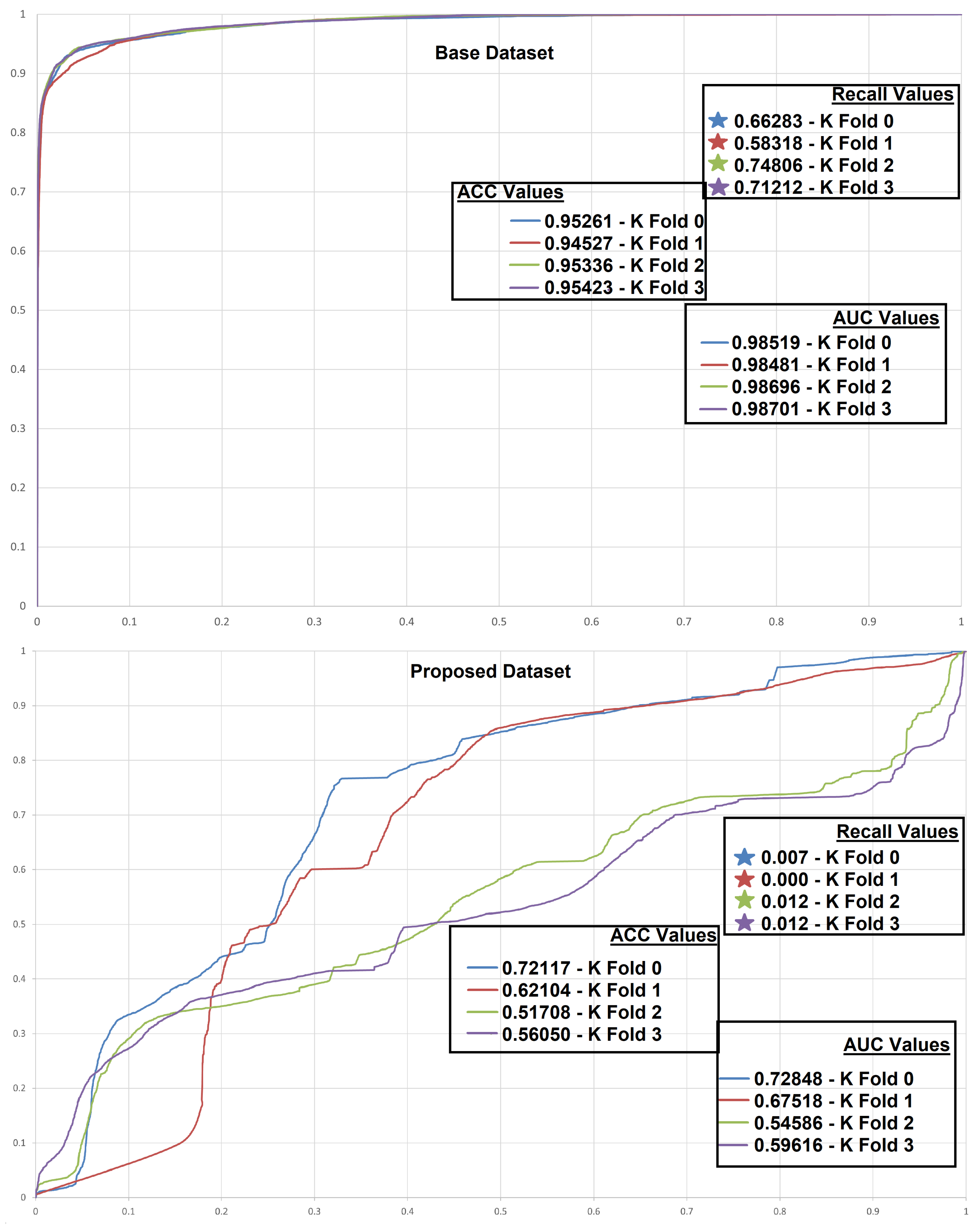

FIGURE 5. Zhang et al. model result comparison across different K-folds.

section of Fig. 5, because the model is not improving when taking 0.001 as a baseline FPR value.

This situation demonstrates the model's inability to accurately classify malware samples, as doubling the proportion

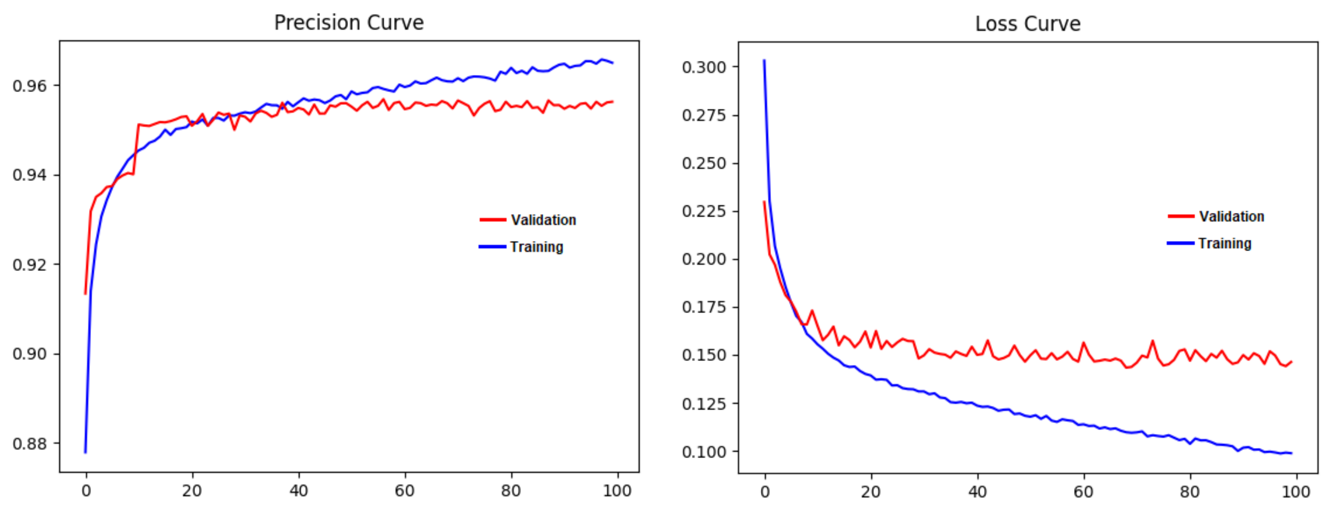

FIGURE 6. Comparison of training and validation precision and loss with our proposed model and dataset.

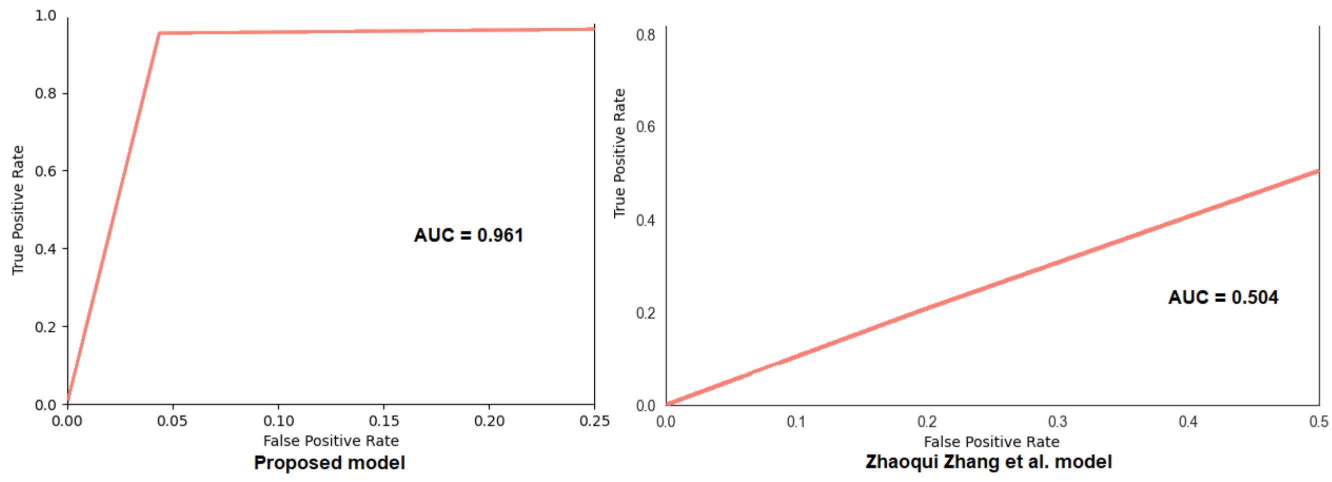

FIGURE 7. Comparison of the ROC curves between both models.

of malware in the test significantly decreased the achieved accuracy. In fact, with such a low TPR value, we can conclude that the malware samples used for training in their dataset are vastly distinct from those present in our proposed dataset.

Using our dataset in the testing phase causes the model presented by Zhang et al. to produce poorer results; consequently, our proposed dataset may be exploiting the implicit limitations of their dataset. However, although testing our own model with their dataset would have greatly benefited our research, we were unable to do so because their dataset already includes feature engineering and unprocessed samples are not provided.

In fact, when we attempt to classify Linux samples using this model, it fails completely, presenting accuracy rates below 50%. It should be noted that the Zhang et al. [17] model was only tested on Windows samples because it was trained entirely with API calls from this operating system.

This poor performance suggests that the model is overly reliant on API function names, negating the benefits of using behavior-oriented representation formats like MIST and resulting in a model that will no longer be functional because the operating system (in this case, Windows) has implemented unrecognized DLLs or API calls. Focusing sample features on patterns and operation categories, which are good indicators of software behavior, can improve robustness and possibly accuracy in these situations.

To evaluate the performance of our model on our own proposed dataset, we used a sample distribution of 80% for training and 20% for testing, with samples split into four distinct K-folds. As shown in Table 3, the proposed model achieves superior results in terms of accuracy, precision, and recall, whereas the model proposed by Zhang et al. is generally inferior in terms of performance. The obtained curves for the training and validation phases are depicted in Fig. 6.

During the execution of the different epochs, it was observed that the model proposed by Zhang et al. achieved good results (over 89% in terms of accuracy) during the initial training steps. This could mean that the model is too deep or complicated for this task, since our approach uses a simpler model architecture and obtains significantly better results.

On the other hand, it is possible that their feature engineering technique is unable to capture features as relevant as those in our proposed dataset. Moreover, the AUC value obtained for our proposed model was 96.1%, whereas for the recall values, as seen in the ROC curve in Fig. 7, higher TPR rates are obtained earlier than in the Zhang et al. model; however, the TPR rates obtained for an FPR value of 0.01 are not as good as those published by Zhang et al.

As can be seen in Table 3, following the training phase, we obtained a second dataset of over a thousand newer samples for the testing phase in order to provide our model with new samples as input. With this additional set of samples, the results are slightly worse but still greater than 90%.

During the testing phase, we measured the Mean Time To Detect (MTTD) of our proposed solution. The MTTD was calculated by splitting the time into two components: the conversion of the report to an image and the actual detection time of the model when the image is used as input. Our feature engineering method takes approximately 271.6 ms per sample to convert the report into an image, while the CNN model takes only 1.8 ms per sample for the detection process. Consequently, the entire solution operates at an average of 273.4 ms per sample.

V. CONCLUSION

In this paper we have analyzed the malware detection problem, described various possible approaches along with their respective drawbacks, and emphasized the use of software behavior to determine the security nature of software.

In this regard, the API call sequence made by each software sample under analysis is crucial, as it can provide an accurate representation of its intent, whether that is to block the system using encryption, infect other computers on the network, or simply remain hidden on the local system. All of these circumstances require API calls to provide the necessary encryption, network connection, or Windows registry functionality.

As previously mentioned, there are not many datasets available that pertain to malware behavior, and the few that do exist are primarily in proprietary formats, based on a particular feature engineering method, or utilize only partial API call information. This results in the development of numerous incompatible datasets and representation formats.

Achieving a standard method for representing the behavior of malware could considerably improve collaboration between different research groups, thereby enhancing feedback and fostering innovation. Moreover, by adopting a more abstract strategy, we would be less reliant on API call names that are specific to particular operating systems, a problem that can render valid datasets useless when API call names change or malware employs newer functionality.

Due to this, we have developed a fully reproducible dataset that includes the entire sampling workflow, from the initial state (JSON reports) to the 2D image transformation process. In addition, this dataset includes samples from two distinct operating systems, with categories serving as the first level of abstraction. We have also developed a machine learning model that achieves promising results when compared to previously published models and even better results with the dataset we propose.

Future research could involve the addition of samples from other operating systems to our model and the improvement of its performance. Also, using machine learning to detect malware in standard computing environments is still a considerable challenge; this is because sample feature extraction, 2D image conversion, and inference processes need to be very efficient for real-time analysis.

REFERENCES

[1] Kaspersky. *Damage Control: The Cost of Security Breaches it Security Risks Special Report Series*. Accessed: Feb. 13, 2022. [Online]. Available: https://media.kaspersky.com/pdf/it-risks-survey-report-cost-of-security-breaches.pdf

[2] CyberEdge Group. *2021 Cyberthreat Defense Report*. [Online]. Available: https://cyber-edge.com/wp-content/uploads/2021/04/CyberEdge-2021-CDR-Report-v1.1-1.pdf

[3] M. Alazab, S. Venkatraman, P. Watters, M. Alazab, and A. Alazab, "Cybercrime: The case of obfuscated malware," in *Global Security, Safety and Sustainability & e-Democracy*, C. K. Georgiadis, H. Jahankhani, E. Pimenidis, R. Bashroush, A. Al-Nemrat, Eds. Berlin, Germany: Springer, 2012, pp. 204–211.

[4] X. Yuan, "PhD forum: Deep learning-based real-time malware detection with multi-stage analysis," in *Proc. IEEE Int. Conf. Smart Comput. (SMARTCOMP)*, May 2017, pp. 1–2.

[5] W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network for representation learning," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2017, pp. 712–717.

[6] A. Nadler, A. Aminov, and A. Shabtai, "Detection of malicious and low throughput data exfiltration over the DNS protocol," *Comput. Secur.*, vol. 80, pp. 36–53, Jan. 2019.

[7] J. Singh and J. Singh, "A survey on machine learning-based malware detection in executable files," *J. Syst. Archit.*, vol. 112, Jan. 2021, Art. no. 101861.

[8] A. Sami, B. Yadegari, H. Rahimi, N. Peiravian, S. Hashemi, and A. Hamze, "Malware detection based on mining API calls," in *Proc. ACM Symp. Appl. Comput.* New York, NY, USA: Association for Computing Machinery, 2010, pp. 1020–1025.

[9] M. Torres, R. Álvarez, and M. Cazorla, "Improving malware detection with a novel dataset based on API calls," in *Proc. Int. Workshop Soft Comput. Models Ind. Environ. Appl.* Cham, Switzerland: Springer, 2022, pp. 289–298.

[10] P. Trinius, C. Willems, T. Holz, and K. Rieck, "A malware instruction set for behavior-based analysis," Univ. Mannheim, Tech. Rep. TR-2009-007, Dec. 2009.

[11] C.-I. Fan, H.-W. Hsiao, C.-H. Chou, and Y.-F. Tseng, "Malware detection systems based on API log data mining," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, vol. 3, Jul. 2015, pp. 255–260.

[12] U. Baldangombo, N. Jambaljav, and S.-J. Horng, "A static malware detection system using data mining methods," 2013, *arXiv:1308.2831*.

[13] E. Masabo, K. S. Kaawaase, and J. Sansa-Otim, "Big data: Deep learning for detecting malware," in *Proc. IEEE/ACM Symp. Softw. Eng. Afr. (SEiA)*, May 2018, pp. 20–26.

[14] D. Rabadi and S. G. Teo, "Advanced windows methods on malware detection and classification," in *Proc. Annu. Comput. Secur. Appl. Conf.* New York, NY, USA: Association for Computing Machinery, 2020, pp. 54–68.

[15] M. Ficco, "Malware analysis by combining multiple detectors and observation windows," *IEEE Trans. Comput.*, vol. 71, no. 6, pp. 1276–1290, Jun. 2022.
[16] S. Gamage and J. Samarabandu, "Deep learning methods in network intrusion detection: A survey and an objective comparison," *J. Netw. Comput. Appl.*, vol. 169, Nov. 2020, Art. no. 102767.
[17] Z. Zhang, P. Qi, and W. Wang, "Dynamic malware analysis with feature engineering and feature learning," in *Proc. AAAI Conf. Artif. Intell.*, Apr. 2020, vol. 34, no. 1, pp. 1210–1217.
[18] Y. Liu and Y. Wang, "A robust malware detection system using deep learning on API calls," in *Proc. IEEE 3rd Inf. Technol., Netw., Electron. Autom. Control Conf. (ITNEC)*, Mar. 2019, pp. 1456–1460.
[19] F. O. Catak, J. Ahmed, K. Sahinbas, and Z. H. Khand, "Data augmentation based malware detection using convolutional neural networks," *PeerJ Comput. Sci.*, vol. 7, p. e346, Jan. 2021.
[20] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 2818–2826.
[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[22] J. J. Luke, R. Joseph, and M. Balaji, "Impact of image size on accuracy and generalization of convolutional neural networks," *Int. J. Res. Anal. Rev.*, vol. 6, no. 1, p. 70, Feb. 2019.
[23] S. Targ, D. Almeida, and K. Lyman, "Resnet in resnet: Generalizing residual architectures," 2016, *arXiv:1603.08029*.
[24] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, Jun. 2006.

MANUEL TORRES received the bachelor's degree in computer science and the master's degree in cybersecurity from the University of Alicante, in 2019 and 2020, respectively, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Artificial Intelligence. In addition to his academic achievements, he has a proven track record of success in system administration and DevSecOps in international companies.

RAFAEL ÁLVAREZ received the bachelor's and master's degrees in computer science, in 2001, and the Ph.D. degree in computer science, in 2005. He is currently an Associate Professor with the Department of Computer Science and Artificial Intelligence, University of Alicante. He is a member of the Computational Security and Cryptology Research Group. His research interests include security, cryptography, machine learning, and their applications in computer science. He has participated in numerous international conferences and he has been published in prestigious journals. He received the Extraordinary Doctorate Award, in 2009.

MIGUEL CAZORLA (Senior Member, IEEE) received the degree in computer engineering and the Ph.D. degree in computer engineering from the University of Alicante, in 1995 and 2000, respectively.

In 1995, he started as an Assistant Professor with the University of Alicante, where he has been a Full Professor, since 2017. He has published more than 70 papers indexed in JCR (with more than 20 in Q1) and more than 100 publications in national and international conferences. He has supervised 19 Ph.D. theses and he is a principal investigator in several national projects (CICYT, Challenges), and having completed multiple transfer contracts with the industry. He is a member of different program committees of national and international conferences. His research interest includes computer vision. From the beginning, he applied these skills to try to solve robotic tasks. In recent years, he has diversified his lines to apply deep learning techniques to different areas (medical image, object recognition, depth estimation, and identification of traffic objects). All his research in recent years has focused on social robotics, that is, applying these techniques to help dependent persons.