RESEARCH ARTICLE OPEN ACCESS

A Technique for Dynamic Malware Detection through Application Programming Interface (API) Calls

D.I. Ekpo*, O.E. Taylor**, D. Matthias***
*(Computer Science, Rivers State University, Nkpolu Port Harcourt
Email: Daniel.ekpo@ust.edu.ng)
** (Computer Science, Rivers State University, Nkpolu Port Harcourt
Email: Taylor.onate@ust.edu.ng)
*** (Computer Science, Rivers State University, Nkpolu Port Harcourt
Email: matthias.daniel@ust.edu.ng)

----------**********----------

Abstract:

Dynamic malware attack through Application Programming Interface calls involves using malicious code to interact with an application's APIs in real-time. The goal is to exploit vulnerabilities in the application or its underlying infrastructure, allowing the attacker to gain access to sensitive data or take control of the system. This type of attack is often used to steal credentials, execute unauthorized commands, or install additional malware. Due to the problem of dynamic malware attacks through API calls, this paper presents a technique for the detection and classification of dynamic malware attacks through API calls. The paper adopted Object Oriented Analysis and Design (OOAD) as the design methodology and used python programming language for the implementation of the system. For the detection and prevention of the dynamic malware attacks, the system trained a Recurrent Neural Network (RNN) algorithm on a dataset comprises different signatures and behavioral patterns of dynamic malware attacks through API calls. The proposed RNN model was able to learn and understand the dataset accurately. The training results shows that the RNN model gave about 98.84% accuracy during training. The RNN model was tested on the test data and the performance of the model gave an accuracy of 99%. The RNN model was deployed for production. A web-based system was developed and integrate the recurrent neural network model into it for easy identification and prevention of dynamic malware attacks from gaining access to the system via API calls.

Keywords — **Dynamic malware, Recurrent Neural Network, Application Programming Interface (API), Security Threats.**

----------**********----------

I. INTRODUCTION

Dynamic malware attacks pose a significant threat to the security and integrity of computer systems and networks. In recent years, cybercriminals have become increasingly sophisticated in their methods, employing advanced techniques to evade traditional security measures. One such technique involves leveraging Application Programming Interface (API) calls to execute malicious activities and exploit vulnerabilities within a system.

APIs serve as the interface between different software components, enabling them to communicate and interact with one another. They provide a convenient and standardized way for developers to access and utilize the functionalities

of various software libraries and services. However, malicious actors can also exploit the openness and flexibility of APIs to carry out attacks.

The study of dynamic malware attacks through API calls aims to understand the techniques and strategies employed by attackers to exploit APIs for malicious purposes. By analyzing the characteristics of API-based attacks, researchers and security practitioners can develop effective countermeasures to detect, prevent, and mitigate such threats.

This research field encompasses various aspects, including identifying and analyzing malicious API calls, detecting anomalous API behaviors, developing machine learning and data mining techniques to identify patterns indicative of API-based attacks, and designing robust security mechanisms to protect against such threats. The ultimate goal is to enhance the security posture of computer systems and networks by improving our understanding of API-based attack vectors.

This paper comprehensively reviews the existing literature on dynamic malware attacks through API calls. We discuss the different types of API-based attacks, the underlying mechanisms employed by attackers, and the state-of-the-art approaches proposed to detect and defend against such attacks. By examining the current research trends and identifying research gaps, we aim to provide insights for future studies and developments in this important field of cybersecurity.

II. LITERATURE REVIEW.

With the goal of creating extremely accurate malware detection based on many kinds of dynamic behaviour characteristics, (Pengbin et al., 2018) offer an efficient dynamic analysis framework named EnDroid. Theft of private information, fraudulent membership to premium services, and malicious service communication are just some of the system-level threats that may be tracked using these capabilities. EnDroid also uses a feature selection algorithm to filter out extraneous data and isolate key behavioural characteristics. By using a runtime monitor to extract behaviour data, EnDroid is able to use an ensemble learning algorithm to determine whether or not an app is harmful. They demonstrate the efficiency of EnDroid on two datasets via experimental results. In addition, their model has the highest classification performance and shows promise in detecting Android malware.

Word embedding was first proposed by (Eslam and Ivan, 2020) as a method for deducing the contextual link between API functions in malware call sequences. In addition, they provide a strategy for grouping functions with comparable contextual characteristics. The experimental findings demonstrate a clear separation between malicious and benign sequences of calls. On the basis of this differentiation, they provide a novel Markov chain-based technique for detecting and predicting malware. They create a semantic transition matrix that represents the true relationship between API functions by simulating the behaviour of malware and goodware API call sequences. The average detection accuracy of their suggested models is 0.990, with a false positive rate of 0.010. They also offer a method that may prevent malicious payloads from executing, rather than discovering them post-execution and then fixing the harm.

An technique based on dynamic analysis with process mining is proposed (Mario et al., 2019) for detecting malware and exploring its phylogeny. The method makes use of process mining methods to define a mobile app's activity by identifying relationships and recurrent execution patterns in the system call traces collected from the app. The run-time fingerprint of the programme is described as a collection of declarative restrictions between system calls, which is what we retrieve as a characterisation. To determine (1) if a programme is malicious or trustworthy, (2) which family of malware it belongs to, and (3) how it varies from other known variations within the same malware family, the so-defined fingerprint of the application in question is compared with those of known malware. It has been proved via empirical research on a dataset of 1200 trustworthy and malicious programmes across 10 malware families that the method has a high degree of discriminating that may be used for the purposes of malware detection and the study of malware evolution.

In this paper, Taylor et al. (2020) described a model for identifying phishing websites that makes use of a deep neural network method and a support vector classifier. The dataset we used contained a

total of 98,019 website urls, including 48,009 genuine urls and 48,009 phishing urls. To prepare the dataset for training, we removed all Nan and finite values during preprocessing. Using feature extraction, we removed the dataset's dimension and several unwanted feature columns, reducing the dataset from 16 to 2 columns: the domain feature column (which contains the domain names/website URLs) and the label feature column (this holds the binary values 0 and 1, where 0 represent a legitimate website Url and 1 represent a phishing website). To create a vector of term/token counts, we used CountVectorizer on text documents in the domain column. CountVectorizer additionally allows for text input to be pre-processed before the vector representation is generated. Their proposed deep learning method achieved an accuracy of 98.33% on the same 98,018 url dataset as the support vector classifier (after training).

Using a Random Forest Classifier and Principal Component Analysis, Taylor and Ezekiel (2022) describe a sophisticated system for identifying behavioral bootnet attacks (PCA). The foundation of the system is a botnet dataset used to develop a reliable model for identifying Bootnet assaults. The pandas, package was used to clean the data in the dataset before processing. To prevent data inconsistency, PCA was employed to reduce the dataset's dimensionality. The PCA output was fed into a random forest classifier to help make predictions. Training the random forest classifier using 1000 estimators. The results of the model are encouraging, indicating an accuracy of almost 99 percent.

Jeremiah and Mattias (2022) proposed a hybrid model that combines logistic regression (LR) and decision tree (DT) with some tunable parameters for the design and training of classifiers in data mining techniques for real-world problems in Python (Spyder IDE) with Sklearn as the underlying data source library. Using logistic regression and Decision Tree models, we were able to verify that the system performed as expected of it. The end outcome was an increase in accuracy from the baseline of 81.42 percent to between 86.33 and 87.62 percent.

A innovative hybrid strategy combining Dynamic Malware Analysis, Cyber Threat Intelligence, Machine Learning (ML), and Data Forensics was developed by Nigat et al. (2021). IP reputation is forecasted in the pre-acceptance stage using the idea of big data forensics, and related zero-day assaults are classified using behavioural analysis utilising the Decision Tree (DT) approach. The suggested method concurrently evaluates the severity, risk score, confidence, and longevity of large data forensic concerns while drawing attention to them. Both the ML approaches used to get the best F-measure, accuracy, and recall scores are compared, and the complete reputation system is compared to current reputation systems to see how well it performs. Our suggested architecture is not only able to cross-reference with external sources, but it can also mitigate security concerns that were previously ignored by legacy reputation search engines.

Android users may now benefit from the MalDozer software created by (Karbab et al., 2018). This Android app relies on deep learning-based series categorization. MalDozer starts with the bare bones of an app's API strategy calls and learns to identify Android malware by separating malicious and benign cases. MalDozer is a malware detection application for Android devices that may be used on smartphones, tablets, and even Internet of Things devices. They use a wide variety of Android malware datasets, ranging from 1K to 33K malicious apps, and 38K benign apps, to evaluate the MalDozer app. With an F1-Score of 96% e99% and a false positive rate of 0.06 % e2% on a comprehensive testing dataset, their results demonstrate that MalDozer can accurately detect malware and assign it to its authentic families.

A new approach for identifying and categorising Android malware has been proposed (McLaughlin et al., 2017). A Deep Convolutional Neural Network technique was used in the suggested framework (CNN). Malware detection and classification relies on a static analysis of the raw opcode sequence recovered from an executable's disassembly. The network intuitively picks up on elements indicative of malware from the primitive opcode sequence, doing away with the need for specially created malware features. Long n-gram-like characteristics, which aren't computationally feasible with current methods, may also be used

because of the network setup. Once the network is ready, it can be efficiently executed on a GPU, allowing a very large number of files to be scanned in a short amount of time.

De-LADY (DLearning-based Android malware detection utilising Dynamic characteristics) is an obfuscation-resistant method proposed by Shihang et al. (2021). Dynamic analysis of a programme running in an emulation environment provides the basis for this. There are a total of 13533 applications across various industries (such as finance, entertainment, and utilities) that are used to test the suggested method. With a 98.08 percent detection rate and an F-measure of 98.84 percent, De-LADY is very successful.

To detect malicious software on Android devices, a new method has been proposed (Kim et al., 2017). Features are enhanced using a fact based or comparability based component extraction approach for effective feature portrayal on malware detection. Their system uses a variety of features to reflect the attributes of Android apps from a variety of perspectives. A multimodal Deep Learning technique for identifying malware applications/files was also presented. They ran a battery of trials with a total of 41,260 cases to see how well their model worked. When compared to other Deep Neural Network models, they discussed the superior accuracy of their own. In addition, they assessed their framework in a number of ways, such as model update efficiency, component usefulness, and representation approach. They also compared their framework's performance to that of other methods, such as a deep learning-based approach.

Traditional Machine-Learning Techniques and Deep learning approaches are compared and contrasted using public and private datasets for malware identification, classification, and grouping (Vinayakumar et al., 2019). Their exploratory research makes use of public and private datasets, both of which include train and test portions that are separate from one another and were collected at different periods. They also suggest a new way of dealing with images that has good limits for use in Machine Learning and Deep Learning. Extensive testing and evaluation of these methods show that they perform better than conventional Machine-learning Algorithms. Overall, their study provides a flexible and composite Deep-Learning system for real-time execution, allowing for the effective visual finding of malware.

(Souri and Hosseini, 2018) provides a comprehensive analysis of the malware identification technologies that make use of Data Mining techniques. Additionally, it divides malware recognition methods into two primary categories: signature-based and behavioral-based approaches. Specifically, they will be focusing on the following areas: (1) providing a rundown of the latest developments related to malware detection methods in Data mining; (2) presenting a precise and organised outline of the latest approaches to dealing with Machine-Learning components; (3) delving into the design of the massive techniques used in the malware detection method; and (4) analysing the important elements of classifying malware methods in Data mining. This research helps researchers get a holistic understanding of evaded malware identification and enables specialists to make informed evaluations moving forward.

To deal with the exponential growth of Android malware, SIGPID is introduced (Li et al., 2018), a malware identification framework based on authorisation usage breakdown. By mining the consent information, they were able to identify the three key authorizations that may be convincing for distinguishing between benign and harmful programmes. In order to distinguish between malicious and safe programmes, SIGPID employs Machine Learning based categorization approaches. Based on their findings, just 22 permissions are really necessary. They contrasted the efficacy of their technique, which required just 22 permissions, to that of a standard procedure that considered every possible authorisation. When using Support Vector Machine (SVM) as the classifier, they are able to achieve results that are comparable to the baseline method in terms of accuracy, recall, precision, and F-measure (more than 90%). They evaluated it in relation to other cutting-edge strategies. Their improved SIGPID is able to differentiate 93.62 percent of malware in the training set and 91.4 percent of tests on unknown/new malware.

According to (Yanfang, 2017) conducted a high-level data mining scan for malware detection. They

begin with a brief introduction to malware and the anti-malware sector before moving on to the technical specifications for malware identification. In addition, they reviewed existing methods for detecting sophisticated malware. The recognition process is broken down into two stages in these methods: component extraction and clustering. Smart malware detection techniques are often shown using extracted components and clustering techniques. They look into the component extraction and clustering techniques in depth. Finally, they make predictions about future trends in malware development after examining the additional problems and difficulties of malware recognition using Data mining techniques.

Taylor et al. (2021) provides a Deep Learning-based strategy for finding malware. A malware dataset, including both malicious and clean files, is used by our system. This dataset has 72 columns. Input data totalled 18929, and the Deep Forward Neural Network Algorithm was run with a 32-batch size and a 50-epoch epoch to train the Deep Learning Model's Dense layer, two input layers, and one output layer. Their system's accuracy improved to around 99.94% after training. For real-time detection and classification of dangerous applications and benign applications, the model was stored and delivered to web using the python framework. They also compared our trained model to other existing systems and found that our proposed model is more accurate.

III. DESIGN METHODOLOGY

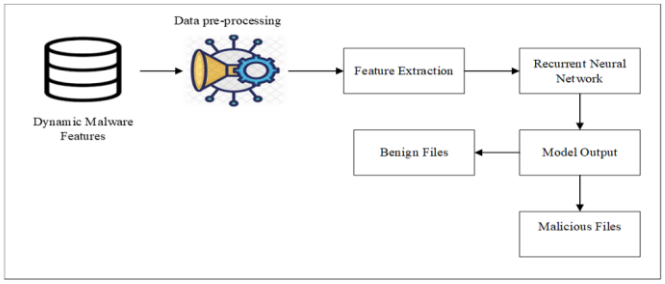

Figure 1: Architecture of the Proposed System.

A. ***Dataset*:** There are 1,079 API call sequences from legitimate software and 42,797 from malware in the dataset. The 'calls' portions of Cuckoo Sandbox reports are parsed to construct API call sequences, with each sequence consisting of the first one hundred consecutive API calls connected with the parent process. Goodware was gathered from portablepps.com and a 32-bit Windows 7 Ultimate directory, while malware was gathered via VirusShare. The dataset was made more diverse by include software available for download online as well as locally developed software. Cuckoo Sandbox, a popular open-source automated malware analysis system that can observe processes' activity while operating in isolation, was used to collect the API call sequences from each sample. Figure 3.3 displays a subset of the dataset.

	sha256	labels	0	1	2
0	5c18291c481a192ed5003084dab2d8a117fd3736359218...	0	LdrUnloadDll	CoUninitialize	NtQueryKey
1	4683faf3da550ffb594cf5513c4cbb34f64df85f27fd1c...	0	NtOpenMutant	GetForegroundWindow	NtQueryKey
2	9a0aea1c7290031d7c3429d0e921f107282cc6eab854ee...	0	GetForegroundWindow	DrawTextExW	GetSystemInfo
3	e0f3e4d5f50afd9c31e51dd9941c5a52d57c7c524f5d11...	0	NtQueryValueKey	LdrUnloadDll	GlobalMemoryStatus
4	ec2b6d29992f13e74015fff0b129150b4afae15c593e4b7...	0	LdrUnloadDll	GetSystemTimeAsFileTime	NtOpenKey
5	9cc731c2a95ea1d5b96396c1a0caf976e2965ee8a54b8e...	0	NtDuplicateObject	RegCloseKey	LdrUnloadDll
6	c8b2364e5f1b4b860cf20509ced195f86c59d41181c8ab...	0	LdrGetProcedureAddress	SetUnhandledExceptionFilter	NtTerminateProcess
7	46822662959c5efe393697dc73ae6a852aae9147751acd...	0	NtQueryValueKey	CoInitializeEx	CoUninitialize
8	282eb13c914a0a886560622 1f51518208d50abac01801d...	0	NtDuplicateObject	DeviceIoControl	GetVolumePathNameW
9	5a9a5ae741322bfea383c1384d616e466c435a24c44e33...	0	CreateToolhelp32Snapshot	LdrUnloadDll	NtOpenSection
10	c626626554cab357086defacab2f15497cee3adc715518...	0	LdrUnloadDll	LoadStringW	NtOpenKey
11	2ab30d33633d0259559452d65e69762b189c9006d3b31f...	0	LdrUnloadDll	NtOpenSection	GetForegroundWindow

Figure 2: Dataset Sample

B. ***Data Pre-processing***: This concerns vetting nan values and cleaning up the dataset from noise. The preprocessing may be seen below in its mathematical representation.

$$w = \frac{(w-f+2p)}{x+1} \quad \text{Eqn. 1}$$

To calculate the width of the output data (d), we need to know the filter size (f), the padding size (P), and the stride (s). Below is the mathematical expression that will be used to refine the data using the gaussian function.

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}} \quad \text{Eqn. 2.}$$

Where G is the Gaussian function with x and y as the input data, σ is the learning rate, and e is the edge of the object

C. Feature Extraction

This is related to the features or columns chosen for usage in the deep learning model's training process. Here, we extracted two useful

features/columns from the original dataset to use as the basis for a new dataset. Name and Malware are the columns in question. The Malware column has values of 0 and 1, where 0 represents benign files and 1 represents malware files, while the Name column comprises 19612 apps and files that are both malicious and safe (Unsafe). A hypervisor is a piece of software that acts as an intermediary between the host computer and the virtual machines it hosts. Thus, data can be gathered from the underlying hardware, the hypervisor, and the virtual machine. To extract and gather these characteristics, we use Xentrace, a hypervisor-based tracking tool, and perf, a Linux-based performance collecting tool. Figure 3 depicts the extracted characteristics of the dataset.

	hash_functions	labels
0	5c18291c481a192ed5003084dab2d8a117fd3736359218...	0
1	4683faf3da550ffb594cf5513c4cbb34f64df85f27fd1c...	0
2	9a0aea1c7290031d7c3429d0e921f107282cc6eab854ee...	0
3	e0f3e4d5f50afd9c31e51dd9941c5a52d57c7c524f5d11...	0
4	ec2b6d29992f13e74015ff0b129150b4afae15c593e4b7...	0
5	9cc731c2a95ea1d5b96396c1a0caf976e2965ee8a54b8e...	0
6	c8b2364e5f1b4b860cf20509ced195f86c59d41181c8ab...	0
7	46822662959c5efe393697dc73ae6a852aae9147751acd...	0
8	282eb13c914a0a9865606221f51518208d50abac01801d...	0
9	5a9a5ae741322bfea383c1384d616e466c435a24c44e33...	0
10	c626626554cab357086defacab2f15497cee3adc715518...	0
11	2ab30d33633d0259559452d65e69762b189c9006d3b31f...	0
12	09f3023554be864f31d80f2e7e7c7e824d79a69ddf84f1...	0
13	e79388de9271b793798aeb38512c8ef7f621e94f473a53...	0
14	c0dd75b2bffa12cc633d2d38fc766a6675cb86fb517f52...	0

Figure 3: Extracted Features

D. ***Recurrent Neural Network****:* The LSM technique was used to teach the model. Malware data will be used to train the LSTM model. The Long Short-Term Memory (LSTM) algorithm is a kind of RNN. TensorFlow Framework and the Keras library will be used to create the LSTM model. Because of the sequential nature of the Keras API, we construct the network in layers. Here are the different levels:

A 100-dimensional embedding in which each input word is converted into a vector. We provide pre-trained weights as a parameter to the embedding (more on this in a moment). If the embeddings are stable, we may leave trainable set to False.

Any words for which an embedding has not been previously taught will be hidden using a Masking layer. Training embeddings should proceed without this layer.

A layer of LSTM cells with dropout to avoid overfitting serves as the network's central processing unit. The sequences are not returned since we are only using a single LSTM layer; if you are using two or more layers, this should be corrected.

A Dense layer that is completely linked and is activated via relu. As a result, the network's capacity for representation grows.

Overfitting the training data is avoided via a Dropout layer.

Dense layer of output connections. The result of this process, which employs softmax activation, is a probability for each word in the vocabulary.

E. ***Output:*** The output shows the output of the system after various inputs has been entered. The output of the system can be either malicious file and Benign Files.

IV. COMPONENT DESIGN

The component design is the breakdown of the component in the proposed system architecture. This is always needful because it shows further other sub-components that were not made known in the design of the system architecture. Figure 4 shows the sub-component of the LSTM architecture.

1 Algorithm for LSTM

Here is a general outline of the LSTM algorithm:

1. Initialize the weights and biases of the LSTM network.
2. For each time step 't' in the input sequence: a. Get the current input 'x_t' and previous hidden state 'h_{t-1}'. b. Calculate the forget gate 'f_t', input gate 'i_t', and output gate 'o_t' using the following equations:
   i. forget gate 'f_t': f_t = σ(W_f . [h_{t-1}, x_t] + b_f)
   ii. input gate 'i_t': i_t = σ(W_i . [h_{t-1}, x_t] + b_i)
   iii. output gate 'o_t': o_t = σ(W_o . [h_{t-1}, x_t] + b_o) c. Calculate the candidate memory cell 'c_~t' using the following equation: c_~t =

tanh(W_c . [h_{t-1}, x_t] + b_c) d. Update the memory cell 'c_t' using the forget gate and candidate memory cell as follows: c_t = f_t * c_{t-1} + i_t * c_~t e. Update the hidden state 'h_t' using the memory cell and output gate as follows: h_t = o_t * tanh(c_t)

3. Repeat steps 2 for all the time steps in the input sequence.

4. Output the final hidden state 'h_T', which summarizes the information from the entire input sequence.

5. Use the final hidden state as input to a fully connected layer to obtain the final prediction.

Note: In the equations above, 'W_f', 'W_i', 'W_o', 'W_c' are the weight matrices, 'b_f', 'b_i', 'b_o', 'b_c' are the bias vectors, and 'σ' is the sigmoid activation function.

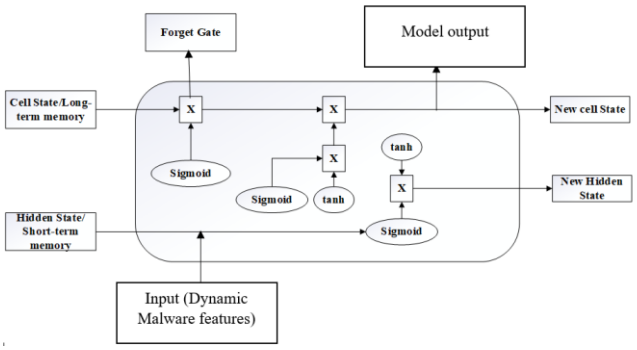

Figure 4: Component Design

The LSTM model developed for this investigation is shown graphically in Figure 3. Output layer, activation functions (sigmoid and tanh), input layer, hidden layer, and cell state make up the LSTM model. The following provides further information on the LSTM architecture:

Forget Gate: calculates the forgotten information f(t) by fitting a sigmoid to the concealed state h-(t) and the input x-(t).

Input Gate: selects which updated cell state,c-(t) data to store. To determine which values to update (,i-(t)), a sigmoid is applied to the hidden state (h-(t1)) and the input (,x-(t)), and a tanh is applied to the hidden state (,h-,t1..) and the input (,x-(t)) to create a vector of new candidate values (,c-(t)).

Cell State update: While not a true gate, this operation updates the state over time by adding,c-(t1) to the current state. combined with the product of the outputs of the input gates (i-(t), c-(t), and the forget gate (f-(t)).

• Output Gate: determines the result ct by reapplying the sigmoid to the now-revealed state,h-(t1). by taking the input,x-(t), multiplying it by the current long-term state, c-(t), and taking the result.

Algorithm 2 Feature vector generation of AP1 calls

1: Δ: Dataset of malware and benign behavior analysis reports [fi]
2: processed_api_arg: List of the generalized API calls and arguments
Given: common_malware_types, common_registry_keywords and Δ
Results: (1) Feature vector of Method 1 [Feature_VectorM1], and
Method 2 [Feature_VectorM2]
3: processed_api_arg = {}
4: foreach fi ∈ Δ do
5: Process the log file and extract its list of API calls (APIij) and arguments (ARGijk)
6: Remove the suffix from the API name [’ExW’, ’ExA’, ’W’, ’A’, ’Ex’]
in APIij ∈ fi
7: foreach ARGijk ∈ APIij do
8: switch (ARGijk)
9: Check if the common malware file types exists in command_line
10: case command_l ine:
11: Call Algorithm 4
12: Check if the regkey value is one of the common regkey for malware
13: case ’regkey’:
14: Call Algorithm 3
15: case ’path’ or ’directory’:
16: Call Algorithm 5
17: Remaining arguments with integer values, convert them into bin-based tags
18: case IsNumber(ARGijk):
19: Call Algorithm 2
20: Remaining arguments with concrete values will not be changed
21: else:
22: processed_api_arg[ARGijk] = value(ARGijk)
23: end switch
24: end foreach
25: Features are constructed using Method 1 and Method 2 formulas
26: M1processed_api_arg = Method1(processed_api_arg)
27: M2processed_api_arg = Method2(processed_api_arg)
28: Generate Method 1 and Method 2 feature vectors from the processed_api_arg using HashingVectorizer function
29: Feature_VectorM1 = HashingVectorizer(M1processed_api_arg)
30: Feature_VectorM2 = HashingVectorizer(M2processed_api_arg)
31: end foreach

32: return Feature_VectorM1, Feature_VectorM2

V. RESULTS

The experimental result is made up of two phases. The first has to do with the exploratory data analysis and the second has to do with the training of the model.

A. Exploratory Data Analysis

This section gives a detailed analysis on the malware features extracted via API calls. In other to get a better training result in terms of model accuracy, precision, and recall, data cleaning processes was carried out. Figure 5. shows a histogram of the total number of files that are malicious, and files that are not malicious.

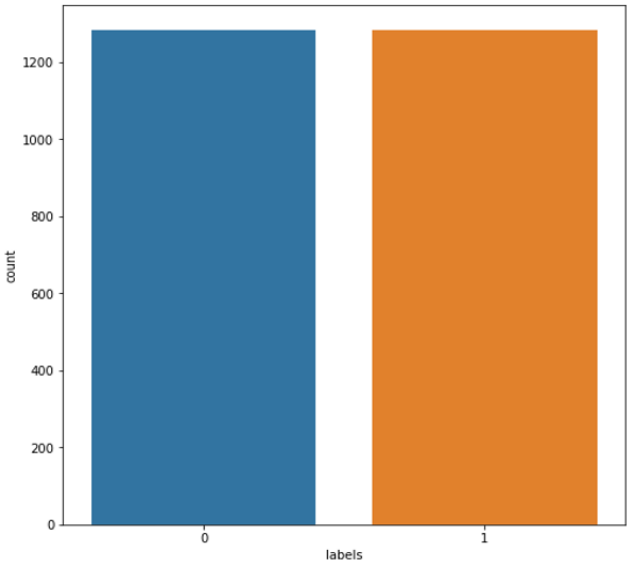

Figure 5: A Countplot of the Dataset
This shows the total number of Benign files and malicious files that are present on the dataset.

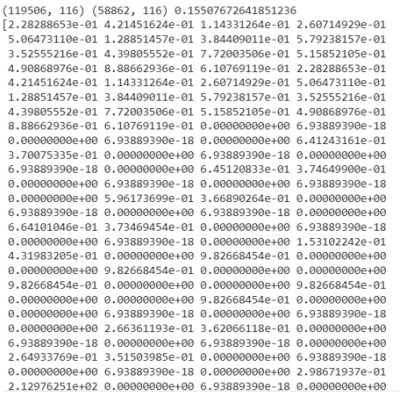

Figure 6: Tokenized and Converted to Data

B. *Model Training*

The model was trained using Recurrent Neural Network. The model was trained using the four layers. The first layer contains an input neuron of 20, and used relu as activation function. The second layer contains an input neuro of 10, and activation function of tanh. The third layer contain an input neuron of 1024, and an activation function of relu, and finally the fourth layer being the output layer used sigmoid as activation function. Other hyper parameters used in training the model are loss= binary_crossentropy, optimizer=adma, epoch, 20 and batch_size =15. The training result which displays the loss and accuracy acquired at each training steps can be seen in Figure 7. The graphical representation of the loss and accuracy values during training and testing can be seen in Figure 8 and 9 respectively. A classification report of the model can be seen in Figure 10. The classification report contains accuracy, precision, recall, and f-score measure. Figure 11 represents the confusion matrix of the model that shows the number of true predictions of the model on the categories 1 and 0. Where 0 represents Benign and 1 represents malicious attacks.

```
Epoch 1/10
65/65 [==============================] - 33s 300ms/step - loss: 0.2634 - accuracy: 0.5034 - val_loss: 0.2500 - val_accuracy:
5272
Epoch 2/10
65/65 [==============================] - 18s 272ms/step - loss: 0.2565 - accuracy: 0.4859 - val_loss: 0.2500 - val_accuracy:
4864
Epoch 3/10
65/65 [==============================] - 17s 256ms/step - loss: 0.2528 - accuracy: 0.5039 - val_loss: 0.2503 - val_accuracy:
5136
Epoch 4/10
65/65 [==============================] - 17s 261ms/step - loss: 0.2536 - accuracy: 0.5063 - val_loss: 0.2588 - val_accuracy:
5136
Epoch 5/10
65/65 [==============================] - 22s 339ms/step - loss: 0.2462 - accuracy: 0.5399 - val_loss: 0.4022 - val_accuracy:
5136
Epoch 6/10
65/65 [==============================] - 17s 266ms/step - loss: 0.0648 - accuracy: 0.9543 - val_loss: 0.2571 - val_accuracy:
5253
Epoch 7/10
65/65 [==============================] - 17s 268ms/step - loss: 0.0229 - accuracy: 0.9961 - val_loss: 0.2690 - val_accuracy:
4981
Epoch 8/10
65/65 [==============================] - 17s 264ms/step - loss: 0.0170 - accuracy: 0.9995 - val_loss: 0.2633 - val_accuracy:
5019
Epoch 9/10
65/65 [==============================] - 18s 274ms/step - loss: 0.0140 - accuracy: 1.0000 - val_loss: 0.2575 - val_accuracy:
5400
```

Figure 7: The Training Process of the Recurrent Neural Network Model Which Tests Displays the Training Steps, Loss Values and Accuracy for 1-10 Epochs (Training Steps).

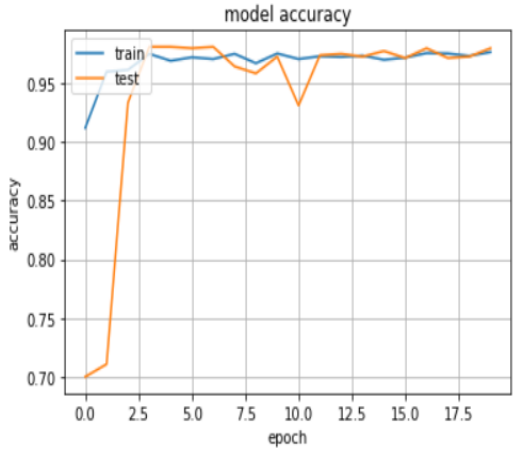

Figure 8: A Graphical Representation of Training Accuracy Vs Training Epochs
Here the model shows a training accuracy of about 99% and a test accuracy of about 98%.

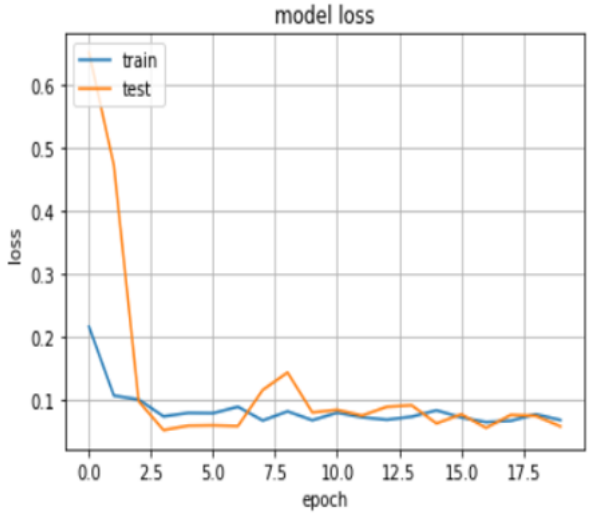

Figure 9: A Graphical Representation of Training Loss Values Vs Training Epochs
Here the model had a loss values below 0.1% for both training and testing.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1021
1	1.00	1.00	1.00	1035
accuracy			1.00	2056
macro avg	1.00	1.00	1.00	2056
weighted avg	1.00	1.00	1.00	2056

Figure 10: Classification Report of the Recurrent Neural Network Model.

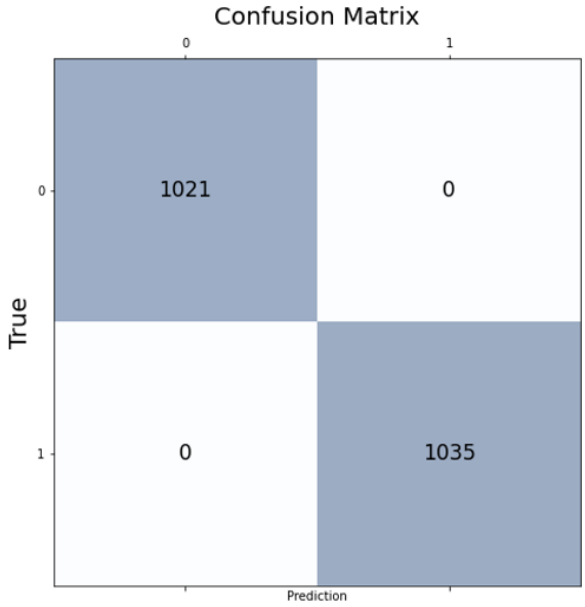

Figure 11: Confusion Matrix of the proposed Recurrent Neural Network
The confusion matrix shows the predicted result vs the actual prediction.

VI. DISCUSSION OF RESULTS

From the experiment conducted Figure 5 described the total number of files that are malicious and benign. This shows that over 1400 files are of malware attack and 1400 are of benign. Before passing the data to the deep learning model, the has function column needs to pass through tokenization process. This is to say that the query column needs to be tokenized and converted to arrays. Figure 6 shows the tokenized and transformed data. In Figure 6, all the features have been converted to numerical form. This was done so that it can fit in the deep learning model. Figure 7 shows the training process of the model. Here the model was trained on 10 steps. Figure 7 also shows the accuracy and loss obtained from each of the step completed. Figure 8 shows the accuracy obtained

for both training and validation test. The training and validation accuracy are used in testing the performance of the model during training and also on a test dataset. The model achieved a training result of about 98% and a test result of about 98%. Figure 9 shows the losses of the model for both training and testing data. The model has a loss value below 0.1 for both training and testing. Figure 10 shows the classification report of the model. The classification report is a summation of accuracy, precision, recall and f- measure. Precision has to do with the correct classification of the model in terms of false positive, false negative, true positive and true negative. The precision score of the model are about 100% correct classification for queries that are benign and 100% correct classification for files that are of malware attack. The support column in Figure 10 shows the total number of classifications that was carried out by the model. Figure 11 shows the confusion matrix of the proposed system. Confusion matrix depicts the total number of correct prediction and the total number of false classifications. The confusion matrix shows that out of 1021 classification on attacks that are of benign, the model predicted correctly for 1021 and predicted falsely for 0 times. Then for attacks that are of malicious attacks, the model predicted correctly 1035 times and predicted falsely for just 0. This shows the performance of the model is in good shape.

VII. CONCLUSIONS

This paper developed a system for the accurate detection of dynamic malware via API calls. This was achieved by analyzing the behavioural pattern of dynamic malware using exploratory data analysis. The exploratory data analysis has to do visualization of data. The visualization of data helps to uncover the patterns of the dynamic malware attack via API calls. A recurrent neural network model was trained for detecting future attacks of dynamic malware via API calls. The model was trained using dynamic malware data. The data comprise different attacks of dynamic malware that were carried out via API calls. The result of the system was compared with other existing systems. Here the results show that the proposed system outperformed the existing system with an accuracy result of 99.99%. Future recommendation of this research will be directed towards the detection of dynamic malware on edge devices. Causal Productions is credited in the revised template as follows: “original version of this template was provided by courtesy of Causal Productions (www.causalproductions.com)”.

ACKNOWLEDGMENT

REFERENCES

[1] Burnap, P., French, R., Turner, F. & Jones, K. (2018). Malware classification using self-859 organizing feature maps and machine activity data. Computer Security, 73, 399–410.

[2] Elhadi, A. A. E., Maarof, M. A. & Barry, B. I. (2013). Improving the detection of malware be- 874 haviour using simplified data dependent API call graph. International Journal Security Application, 7 (5), 875 29–42.

[3] Eslam, A. & Ivan, Z. (2018). A dynamic Windows malware detection and prediction method based on contextual understanding of API call sequence. Computers and Security, 30(40), 1-15.

[4] Gandotra, E., Bansal, D. & Sofat, S. (2014). Malware analysis and classification: a survey. 885 Journal of Information Security, 5 (02), 56.

[5] Gibert, D., Mateu, C. & Planes, J. (2020). The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. Journal of Network and Computer Applications, 153, 1-22, 2020.

[6] Karbab, E. B., Debbabi, M., Derhab, A. & Mouheb, D. (2018). MalDozer: Automatic framework for android malware detection using deep learning, Digital Investigation 24, 548-559.

[7] Kim, T., Kang, B., Rho, M., Sezer, S. & Gyu, E. (2019). A Multimodal Deep Learning Method for Android Malware Detection using Various Features, in IEEE Transactions on Information Forensic and Security, 10(3), 773-778.

[8] Li, J., Sunk, L., Yan, Q., Zhiqiang, L. Srisaan, W. & Heng, Y. (2018). Significant Permission Identification for Machine Learning Based Android Malware Detection, in IEEE Transactions on Industrial Informatics, 14(7), 3216-3225.

[9] Mario, L., Marta, C., Damiano, D., Fabio, M. & Francesco, M. (2019). Dynamic malware detection and phylogeny analysis using process mining. International Journal of Information Security, 18, 257–284.

[10] McLaughlin, N. Rincon, J., Kang, B., Yerima, S., Miller, P., Sezer, S., Safaei, Y., Trickel, E., Zhao, Z., Doupe, A, & Ahn, G. (2017). Deep Android Malware Detection, Proceeding on the Seventh ACM on Conference on Data and Application Security and Privacy, 301-308.

[11] Nighat, U., Saeeda, U., Fazlullah, K., Mian, A., Ahthasham S., Mamoun A. & Paul W. (2021). Intelligent Dynamic Malware Detection using Machine Learning in IP Reputation for Forensics Data Analytics. Future Generation Computer Systems118 (2021), 124–141.

[12] Pengbin, F., Jianfeng M., Cong S., Xinpeng X. & Yuwan M. (2018). A Novel Dynamic Android Malware Detection System with Ensemble Learning. IEEE Access, 6, 30996-31011.

[13] Qiao, Y., Yang, Y., He, J., Tang, C. & Liu, Z. (2014). CBM: free, automatic malware anal- 923 ysis framework using API call sequences. In: Knowledge Engineering and Man- 924 agreement. Springer, Berlin, Heidelberg, 225–236.

[14] Ezekiel, P. S., Taylor, O. E., & Deedam-Okuchaba, F. B. (2020). A model to detect phishing websites using support vector classifier and a deep neural network algorithm. IJARCCE, 9(6), 188-194.

[15] Taylor, O. E., & Ezekiel, P. S. (2022). A smart system for detecting behavioural botnet attacks using random forest classifier with principal component analysis. European Journal of Artificial Intelligence and Machine Learning, 1(2), 11-16.

[16] Taylor, O. E., Ezekiel, P. S., & Sako, D. J. S. (2021). A Deep Learning Based Approach for Malware Detection and Classification.

[17] Jeremiah P. R., Matthias D. (2022). Phishing URL Detection Using Machine Learning Classification Algorithms. Journal of Web Engineering & Technology, 9(3), 22-33.