International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-12 Issue-8, July 2023

# Proposing A New Approach for Detecting Malware Based on the Event Analysis Technique

**Nguyen Duc Viet, Dang Dinh Quan**

***Abstract:*** ***The attack technique using malware distribution forms is a dangerous, difficult-to-detect, and preventable attack method. Current malware detection studies and proposals often rely on two primary methods: utilising sign sets and analysing abnormal behaviours using machine learning or deep learning techniques. This paper will propose a method to detect malware on Endpoints based on Event IDs using deep learning. Event IDs are behaviours of malware tracked and collected on the operating system kernel of endpoints. The malware detection proposal based on Event IDs represents a new research approach that has not been extensively studied or proposed. To achieve this purpose, this paper proposes combining various data mining methods and deep learning algorithms. The data mining process is presented in detail in Section 2 of the paper.***

***Keywords: Malware detection; Endpoint; Event analysis technique; deep learning; Doc2Vec***

## I. INTRODUCTION

Two currently commonly used methods for malware detection include the sign-based detection method and the abnormal behaviour-based detection method [1, 2, 3, 4, 5]. Detection methods based on anomalous behaviour have been highly effective due to their ability to detect new types of malware. Behaviour-based detection approaches often seek ways to extract anomalous behaviours of malware and then employ methods and algorithms to classify it. However, it can be observed that a common characteristic of these methods is the use of techniques to extract signs and behaviours of malware based on sample datasets. These datasets are built using virtualisation tools or static analysis and network monitoring tools. Regarding virtualization tools, studies often use the Sandbox tool [6] to execute and extract malicious's behaviors. The disadvantage of Sandbox tools is that they only record behaviours within a specific time, so it will not be possible to analyse malware's behaviour comprehensively. Regarding datasets collected during the static analysis process, using them only detects anomalies when malware has spread and connected to steal data. Therefore, these traditional approaches are always bypassed by malware. To address these issues, this paper proposes a novel approach that analyses the abnormal behaviours of Event IDs. The characteristic of our approach is that, instead of using virtualisation tools to collect and extract malware's abnormal behaviours, this approach relies on Event IDs generated by the malware as a basis for detecting abnormal signs and behaviours of malware. These Event IDs are then analysed using various data mining methods to identify and aggregate malware's abnormal behaviours. Next, the Seq2Vec algorithm is proposed for synthesising and normalising the features of Event IDs. Finally, to conclude on the existence of malware in the system, we utilise deep learning algorithms such as Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), and Long Short-Term Memory (LSTM). The novelty and scientific quality of our research are as follows:

- Proposing a malware detection method based on Event IDs. This is a new approach for detecting malware on Endpoints. This approach has not yet been published in any peer-reviewed publications.
- Proposing a method to analyse malware's abnormal behaviours based on Event IDs using the Seq2Vec technique. Although the use of the Seq2Vec model to normalise text data is a common problem, applying this model to normalise malware data presents a new challenge that has not been extensively studied or used in many works. Notably, the application of this model to the process of normalising Event IDs has not been proposed by any existing research.

## II. RELATED WORKS

Studies [1, 2, 3] presented some malware detection methods. In the research [7], Zhong et al. proposed a method of using multiple deep-learning layers for malware detection. Specifically, in their proposed model, the authors proposed a detection method based on 5 phases: Phase 1: Choosing prominent static and dynamic features; Phase 2: Using the parallel improved K-means algorithm to partition the dataset into multiple one-level clusters; Phase 3: Generating multiple sub-clusters in parallel; Phase 4: Building the deep learning model for each sub-cluster in parallel; Phase 5: Classifying samples as malware or benign based on decision values of deep learning models. In the study [8], Fei Xiao et al. proposed a malware detection method using the Stacked Auto Encoders (SAEs) deep learning network. In the experimental section, the authors compared and evaluated the SAE model with other machine learning and deep learning algorithms. Experimental results showed that the SAE model brought better results than other models. Studies [9, 10] proposed a method to detect malware based on some machine learning algorithms such as Decision Tree (DT), K-Nearest Neighbour (KNN), Naïve Bayes (NB), and Support Vector Machine (SVM). In studies [11, 12] the authors proposed some malware detection methods based on Window API calls using machine learning and deep learning algorithms.

**Manuscript received on 12 June 2023 | Revised Manuscript received on 23 June 2023 | Manuscript Accepted on 15 July 2023 | Manuscript published on 30 July 2023.**

*Correspondence Author(s)

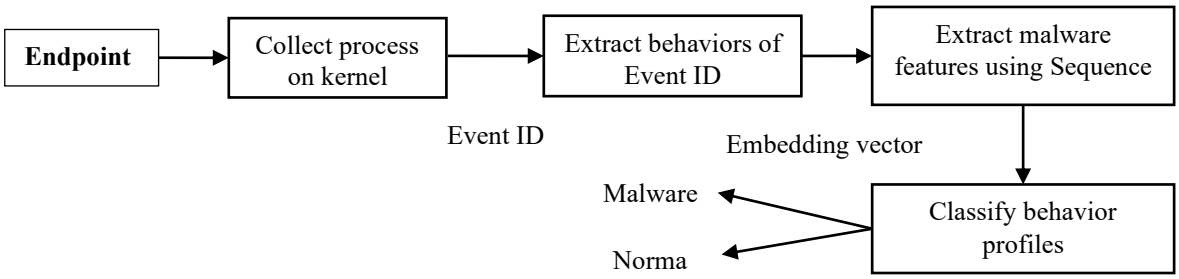
**Nguyen Duc Viet***, Institute of Post and Telecommunications Technology, Hanoi University, Vietnam. E-mail: vietnd@ptit.edu.vn, ORCID ID: 0009-0009-5609-2905

**Dang Dinh Quan**, Lecturer, Faculty of Information Technology, Hanoi University, Vietnam. E-mail: quandd@hanu.edu.vn, ORCID ID: 0009-0009-0531-5523

In addition, the report [13] listed some technology solutions for detecting malware on Endpoints (Endpoint Detection & Response) based on rule sets and behaviors. Accordingly, the technology solutions include Trend Micro EDR Apex One, Palo Alto Networks Traps, WildFire, Kaspersky EDR, Carbon Black EDR, and Falcon.

## III. THE PROPOSED MODEL ARCHITECTURE

### 3.1. The Proposed Model Architecture




:

**The architecture of the proposed APT malware detection model**

From Figure 1, the operation process of the system is as follows:

**Step 1: Collect and process Event IDs on Endpoints.** To perform the task of collecting and extracting these processes, we install and configure the primary tool, Sysmon [14]. These tools have the function of collecting processes recorded by the operating system and transferring them to the processing and monitoring centre. The methods in Sysmon are described in detail in Table I of Section 2.2.

**Step 2: Identify abnormal behaviours associated with Event IDs.** At this step, abnormal features and behaviours are extracted from the Event IDs collected from the client side. These features and behaviours are the basis for malware detection. Details of abnormal behaviours associated with Event IDs are presented in Section 2.3.

**Step 3: Identify and extract abnormal behaviours of malware.** As is known, in step 2, the research has extracted anomalous behaviours in Event IDs. Here, each file has different characteristics and a different number of Event IDs. Therefore, need a method to normalize and process these files. To accomplish this task, we propose to use the Seq2Vec model. Accordingly, each Event ID is considered as a "word", and a file is a collection of words. Finally, the file consisting of words is normalised into a homogeneous vector using the Seq2Vec model. Details of this process are described in Section 2.4.

**Step 4: Detect malware**. At this step, the malware's behaviours, which were normalised and built in step 3, are classified by a deep learning algorithm to determine the presence of malware in the system. This process is presented in detail in section 2.5 of the paper

### 3.2. The Method to Extract Processes of Malware

In this paper, to collect malware's behaviours on the operating system kernel, we propose to use the Sysmon tool [14]. The Sysmon tool is one of the powerful tools developed by Microsoft to support the task of collecting and analyzing anomalous behavior on Endpoints using the Windows operating system. Accordingly, the main 22 behaviours collected by the Sysmon tool on the Endpoints' operating system kernel are presented in the report [14] including Process creation, Network connection, Sysmon service state changed, Process terminated, Driver loaded, Create Remote Thread, etc

### 3.3. The Method to Extract Abnormal Features of Malware Based on The Processes

Thus, based on 22 Event IDs collected in the operating system kernel by the Sysmon tool, this paper will analyse these Event IDs to identify anomalous behaviours associated with each Event ID. Table 1 below lists abnormal behaviours found in Event IDs. These behaviours are the new anomalous behaviours proposed by you

**Table 1 : List of abnormal behaviors collected on the operating system kernel**

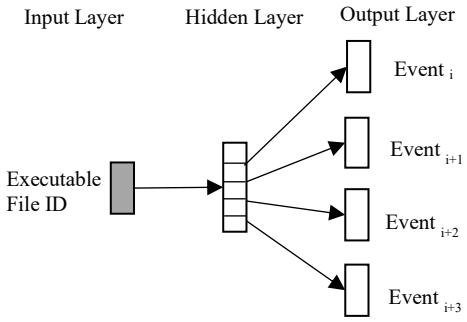
| No. | Type | Feature name | Description |
|---|---|---|---|
| 1 | KEYSTROKES Loggers | Get Async Key State | Poll the state of each key on the keyboard using the function. |
| 2 | | Get Key State | Retrieves the status of the specified virtual key |
| 3 | | Set Windows Hook | Installs an application-defined hook procedure into a hook chain |
| 4 | Network traffic monitor | WSA Socket | Create a raw socket |
| 5 | | socket | Create a raw socket |
| 6 | | bind | Bind socket to an interface |
| 7 | | WS AIoctl | Put the interface (NIC) into Promiscuous mode |
| 8 | | ioctlsocket | Put the interface (NIC) into Promiscuous mode |
| 9 | Downloader | URL Download To File | Download the file and save it to disk |
| 10 | Execution | Win Exec | Execute file |

| | | | |
|---|---|---|---|
| 11 | | Load Module | Loads and executes an application or creates a new instance of an existing application. |
| 12 | | Load Packaged Library | Loads the specified packaged module |
| 13 | | Create Process | Create a new process |
| 14 | | Shell Execute | Execute file |
| 15 | HTTP CNC Traffic | Internet Open | Initialises an application's use of the Windows Internet functions |
| 16 | | Internet Connect | Url Input |
| 17 | | HTTP Open Request | Build an HTTP request |
| 18 | | HTTP Add Request Headers | Build an HTTP request |
| 19 | | HTTP Send Request | Send HTTP Request |
| 20 | | Internet Read File | Read Response |
| 21 | Droppers | Find Resource | Find Resource |
| 22 | | Load Resource | Retrieves a handle that can be used to obtain a pointer to the first byte of the specified resource in memory. |
| 23 | | Size Of Resource | Retrieves the size of the resource |
| 24 | | Lock Resource | Retrieves a pointer to the specified resource in memory. |
| 25 | DLL Injection | Set Windows Hook | Install the filter function in the hook chain of the remote process Works only for a GUI application |
| 26 | | Load Library | Load the malicious DLL into the attacking process's address space. |
| 27 | | Get Proc Address | Retrieve the address of the filter function on the remote process. |
| 28 | | Get Windows Thread ProcessId | Get ID of Target thread. |
| 29 | | Broadcast System Message | The attacking process uses this to send messages to the victim process (internally). |
| 30 | | Virtual Alloc | Standard windows api call that allows one process to allocate memory space inside another process |
| 31 | | Write Process Memory | Writes data to an area of memory in a specified process |
| 32 | | Get Module Handle | Allows the process to determine how to access some DLL that might be loaded into the memory space |
| 33 | | Get Proc Address | Retrieves the address of an exported function or variable from the specified dynamic-link library |
| 34 | | Create Remote Thread | Create a remote thread inside a remote process |
| 35 | Hooking | Get Proc Address | Locate the address of a function to hook |
| 36 | | Virtual Protect | Set memory protection to read/write |
| 37 | | Read Process Memory | Save the first few bytes of the victim |
| 38 | | Virtual Protect | Restore memory permission to the original value |
| 39 | Process hollowing | Create Process | Create a process in a suspended state. |
| 40 | | NT Unmap View of Section | Unmap the contents of the original process from memory |
| 41 | | Virtual Alloc | Allocate a new memory address in the hollow process |
| 42 | | Write Process Memory | Brand new code is injected into the hollow process Resume Thread -> Resume the process |
| 43 | Anti-Debugger/VM detection | Get Tick Count | Identify the time to detect if a debugger is present |
| 44 | | Count Clipboard Formats | API call to determine whether the victim's clipboard was empty |
| 45 | | Get Foreground Window | API call to check if the colour of the foreground window was changing, assuming automated sandbox tools don't switch active windows around |
| 46 | | Isdebuggerpresent | Detect debugger |
| 47 | Shell Code | Get EIP | Methods SHELLCODE often uses to determine its location in memory. |
| 48 | File and Directory | Create File | Creates or opens a file or I/O device |
| 49 | | Open File | Open a file |
| 50 | | Find First File | Searches a directory for a file or subdirectory with a name that matches a specific name |
| 51 | | Find Next File | Continues a file search |
| 52 | | Get Windows Directory | Retrieves the path of the Windows directory. |
| 53 | | remove | Deletes the file specified by path |
| 54 | | Get Temp Path | Returns the path of the current user's temporary folder |
| 55 | | Delete File | Deletes the file specified by path |
| 56 | Registry Keys | Reg Open Key | Opens the specified registry key |
| 57 | | Reg Create Key | Creates the specified registry key |
| 58 | | Reg Set Value | Sets the data for the default or unnamed value of a specified registry key |
| 59 | PowerShell | System | executes an internal operating system command |
| 60 | | Win Exec | Runs the specified application |
| 61 | Service | Create Service | Creates a service object and adds it to the specified service control manager database. |
| 62 | | Control Service | Sends a control code to a service |
| 63 | | Start Service Ctrl Dispatcher | Connects the main thread of a service process to the service control manager |
| 64 | Process | Create Process | Create a new process |

| | | |
|---|---|---|
| 65 | Get Process ID | Retrieves the process identifier of the specified process |
| 66 | Process32First | Retrieves information about the first process encountered in a system snapshot |
| 67 | Process32Next | Retrieves information about the following process recorded in a system snapshot |
| 68 | Open Process | Opens an existing local process object |

### 3.4. The Method to Build Malware Behaviour using Sequence

As is known, each malware has a different number of Event IDs, making it a challenging task to standardise the length of each file. In this paper, each executable file is considered a document, and each event is a "word" in the document. The next task is how to normalize a document into a uniform vector. To perform this task, this study proposes to use the Seq2Vec model. The Seq2Vec method was proposed by Dhananjay et al. in 2016 [15]. The characteristic of this method is to vectorize files by using the Doc2Vec algorithm. In which, Doc2Vec, which was introduced by Quoc Le and Mikolov [16], includes 2 main models: Distributed Memory Model of Paragraph Vectors (PV-DM) and Distributed Bag of Words version of Paragraph Vector (PV-DBOW). In this paper, we use the PV-DBOW model. This is a similar model to the Skip-gram model for word2vec. The difference is that the input of Skip-gram is a word, while the input of PV-DBOW is a document ID (in this study, it is an executable file ID). In this model, only the softmax weights need to be stored, instead of both the softmax weights and word vectors, as in the PV-DM model. As a result, the Doc2Vec model represents processes into corresponding vectors. Figure 2 below illustrates how to vectorise an executable file using the PV-DBOW model.




**: How the PV-DBOW model works for vectorizing an executable file**

The process of applying the Seq2Vec model to the task of standardizing malware data has the following steps:

**Step 1:** Sorting the processes in the order of appearance. Representing a file as a sequence: a file has many processes, consider a file as a record and a process as a word.

**Step 2:** Vectorizing the file by using the Doc2vec algorithm using the Skip-gram model. This paper configures the Seq2Vec model with output parameters of 64, 128, and 256 features, respectively.

### 3.5. Classification Method

After the malware's processes are collected and its features are extracted and normalised, we obtain a unique vector representing the malware's features. Next, based on this feature vector, this study aims to evaluate and conclude which files are standard and which are malicious. This paper utilises various deep learning and machine learning algorithms to classify files as either usual or malware. Specifically, we propose using the following deep learning algorithms and models: Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), and Random Forest (RF). Regarding the MLP network, the study [17] presented the MLP network architecture in detail. It is built by simulating how neurons work in the human brain. MLP networks typically have three or more layers: one input layer, one output layer, and at least one hidden layer. Additionally, the efficiency of the MLP network depends on the choice of activation function. This paper will tune the activation function parameter to evaluate the effectiveness and suitability of activation functions for the malware detection task. The CNN network is a basic layer set consisting of a convolution layer, a nonlinear layer, and a fully connected layer. The structure and the terms (stride, padding, Max Pooling) of CNN were presented in detail in the research [18]. In this paper, we choose to use the ReLU activation function for CNN. Regarding the LSTM network, it was introduced in the study [19] with the ability to remember information for a long time. This is expressed in the structure of the ports in each memory cell. A memory cell consists of three primary components: the input gate, the forget gate, and the output gate. Firstly, the forget gate decides what information should be discarded in the cell state. Next, the input gate decides what information is updated into the cell state. Finally, the output gate calculates the desired output. During this process, the cell state is propagated through and updated as it passes through all nodes

## IV. EXPERIMENTS AND EVALUATION

### 4.1. Experimental Dataset

In this paper, we use normal and malware data from the source [20]. Specifically, we collected 52,135 malware files, including Agent Tesla, Azorult, Emotet, Formbook, Gandcrab, Hawkeye, Lokibot, Njrat, Pony, Qbot, Quasar, Remcos, Trickbot, Ursnif, and Vidar, among others. Regarding normal data, the research seeks ways to collect files including PE EXE, PE DLLs, JAVA, HTML, Documents, Adobe Flash, Microsoft Office, etc. The total number of malware files is 25,437.

### 4.2. Experimental Scenario

This study divides the experimental dataset into distinct components and then conducts experiments to evaluate the accuracy of the proposed models based on these sub-datasets. The whole process of separating the experimental dataset into scenarios is chosen at random, where 80% of the dataset is used for training and the remaining 20% is used for testing. To evaluate the effectiveness of the proposal in the study, we conduct 2 evaluation scenarios as follows:

**Scenario 1:** Compare and evaluate the effectiveness of deep learning methods. For this scenario, we assess the following algorithms: MLP,


Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

CNN, and LSTM. During the experiment, we tune the parameters to see the effectiveness of the deep learning models.

Thus, in this scenario, the paper evaluates three models, including Seq2Vec-MLP, Seq2Vec-CNN, and Seq2V-LSTM.

**Scenario 2:** Compare and evaluate the deep learning models with some other approaches on the same dataset.

### 4.3. Classification Measures

This paper uses 4 following four measures to evaluate the accuracy of models:

1. **Accuracy:** The ratio between the number of samples classified correctly and the total number of samples.
2. **Precision:** The ratio between the actual positive value and the total number of samples classified as positive. The higher the precision value, the more accurate the detection of malicious samples.
3. **Recall:** The ratio between the actual positive value and the total real malicious samples. The higher the recall value, the lower the rate of missing positive samples.

**F1-score:** The harmonic mean of precision and recall

### 4.4. Experimental Results

#### *4.4.1. Experimental Results of Scenario 1*

Our purpose in scenario 1 is to compare and evaluate the classification ability of the deep learning model in the malware detection problem based on the different measures presented in the previous sub-section. The experimental results of scenario 1 are presented in Tables 2, 3, and 4 below

**Table 2. Experimental results using the Seq2Vec-MLP model**

| Parameter | | Evaluation | | | | | |
|---|---|---|---|---|---|---|---|
| Features | Layers | Acc | Pre | Rec | F1 | Train time | Test time |
| 64 features | 2 | 96.86 | 94.47 | 89.15 | 91.43 | 1144.02 | 2.69 |
| | 4 | 96.9 | 94.65 | 89.14 | 91.82 | 1282.52 | 2.65 |
| 128 features | 2 | 96.88 | 93.87 | 89.89 | 90.11 | 1222.44 | 2.24 |
| | 4 | 97.07 | 95.2 | 89.53 | 92.28 | 1282.52 | 2.65 |
| 256 features | 2 | 96.96 | 94.86 | 89.27 | 91.98 | 1185.45 | 2.21 |
| | 4 | 96.05 | 94.53 | 89.06 | 92.18 | 1282.63 | 2.68 |

**Table 3. Experimental results using the Seq2Vec-CNN model**

| Parameter | | Evaluation | | | | | |
|---|---|---|---|---|---|---|---|
| Features | Layers | Acc | Pre | Rec | F1 | Train time | Test time |
| 64 features | 96 | 96.64 | 92.34 | 90.3 | 91.3 | 1552.34 | 2.89 |
| | 32-64 | 96.86 | 94.85 | 88.73 | 91.69 | 1762.69 | 3.2 |
| 128 features | 512 | 96.88 | 93.96 | 89.81 | 91.84 | 2895.74 | 5.22 |
| | 128-256 | 96.88 | 93.84 | 89.94 | 91.85 | 2482.65 | 5.25 |
| 256 features | 512 | 96.87 | 93.23 | 90.52 | 91.85 | 2962.68 | 5.23 |
| | 64-128 | **96.89** | **94.03** | **89.77** | **91.85** | **2242.75** | **3.59** |

**Table 4. Experimental results using the Seq2Vec-LSTM model**

| Parameter | | Evaluation | | | | | |
|---|---|---|---|---|---|---|---|
| Features | Layers | Acc | Pre | Rec | F1 | Train time | Test time |
| 64 features | 512-512-128 | 96.73 | 93.97 | 88.98 | 91.41 | 1105 | 8.2 |
| | 256-256-512-128 | 96.78 | 94.13 | 89 | 91.53 | 1119 | 9.4 |
| 128 features | 128-512-256 | 96.88 | 94.6 | 89.14 | 91.8 | 925.43 | 8.64 |
| | 256-512-256-128 | 96.85 | 94.26 | 89.3 | 91.71 | 1141 | 9.6 |
| 256 features | 128-172-256 | **96.93** | **94.43** | **89.57** | **91.94** | 1309.8 | 10.5 |
| | 172-512-256-512 | 96.86 | 94.57 | 89 | 91.74 | 1826.8 | 12.67 |

Based on the experimental results in Table 2, the Seq2Vec-MLP model demonstrated varying efficiency when its parameters were adjusted. However, this change is not too significant because the difference between models is only about 0.1%. Regarding the time variation between models, it is evident that when the number of hidden layers in the MLP model and the number of features in the Seq2Vec model are increased, the training time increases markedly. Regarding the accuracy of the Seq2Vec-MLP model, the model gave the highest results at the parameter {Seq2Vec: 256 features, MLP: 2 layers}. From the results in Table 3, we see that the Seq2Vec-CNN model has many similarities with the Seq2Vec-MLP model. Specifically, as the number of layers and features in the model increases, training and testing times also increase significantly. Additionally, regarding the efficiency of the detection process, the models yielded different results when the parameters were changed. However, this change is irregular. As complexity increased, accuracy did not always increase. The Seq2Vec-CNN model achieved the best results, with Accuracy, Precision, Recall, and F1-score measures of 96.89%, 94.03%, 89.77%, and 91.85%, respectively. The experimental results in Table 4 show that the Seq2V-LSTM model worked relatively effectively for both tasks of classifying malware and regular file. The best Accuracy detection result is 96.93%. This result is about 0.2% higher than the lowest result. Regarding the correct classification of regular files, this model achieved the best results, with 94.57% accuracy, when using 256 features and 4 LSTM layers. Additionally, regarding the correct classification of malware, with an efficiency of 89.57%, the Seq2V-LSTM model has demonstrated superiority compared to other models using CNN or MLP.

In terms of detection time, the more complex the model, with many LSTM layers and an extended feature vector, the more processing time is required for the Seq2V-LSTM model.

*4.4.2. Experimental Results of Scenario 2*

Our purpose in this scenario is to experiment with some other models and approaches in the malware detection task. Accordingly, in addition to the Seq2Vec-CNN model proposed in the study [21] (the experimental results of this model are shown in Table 3), we conduct experiments with the Seq2Vec-RF model [22]. This model was proposed in the study [22]. Table 5 below describes the experimental results of this model.

**Table 5. Experimental results using Seq2Vec-RF [22]**

| Parameter | | Evaluation | | | | | |
|---|---|---|---|---|---|---|---|
| **Features** | **Trees** | **Acc** | **Pre** | **Rec** | **F1** | **Train time** | **Test time** |
| 64 features | 10 | 96.01 | 94.01 | 85.03 | 89.3 | 65.97 | 0.22 |
| | 50 | 96.57 | 93.97 | 88.2 | 91 | 342.16 | 01.03 |
| | 100 | 96.61 | 93.91 | 88.17 | 90.95 | 680.71 | 2.1 |
| 128 features | 10 | 96.32 | 94.23 | 86.45 | 90.17 | 93.46 | 0.27 |
| | 50 | 96.79 | 94.61 | 88.49 | 91.45 | 473.74 | 1.36 |
| | 100 | 96.74 | 94.33 | 88.63 | 91.39 | 946.85 | 2.58 |
| 256 features | 10 | 96.41 | 94.15 | 86.93 | 90.4 | 140.71 | 0.39 |
| | 50 | 96.87 | 94.41 | 89.15 | 91.7 | 686.3 | 1.71 |
| | 100 | **96.81** | **94.13** | **89.16** | **91.58** | **1393.67** | **3.36** |

The experimental results in Table 5 show that the Seq2Vec-RF model performed best (with Accuracy, Precision, Recall, and F1-score measures of 96.81%, 94.13%, 89.16%, and 91.58%, respectively) when the RF algorithm utilised 100 decision trees and Seq2Vec employed 256 features.

*4.4.3. Discussion*

Table 6 below summarizes the results of the process of implementing the two comparison scenarios that we have evaluated

**Table 6. Comparison table of malware detection results of some models**

| Model | Evaluation | | | | | |
|---|---|---|---|---|---|---|
| | **Acc** | **Pre** | **Rec** | **F1** | **Train time** | **Test time** |
| Seq2Vec-RF [22] | 96.81 | 94.13 | 89.16 | 91.58 | **1393.67** | **3.36** |
| Seq2Vec- CNN [22] | 96.89 | 94.03 | 89.77 | 91.85 | 2242.75 | 3.59 |
| **Seq2V-LSTM** [our proposal] | **96.93** | **94.43** | **89.57** | **91.94** | 1309.8 | 10.5 |

Comparing the results in Table 6, we observe that our proposed Seq2V-LSTM model yields better results than those proposed in other studies. However, this difference is not too significant. This demonstrates that the Seq2V-LSTM model has been effective in extracting features and classifying malware's abnormal behaviours, outperforming other studies. In terms of training and testing time, the Seq2Vec-LSTM model took more time than all other models.

## V. CONCLUSION

Detecting malware on Endpoints is a challenging task. This paper proposes an approach for detecting malware on Endpoints based on the abnormal behaviours of Event IDs using deep learning. Our new proposal in this study has demonstrated superiority, outperforming other methods on the same experimental dataset. This suggests that detecting malware based on Event IDs within the operating system kernel is a reasonable and accurate approach. Additionally, the proposal of using the Seq2Vec model for the task of synthesising and extracting malware features based on Event IDs has achieved high efficiency. This model has successfully standardised malware behaviours to help the malware identification system become more efficient. In the future, to improve the efficiency of the malware detection process on Endpoints, the authors propose 2 improved methods: i) find ways to build relationships between Event IDs, and ii) propose new embedding methods to standardise malware's features.

## DECLARATION

| | |
|---|---|
| Funding/ Grants/ Financial Support | No, we did not receive. |
| Conflicts of Interest/ Competing Interests | No conflicts of interest to the best of our knowledge. |
| Ethical Approval and Consent to Participate | No, the article does not require ethical approval or consent to participate, as it presents evidence. |
| Availability of Data and Material/ Data Access Statement | Not relevant. |
| Authors Contributions | All authors have equal participation in this article. |

## REFERENCES


1. Yanfang Ye, Tao Li, Donald Adjeroh, S. Sitharama Iyengar, *A survey on malware detection using data mining techniques*, ACM Comput. Surv, **50**, 2017. [CrossRef]
2. Daniel Gibert, Carles Mateu, Jordi Planes, *The rise of machine learning for detection and classification of malware: Research developments, trends and challenges*, Journal of Network and Computer Applications, **153**, pp. 1-22, 2020. [CrossRef]
3. Ucci, Daniele & Aniello, Leonardo, *Survey on the Usage of Machine Learning Techniques for Malware Analysis*, Computers & Security, **81**, 2017. [CrossRef]
4. Sanjay Sharma, C. Rama Krishna, Sanjay K. Sahay, *Detection of Advanced Malware by Machine Learning Techniques*, 2019. arXiv:1903.02966. [CrossRef]
5. Alireza Souri, Rahil Hosseini, *A state-of-the-art survey of malware detection approaches using data mining techniques*, **8**, no. 3, pp 1-22, 2018. [CrossRef]

6. Important Information Regarding Sandboxie Versions. https://www.sandboxie.com/. (Accessed on 26 August 2020)
7. Zhong Wei, Gu Feng, *A Multi-Level Deep Learning System for Malware Detection*, Expert Systems with Applications, **133**, 2019. [CrossRef]
8. Fei Xiao, Zhaowen Lin, Yi Sun, Yan Ma, *Malware Detection Based on Deep Learning of Behaviour Graphs*, Mathematical Problems in Engineering. [CrossRef]
9. M. Fan, J. Liu, X. Luo et al., *Android malware familial classification and representative sample selection via frequent subgraph analysis*, IEEE Transactions on Information Forensics and Security, **13**, no. 8, pp. 1890–1905, 2018. [CrossRef]
10. Z. Lin, X. Fei, S. Yi, Y. Ma, C.-C. Xing, J. Huang, *A secure encryption-based malware detection system*, KSII Transactions on Internet and Information Systems, **12**, no. 4, pp. 1799–1818, 2018. [CrossRef]
11. B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, *Deep learning for classification of malware system call sequences*, in proceedings of the Australasian Joint Conference on Artificial Intelligence, Lecture Notes in Comput. Sci., pp. 137–149, 2016. [CrossRef]
12. B. S. Abhishek and B. A. Prakash, *Graphs for malware detection: the next frontier*, in proceedings of the 13th International Workshop on Mining and Learning with Graphs (MLG), 2017.
13. Endpoint Detection and Response Solutions Market- https://www.gartner.com/reviews/market/endpoint-detection-and-response-solutions. (Accessed on 26 August 2020).
14. Sysmon v10.42. https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon (Accessed on 26 August 2021).
15. Dhananjay Kimothi, Akshay Soni, Pravesh Biyani, James M. Hogan, *Distributed Representations for Biological Sequence Analysis*. arXiv:1608.05949v2.
16. Quoc V. Le, Tomas Mikolov, *Distributed Representations of Sentences and Documents*. arXiv:1405.4053.
17. Daniel Svozil, Vladimir Kvasnicka, Jiří Pospíchal, *Introduction to multi-layer feed-forward neural networks*, Chemometrics and Intelligent Laboratory Systems, **39**, no. 1, pp. 43-62, 1997 [CrossRef]
18. Keiron O'Shea, Ryan Nash, *An Introduction to Convolutional Neural Networks*. arXiv, arXiv:1511.08458.
19. Sepp Hochreiter, Jürgen Schmidhuber, *Long Short-Term Memory*, Neural Computation, **9**, no. 8, pp. 1735 - 1780, 1997. [CrossRef]
20. Malware hunting with live access to the heart of an incident. https://app.any.run/ (Accessed on 26 August 2021).
21. S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, T. Yagi, *Malware Detection with Deep Neural Network Using Process Behaviour*, in proceedings of 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), pp. 577-582, 2016. [CrossRef]
22. Mehadi Hassen, Mehadi Hassen, *Scalable Function Call Graph-based Malware Classification*, in proceedings of the Seventh ACM Conference on Data and Application Security and Privacy, pp. 239–248, 2017. [CrossRef]

## AUTHOR PROFILE

**Nguyen Duc Viet** Received a PhD in 2016. Currently working at the Institute of Post and Telecommunications Technology. Research area: Radio positioning, Multilateration monitoring system

**Quan Dang-Dinh** is a lecturer at the Faculty of Information Technology, Hanoi University. His research interests include email prioritization, network security and malware detection