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

Vu Ngoc Son [1†],

Information Assurance dept. FPT University, Hanoi, Vietnam

Summary
The attack technique by the malware distribution form is a dangerous, difficult to detect and prevent attack method. Current malware detection studies and proposals are often based on two main methods: using sign sets and analyzing abnormal behaviors 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 behaviors of malware tracked and collected on Endpoints' operating system kernel. The malware detection proposal based on Event IDs is a new research approach that has not been studied and proposed much. To achieve this purpose, this paper proposes to combine different 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

1. Introduction

Two currently commonly used methods for malware detection include the sign-based detection method and the abnormal behavior-based detection method [1, 2, 3, 4, 5]. Detection methods based on anomalous behavior have been highly effective due to their ability to detect new malware types. Behavior-based detection approaches often seek ways to extract anomalous behaviors of malware and then use methods and algorithms to classify malware. However, it can be seen that the common characteristic of these methods is the use of methods to extract signs and behaviors of malware based on sample datasets. These datasets are built based on virtualization 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 only recording behaviors in a certain time, so it will not be possible to fully statistics malware's behavior. 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 solve these problems, this paper proposes a new approach based on analyzing abnormal behaviors of Event IDs. The characteristic of our approach is that instead of using virtualization tools to collect and extract malware's abnormal behaviors, this approach relies on Event IDs generated by the malware as a basis for detecting abnormal signs and behaviors of malware. These Event IDs are then analyzed by using different data mining methods to seek and aggregate malware's abnormal behaviors. Next, the Seq2Vec algorithm is proposed to be used to synthesize and normalize features of Event IDs. Finally, to conclude about the existence of malware in the system, we use deep learning algorithms such as Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), 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 been published yet by any publications.
- Proposing a method to analyze malware's abnormal behaviors based on Event IDs using the Seq2Vec technique. Although the use of the Seq2Vec model to normalize text data is a common problem, when applying this model to normalizing malware data, it is a new problem and has not been studied and applied by many works. Especially, the application of this model to the process of normalizing Event IDs has not been proposed by any research.

2. 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 AutoEncoders (SAEs) deep

Manuscript received December 5, 2023
Manuscript revised December 20, 2023
https://doi.org/10.22937/IJCSNS.2023.23.12.7

learning network. In the experimental section, the authors compared and evaluated the SAEs model with other machine learning and deep learning algorithms. Experimental results showed that the SAEs 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 Neighbor (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. In addition, the report [13] listed some technology solutions for detecting malware on Endpoints (Endpoint Detection & Response) based on rule sets and behaviors. Accordingly, technology solutions include Trend Micro EDR Apex One, Palo Alto Networks Traps, WildFire, Kaspersky EDR, Carbon Black EDR, and Falcon.

3. The Proposed Model Architecture

3.1. The proposed model architecture

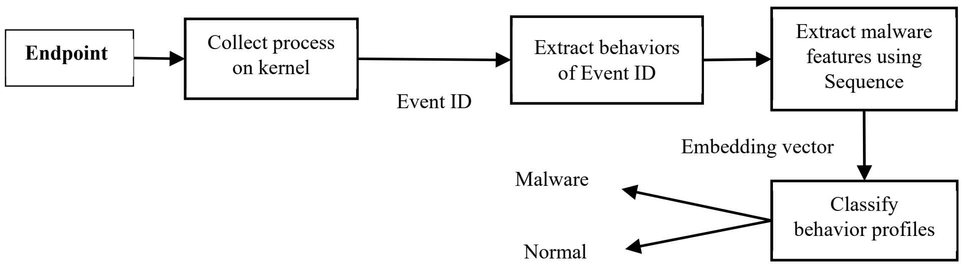
Endpoint
Collect process on kernel
Event ID
Extract behaviors of Event ID
Extract malware features using Sequence
Embedding vector
Classify behavior profiles
Malware
Normal

Figure 1 The architecture of the proposed APT malware detection model

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

1. **Step 1: Collect and process Event IDs on Endpoints.** To perform the task of collecting and extracting these processes, we install and configure the main tool, Sysmon [14]. These tools have the function of collecting the processes recorded by the operating system and transferring these processes to the processing and monitoring center. The processes in Sysmon are described in detail in Table I of Section 2.2.
2. **Step 2: Extract abnormal behaviors of Event IDs.** At this step, abnormal features and behaviors are extracted from the Event IDs collected from the client-side. These features and behaviors are the basis for malware detection. Details of abnormal behaviors of Event IDs are presented in Section 2.3.
3. **Step 3: Extract abnormal behaviors of malware.** As is known, in step 2, the research has extracted anomalous behaviors in Event IDs. Here each file has different characteristics and 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 normalized to a homogenous vector using the Seq2Vec model. Details of this process are described in Section 2.4.
4. **Step 4: Detect malware**. At this step, the malware's behaviors, which are normalized and built in step 3, are classified by a deep learning algorithm to conclude about 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 behaviors 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 behaviors 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, CreateRemoteThread, 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 analyze these Event IDs to collect anomalous behaviors in each Event ID. Table 1 below lists abnormal behaviors found in

Event IDs. These behaviors are the new anomalous behaviors proposed by ours

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

No.	Type	Feature name	Description
1	KEYSTROKES Loggers	GetAsyncKeyState	Poll the state of each keys on the keyboard using the function.
2		GetKeyState	Retrieves the status of the specified virtual key
3		SetWindowsHook	Installs an application-defined hook procedure into a hook chain
4	Network traffic monitor	WSASocket	Create a raw socket
5		socket	Create a raw socket
6		bind	Bind socket to an interface
7		WSAIoctl	Put interface (NIC) in to Promiscuous mode
8		ioctlsocket	Put interface (NIC) in to Promiscuous mode
9	Downloader	URLDownloadToFile	Download file and save to disk
10	Execution	WinExec	Execute file
11		LoadModule	Loads and executes an application or creates a new instance of an existing application.
12		LoadPackagedLibrary	Loads the specified packaged module
13		CreateProcess	Create new process
14		ShellExecute	Execute file
15	HTTP CNC Traffic	InternetOpen	Initializes an application's use of the WinINet functions
16		InternetConnect	Url Input
17		HttpOpenRequest	Build HTTP request
18		HttpAddRequestHeaders	Build HTTP request
19		HTTPSendRequest	Send HTTP Request
20		InternetReadFile	Read Response
21	Droppers	FindResource	Find Resource
22		LoadResource	Retrieves a handle that can be used to obtain a pointer to the first byte of the specified resource in memory.
23		SizeOfResource	Retrieves the size of resource
24		LockResource	Retrieves a pointer to the specified resource in memory.
25	DLL Injection	SetWindowsHook	Install the filter function in the hook chain of the remote processWorks only for GUI application
26		LoadLibrary	Load the malicious DLL into attacking process's address space.
27		GetProcAddress	Retrieve the address of the filter function on the remote process.
28		GetWindowsThread ProcessId	Get ID of Target thread.
29		BroadcastSystemMessage	This is used to send message by attacking process to victim process (internally).
30		VirtualAlloc	Standard windows api call that allows one process to allocate memory space inside another process
31		WriteProcessMemory	Writes data to an area of memory in a specified process
32		GetModuleHandle	Allows the process to determine how to access some dll that might be loaded into the memory space
33		GetProcAddress	Retrieves the address of an exported function or variable from the specified dynamic-link library
34		CreateRemoteThread	Create a remote thread inside a remote process
35	Hooking	GetProcAddress	Locate address of a function to hook
36		VirtualProtect	Set memory protection to read/write
37		ReadProcessMemory	Save first few bytes of victim
38		VirtualProtect	Restore memory permission to original value
39		CreateProcess	Create a process in suspended state.

40	Process hollowing	NtUnmapViewOfSection	Unmap contents of the original process from memory
41		VirtualAlloc	Allocate new memory address in to the hollow process
42		WriteProcessMemory	Brand new code is injected to the hollow process ResumeThread -> Resume the process
43	AntiDebugger /VM detection	GetTickCount	Identify the time to detect a debugger is present
44		CountClipboardFormats	API call to determine whether victim's clipboard was empty
45		GetForeGroundWindow	API call to check if the color of the foreground window was changing,assuming automated sandbox tools doesn't switch active windows around
46		Isdebuggerpresent	Detect debugger
47	ShellCode	GetEIP	Methods SHELLCODE often uses to determine where in memory it is loaded.
48	File and Directory	CreateFile	Creates or opens a file or I/O device
49		OpenFile	Open a file
50		FindFirstFile	Searches a directory for a file or subdirectory with a name that matches a specific name
51		FindNextFile	Continues a file search
52		GetWindowsDirectory	Retrieves the path of the Windows directory.
53		remove	Deletes the file specified by path
54		GetTempPath	Returns the path of the current user's temporary folder
55		DeleteFile	Deletes the file specified by path
56	Registry Keys	RegOpenKey	Opens the specified registry key
57		RegCreateKey	Creates the specified registry key
58		RegSetValue	Sets the data for the default or unnamed value of a specified registry key
59	PowerShell	System	executes an internal operating system command
60		WinExec	Runs the specified application
61	Service	CreateService	Creates a service object and adds it to the specified service control manager database.
62		ControlService	Sends a control code to a service
63		StartServiceCtrlDispatcher	Connects the main thread of a service process to the service control manager
64	Process	CreateProcess	Create new process
65		GetProcessId	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 next process recorded in a system snapshot
68		OpenProcess	Opens an existing local process object

3.4. The method to build malware behavior using Sequence

As is known, each malware has the different number of Event IDs, so it is a difficult task to uniform the length of each file. In this paper, each executable file is considered as a document and each Event as 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 as 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 softmax weights need to be stored instead of both softmax weights and word vectors as PV-DM model. As a result, the Doc2Vec model represents processes into corresponding vectors. Figure 2 below illustrates how to vectorize an executable file using the PV-DBOW model.

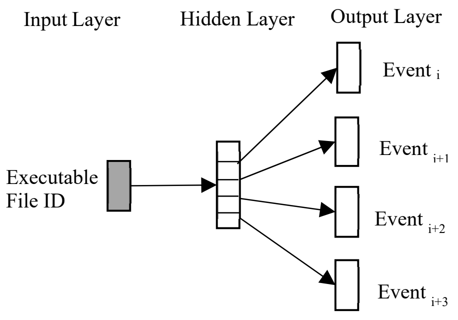

Figure 2 How the PV-DBOW model works for vectorizing an executable file

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 malware's processes are collected, and features are extracted and normalized, we obtain a unique vector representing features of the malware. Next, based on this feature vector, this study evaluates to conclude which are normal files and which are malicious files. This paper uses some deep learning and machine learning algorithms to classify files as normal or malware. Specifically, we propose to use some deep learning algorithms and models: Multilayer Perceptron (MLP), Convolutional Neural Network (CNN), Long short term memory (LSTM), 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 usually have 3 or more layers: 1 input layer, 1 output layer, and more than 1 hidden layer. Besides, the efficiency of the MLP network depends on the activation function. This paper will tune-fine 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 convolution layer + nonlinear layer, fully connected layer. The structure and the terms (stride, padding, MaxPooling) of CNN were presented in detail in the research [18]. In this paper, 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 main components: input gate, forget gate, and 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

4. 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 Agentesla, Azorult, Emotet, Formbook, Gandcrab, Hawkeye, Lokibot, Njrat, Pony, Qbot, Quasar, Remcos, Trickbot, Ursnif, Vidar, etc. 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 different components and then conducts experiments and evaluates the accuracy of the proposed models based on these experimental sub-datasets. The whole process of dividing the experimental dataset for the scenarios is chosen at random in which 80% of the dataset is used for training, 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 conduct the evaluation with the following algorithms: MLP, CNN, LSTM. During the experiment, we tune-fine the parameters to see the effectiveness of the deep learning models. Thus, in this scenario, the paper evaluates 3 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 measures to evaluate the accuracy of models:

1. **Accuracy:** The ratio between the number of samples classified correctly and total number of samples.
2. **Precision:** The ratio between the true positive value and total number of samples classified as positive. The higher value of precision, the more accurate in malicious sample detection.
3. **Recall:** The ratio between the true positive value and the total real malicious samples. The higher value of recall, the lower 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, 4 below

Table 2. Experimental results using 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 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 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 gave different efficiency when changing the parameters of this model. However, this change is not too large because the difference between models is only about 0.1%. Regarding the time variation between models, obviously, when increased the number of hidden layers of the MLP model and the number of features of the Seq2Vec model, the training time increased 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, seeing that the Seq2Vec-CNN model has many similarities with the Seq2Vec-MLP model. Specifically, in terms of training and testing time, when increased the number of layers and features in the model, the training and testing time increased greatly. Besides, regarding the efficiency of the detection process, the models also gave different results when changing the parameters. However, this change is irregular. When the complexity increased, the accuracy did not always increase. Seq2Vec-CNN model had the best results with Accuracy, Precision, Recall, 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 normal file. The best Accuracy detection result is 96.93%. This result is about 0.2% higher than the lowest result. Regarding correctly classifying normal files, this model gave the best results as 94.57% when using 256 features and 4 LSTM layers. In addition, regarding correctly classifying malware, with an efficiency of 89.57%, the Seq2V-LSTM model has shown superiority compared to other models using CNN or MLP. In terms of detection time, obviously, the more complex the model with many LSTM layers and the extension of the 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 showed 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 worked best (with Accuracy, Precision, Recall, and F1-score measures of 96.81%, 94.13%, 89.16%, and 91.58%, respectively) when the RF algorithm uses 100 decision trees and Seq2Vec uses 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 [21]	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, seeing that our proposed Seq2V-LSTM model brought better results than the models proposed in other studies. However, this difference is not too significant. This shows that the Seq2V-LSTM model has worked effectively in the task of extracting features and classifying malware's abnormal behaviors compared to other studies. In terms of training and testing time, the Seq2Vec-LSTM model took more time than all other models.

5. Conclusion

Detecting malware on Endpoints is a difficult and challenging task. This paper proposed an approach for detecting malware on Endpoints based on abnormal behaviors of Event IDs using deep learning. Our new proposal in this study has shown superiority when it gave better performance than other methods on the same experimental dataset. This shows that the approach of detecting malware based on Event IDs on the operating system kernel is reasonable and correct. Besides, the proposal of using the Seq2Vec model for the task of synthesizing and extracting malware's features based on Event IDs has brought high efficiency. This model has successfully standardized malware's behaviors to help the malware identification system to be more efficient. In the future, in order 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, ii) propose new embedding methods to standardize malware's features.

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. https://doi.org/10.1145/3073559.

[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.

[3] Ucci, Daniele & Aniello, Leonardo, *Survey on the Usage of Machine Learning Techniques for Malware Analysis*, Computers & Security, **81**, 2017. https://doi.org/10.1016/j.cose.2018.11.001.

[4] Sanjay Sharma, C. Rama Krishna, Sanjay K. Sahay, *Detection of Advanced Malware by Machine Learning Techniques*, 2019. arXiv:1903.02966.

[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. https://doi.org/10.1186/s13673-018-0125-x.

[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. https://doi.org/10.1016/j.eswa.2019.04.064.

[8] Fei Xiao, Zhaowen Lin, Yi Sun, Yan Ma, *Malware Detection Based on Deep Learning of Behavior Graphs*, Mathematical Problems in Engineering. https://doi.org/10.1155/2019/8195395

[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.

[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.

[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.

[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

[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.

[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 Behavior*, in proceedings of 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), pp. 577-582, 2016. https://doi.org/10.1109/COMPSAC.2016.151

[22] Mehadi Hassen, Mehadi Hassen, *Scalable Function Call Graph-based Malware Classification*, in proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, pp. 239–248, 2017. https://doi.org/10.1145/3029806.3029824.