Computers, Materials & Continua
DOI:10.32604/cmc.2021.014510
Article

Tech Science Press

An Effective Memory Analysis for Malware Detection and Classification

Rami Sihwail*, Khairuddin Omar and Khairul Akram Zainol Ariffin

Department of Information Science & Technology, Universiti Kebangsaan Malaysia, Selangor, 43600, Malaysia
*Corresponding Author: Rami Sihwail. Email: P91206@siswa.ukm.edu.my
Received: 25 September 2020; Accepted: 08 November 2020

Abstract: The study of malware behaviors, over the last years, has received tremendous attention from researchers for the purpose of reducing malware risks. Most of the investigating experiments are performed using either static analysis or behavior analysis. However, recent studies have shown that both analyses are vulnerable to modern malware files that use several techniques to avoid analysis and detection. Therefore, extracted features could be meaningless and a distraction for malware analysts. However, the volatile memory can expose useful information about malware behaviors and characteristics. In addition, memory analysis is capable of detecting unconventional malware, such as in-memory and fileless malware. However, memory features have not been fully utilized yet. Therefore, this work aims to present a new malware detection and classification approach that extracts memory-based features from memory images using memory forensic techniques. The extracted features can expose the malware's real behaviors, such as interacting with the operating system, DLL and process injection, communicating with command and control site, and requesting higher privileges to perform specific tasks. We also applied feature engineering and converted the features to binary vectors before training and testing the classifiers. The experiments show that the proposed approach has a high classification accuracy rate of 98.5% and a false positive rate as low as 1.24% using the SVM classifier. The efficiency of the approach has been evaluated by comparing it with other related works. Also, a new memory-based dataset consisting of 2502 malware files and 966 benign samples forming 8898 features and belonging to six memory types has been created and published online for research purposes.

Keywords: Cybersecurity; feature selection; machine learning; malware dataset; malware detection; memory analysis; memory features

1 Introduction

The number of malware attacks has increased dramatically in the last few years. The attacks have targeted information in many sectors like banks, information technology firms, online government services, and even hospitals. According to the AV-TEST security report, the number of detected malware in the first half of 2020 reached 1075.43 million, while 1001.51 million was

detected in the whole year of 2019 [1]. Many malware files are created due to attackers' automated tools, which produce thousands of malware files daily [2]. According to authors in [3], malware files are used to attack specific targets and trap big data. Therefore, it is recommended to perform malware analysis to detect malware early [4,5].

Today's anti-virus software (AV) commonly use a signature-based technique to detect malware. This technique is fast and can detect known malware with a minimal false-positive rate (FPR). However, the signature-based technique fails to discover unknown malware with no signature, and it is easily defeated by anti-analysis techniques such as evasion, encryption, and packing techniques [3]. Accordingly, it is a requirement for a signature-based technique to update its signature database continuously. In the research field, malware analysis is divided into three major categories: static, behavior, and memory analysis.

In static analysis, malicious files are studied without being executed, and the required features are extracted accordingly. The signature-based has relied on static analysis because it is safe and fast. However, recent malware files use obfuscation techniques, such as the insertion of dead code, register reassignment, the substitution of instruction, and code manipulation to avoid static analysis detection [4]. In contrast, behavior analysis executes and monitors malicious files in a controlled environment. Unlike static analysis, behavior analysis is not vulnerable to obfuscation techniques, but it consumes excessive time and resources. Further, smart malware files may take different actions if placed in an analysis environment, such as behave normally, terminate themselves, or turn into a sleep mode [6]. Furthermore, behavior analysis often shows one side of malware execution because there is no human interaction on malware execution [7]. That means malware activities could not be detected entirely as the rest of the execution paths remained unexposed during malware behavior analysis.

Moreover, malware authors started to utilize technology to develop more sophisticated malware files that are able to evade detection by anti-malware tools. These malware files can attack and compromise a target system without leaving any trace, and it is almost impossible to be reverse-engineered. Two types of unconventional malware files are fileless malware and in-memory malware. Unconventional malware files are challenging to be detected by traditional techniques. However, it is possible to observe their features and study their behaviors in memory [8].

Memory analysis has been proven to be a powerful analysis technique that can effectively study malware behaviors [9]. A considerable amount of information can be found in memory, such as active and terminated processes, Dynamic Link Libraries (DLL), running services, registry, and active network connections. Further, examining memory can detect process/ DLL hooking techniques used by malware to appear as a legitimate process. Additional information can be explored, such as the running operating system and the computer's general state [10]. Furthermore, the analysis provides accurate information about malware behaviors by extracting memory-based features that can express malware activities and characteristics. Memory-based features can also overcome some of the behavior analysis limitations, such as the single view of execution and malware's disguised behaviors during the analysis [11]. Besides, memory analysis has better monitoring of rootkit behaviors, including API hooking, hidden, and process injection [12]. Therefore, a considerable amount of research has chosen memory techniques to handle malware evasion techniques effectively and recommended memory analysis to deep dive into the obfuscated malware [13,14]

It is worth mentioning that malware research lacks reference datasets, where few researchers have shared the datasets used in their works. According to [15], there is no available malware

dataset that can be considered a reference dataset. Therefore, it is common for researchers to download live malware files from AV agents, such as VirusTotal, VirusShare, VX_heaven, and other repositories to perform malware analysis and detection. Although there are some existing datasets like Microsoft big challenge dataset published in 2015, the dataset was created based on static analysis, and it has the same advantages and limitations of the analysis technique [2].

The research proposes a malware analysis and detection approach that focuses on collecting data from memory images (dump). The main contributions of this paper are summarized as follows:

- To propose a memory analysis method that can overcome the limitations of the standard analysis techniques in feature extraction.
- To propose an accurate malware detection and classification approach based on memory analysis.
- To study the new set of features extracted from memory images, such as Application Programming Interface (APIs), DLLs, handles, privileges, networking, and code injection.
- To propose a new benchmark dataset based on six types of memory features. The dataset is published online and available for research purposes.

The rest of the paper is organized as follows: Section 2 presents the related work. The proposed malware detection and classification approach and the memory-based dataset are described in detail in section 3. Section 4 shows experimental results and discussions. Finally, the conclusion and future works are presented in Section 5.

2 Related Work

A considerable amount of research has focused on extracting API call features from program code in the static analysis and use them to detect similar malware [16,17]. Other researchers, such as in [18] and [19], proposed converting API features into images. In [20], Sun et al. proposed a static analysis method to extract opcode sequences from malicious portable executable (PE) files. Static analysis is safely performed, consumes low resources, and can show all possible malware execution views (paths), but today's malware can easily defeat it. Kolosnjaji et al. [21] investigated the vulnerability of static analysis and proposed an attack that could evade detection by modifying a few bytes at the end of each malware Portable Executable (PE) file.

In contrast, behavior analysis is simple, flexible, and robust to code obfuscation [6]. In work by Mohaisen et al. [22], used machine learning techniques to classify the Zeus malware. Several features such as registry, file system, and network features were used to train the classifier. Further, several authors used API calls to represent malware behaviors. In [23], Liang et al. proposed a malware approach based on behavioral analysis that creates a multilayer dependency chain based on measuring the API calls dependency relationships. Their approach calculates the degree of similarity between malware samples. Also, Galal et al. introduced a technique that collects valuable information about the hooked API calls and their used parameters. The method creates sequences for all API calls that share common semantic purposes. The authors used Decision Tree classifier with a reported accuracy of 97.19% [24]. Similarly, Ding et al. [25] proposed an approach for malware detection that relies on extracted API features to represent malware as dependency graphs. However, it is difficult to monitor all possible execution paths during behavior analysis. The analysis time of behavior analysis, based on the literature, is between 60 and 300 seconds. Accordingly, during the analysis, the malware's executed parts only appear in the behavior analysis report [6,26].

Alternatively, memory analysis has attracted several malware researchers. Vömel et al. in [27] surveyed the main memory acquisition and analysis techniques. In [9], Rathnayaka et al. have observed that successful malware infection leaves a memory footprint. Zaki et al. in [28] studied the artifacts left by rootkits in the kernel-level, such as driver, module, SSDT hook, IDT hook, and callback. The experiments proved that certain activities, such as callback functions, modified drivers, and attached devices, are the most suspicious activities at the kernel-level.

In [29], Aghaeikheirabady presented an analysis approach that extracts features available in memory, such as function calls, DLLs, and registry, and compares the information available in different memory structures to increase the accuracy. The approach relies on the frequencies of the extracted features to classify them, and an overall accuracy of 98% is measured by applying Naïve Bayes. However, a significant drawback is the high FPR that exceeded 16%. Similarly, in [30], Mosli et al. introduced a technique that detects malware based on extracting three features from memory images; API calls, registry, and imported libraries. However, the experiments were performed on each feature individually, and maximum accuracy of 96% was achieved using the SVM classifier on the registry activities feature. Afterwards, in their next work [10], Mosli et al. utilized the process handles available in memory to detect malware. The experiment has found that the most handles used by malware are process handles, mutants, and section handles. However, a modest accuracy slightly higher than 91% was achieved by their approach when applying the Random Forest classifier. Likewise, Duan et al. [31] presented an approach to extract live DLL features from memory and employed them to detect malware variants that use the same DLLs. The experimental result showed an accuracy of 90% achieved using the Hidden Naïve Bayes classifier.

Furthermore, Dai et al. [11] proposed a malware detection and classification approach based on extracting memory images and converting them into fixed-size greyscale images. The approach then extracted the features from the images, using a histogram of a gradient, and used them to classify malware. An accuracy of 95.2% was obtained using the neural network (MLP) classifier. Moreover, the authors of this work previously combined API calls from behavior analysis and memory analysis into one vector to represent each sample. A dataset was used, which consisted of 1200 malware and 400 benign files, to train the SVM classifier. The work confirmed that memory analysis could overcome the limitations of behavior analysis [32].

In this paper, new memory features that can reflect malware files' characteristics and expose their hidden behaviors have been extracted from the memory dumped files. The memory dumps were generated during the execution of malware/benign samples in a controlled environment. In the experiments, the effectiveness of the proposed approach on malware detection and classification has been demonstrated and measured by three evaluation metrics: classification accuracy, FPR, and T-Test. To the best of our knowledge, this is the first work that aims to examine such kind of malware behaviors in memory, including a process interaction with the operating system, attempting to gain a higher privilege to perform specific tasks, communicate with an external command and control site, and injecting malicious code into another legitimate process or DLL file. The approach has also utilized the six types of extracted memory-based features to enhance malware detection and classification capabilities. Finally, a benchmark dataset is created and published online for future malware research work.

3 Methodology and Framework

3.1 Malware Detection and Classification Approach

The proposed malware detection and classification approach is divided into five main steps. The first step is to utilize the Cuckoo sandbox to execute the malicious/ benign samples in a controlled environment and generate a memory dump at the end of the sample execution process. In the second step, six memory-based features are extracted from the memory images using the Volatility tool. The extracted features are converted to binary vectors in the dataset at the end of the third step. In the fourth step, two feature selection techniques are applied to remove the redundant and irrelative features from the dataset. The final step is to perform malware classification using some machine learning techniques. The general architecture of the proposed approach is shown in Fig. 1.

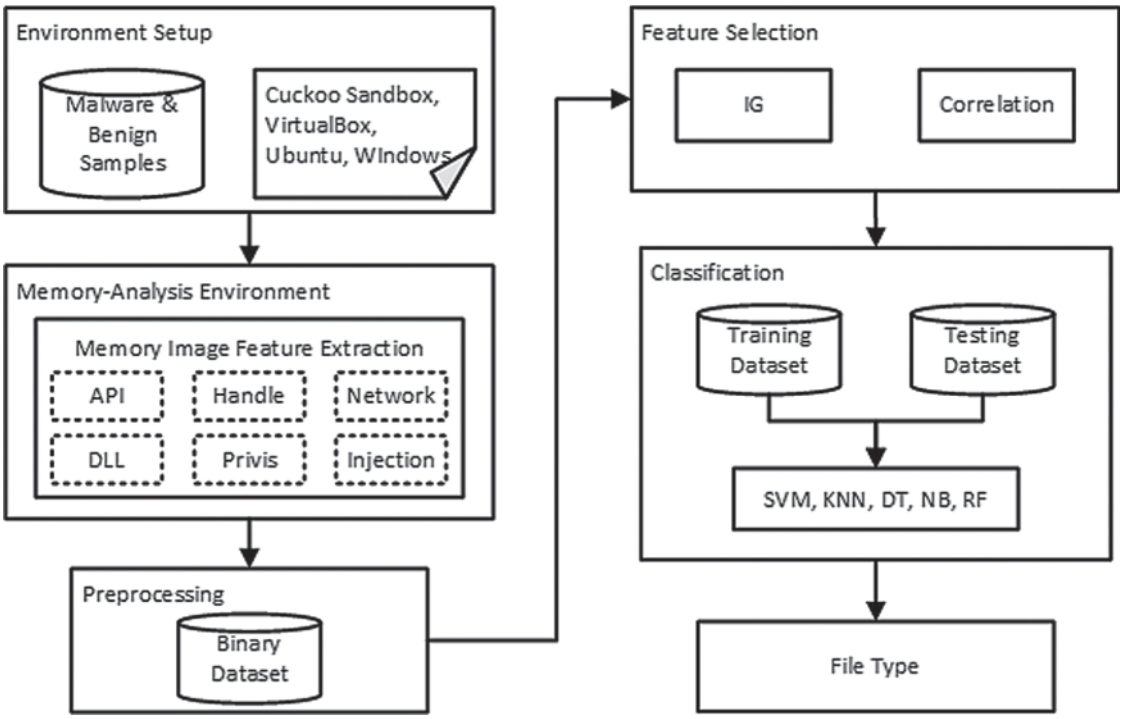

Figure 1: Architecture of malware detection & classification approach

3.2 Sample Collection

For the experiments, we used 2502 malware files and 966 benign files. The malicious files were downloaded from the VirusTotal repository and captured between 2015 and 2019. We targeted different malware across the last five years to include more malicious behaviors in our dataset as malware developers continuously change their tactics and strategies [33]. VirusTotal was used to label the samples based on their families. Almost an equal number of the following malware types were used in the experiment: Backdoor, Keylogger, Downloader, Adware, and Ransomware. In contrast, the benign files were collected manually from the Windows operating system's files and other utility applications.

3.3 Execution Process

In this stage, every malware/benign sample was individually executed in a controlled environment, consisting of VirtualBox version 6.0 running the Windows 7 operating system as a guest machine. The host machine consists of some software, including the Ubuntu 18.04 LTS operating system, the Cuckoo Sandbox, and the Volatility memory forensic tool. The sandbox is a virtual machine environment that monitors the executed sample's behaviors, such as API calls, DLLs, registry operations, and files activities [11]. While a sample executes in the guest machine, it is monitored and analyzed simultaneously by the Cuckoo Sandbox in the host machine. We configured the Cuckoo Sandbox to generate memory dump in the final stage before sandbox execution timeout, when a sample crashes, or before a process terminates itself. The execution timeout was set to 120 seconds for all the experiments. Finally, the memory dumped files are saved to the host machine. At the end of this stage, a memory dumped file is created for each malware and benign sample.

3.4 Feature Extraction

Windows operating systems use Portable Executable (PE) file format for executables and DLLs. PE file stores essential information needed by the operating system loader to manage the wrapped executable code. A PE header is one of the sections that contain vital information, such as DLLs and API calls. However, once the executable is loaded to the memory, the Windows loader loads the required DLLs based on the functions that the application will require to use.

We extracted six types of memory features from each sample's corresponding memory dump using the Volatility tool. Volatility is an open-source memory forensics tool that offers a wide range of plugins for memory analysis tasks. The extracted features aim to reveal malware behaviors. Therefore, six types of memory-based features were extracted as follows:

3.4.1 API Calls Feature

The first selected feature is API calls. Programs use API calls to communicate with the operating system. For example, API calls are used when programs or malware intend to perform specific actions, such as a call for interaction with a file, displaying something on the screen, reading keyboard input, and more. A PE file stores the address of API functions in the IAT table. However, malware can overwrite the API's location in the IAT table to force a process to call an attacker function instead of the original API. Certain well-known malware files hook the IAT table, such as Zeus, FinFisher, and Stuxnet.

In our proposed approach, besides exploring the IAT table, we also searched the memory area, including the kernel that belongs to the corresponding malware, to check which API functions it calls. In other words, the approach does not depend on the IAT table alone to import the API functions due to the high probability that the IAT table may not be reconstructed correctly in the memory image. For example, Coreflood Trojan horse and botnet malware deleted its PE header after it is injected into the target process.

3.4.2 DLL Feature

The second selected feature is the DLL feature. A DLL is an executable file that contains functions called exported functions or exports. Other programs can use the functions by importing them from the DLL. Windows operating system has many DLLs that contain API functions. The malware injects DLLs aiming to insert malicious code into a legitimate process. Once the

malicious DLL is injected, the execution flow is transferred to the malicious memory space [34], as illustrated in Fig. 2.

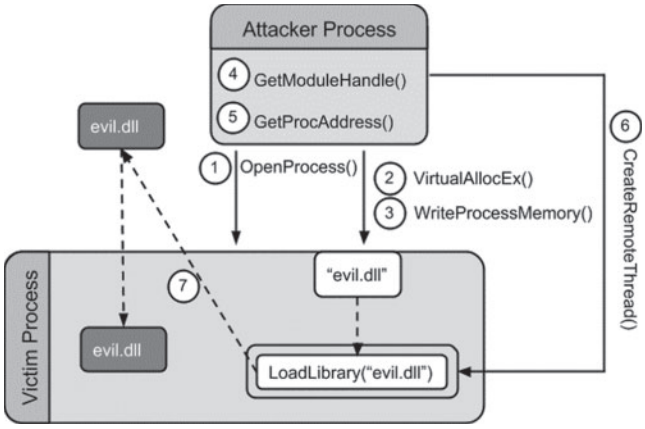

Figure 2: Malware DLL injection technique

Malware file performs DLL injection to transfer the execution flow to a memory space that has been previously occupied by malware. Common DLL injection is performed by malware as follows: the malware first calls the API "VirtualAllocEx" to assign memory into the victim's address space, and then it calls "WriteProcessMemory" to write the DLL path into the allocated memory. After that, "LoadLibrary" is called for the DLL load. Finally, the malware calls function like "CreateRemoteThread," "NtCreateThreadEx" or "RtlCreateUserThread" to create the thread in the target process [35].

The approach extracts the loaded DLLs, by a process, in a memory image by walking through the double link list of "_LDR_DATA_TABLE_ENTRY" structure in memory, which is pointed to by the process environment block (PEB). Basically, all DLLs called by a process are automatically loaded to this list.

3.4.3 Process Handle Feature

The process handle is the third feature in the approach. Michael Hale Ligh defines a handle in [36] as "A handle is a reference to an open instance of a kernel object, such as a file, registry key, process, or thread." There are almost 40 different types of referenced objects that malware can access. Therefore, by examining the memory image to figure the type of handles a malicious process was accessing, it is possible to gather information about malware behaviors and characteristics, such as reading from a file, writing to a file, and accessing a registry, and remote file controlling. In memory, the Object Table in the EPROCESS structure points to a handle table, which contains handles to all open objects accessed by the owner process.

3.4.4 Privilege Feature

The next feature to consider is the privilege. A privilege is the permission-giving to a process to do specific tasks, such as changing the time, loading kernel drive, shutting down the computer, or debugging a process. The approach investigates the examined sample to look for any suspicious request to enable a specific privilege or gain a higher privilege to perform a particular task. The approach treats the privilege features as a Boolean feature. Therefore, any attempt to enable higher privilege is marked as "Yes" by the approach.

3.4.5 Networking Feature

Analyzing Networking features is very important in acquiring information about the attacker. A malicious file needs to communicate with the attacker to receive commands, other malicious files, or send user information or user files. The proposed approach looks for any established foreign connection for TCP and UDP protocols and the connection state. However, as attackers usually use fake or temporary IP addresses, it is not worth considering the extracted IP address. Therefore, the approach marks the existence of an external connection as a network feature rather than the IP address itself.

3.4.6 Code Injection Feature

The last feature in the dataset is code injection. Code injection is one of the popular techniques used in malware attacks to inject malicious code into another legitimate process. The technique forces the legitimate process to run code on its behalf, such as stealing data and information from the system or downloading other malicious files. The main techniques malware files use to inject code are; writing into the memory of a legitimate process and forcing the legitimate process to load malicious DLL by changing the registry key values.

The approach looks for any artifacts in memory that lead to code injection. The approach can also find hidden code injection performed by malware by examining the Virtual Address Descriptor (VAD) tree structure. VAD is used by the Windows memory manager to allocate the process's memory ranges in memory. Basically, an entry is added to the VAD tree structure whenever a process uses the "VirtualAlloc" function call to allocate a new memory region [37]. Therefore, the approach looks for any code injection artifacts by checking the integrity of the entire VAD tree structure. Each of the extracted feature types reveals a different feature of malware behaviors, such as tasks malware performs, a function it uses, information about its behaviors, gaining privileges, communicating with an external source, and others. We believe that the extracted features increase the ability of the approach to identify malware behaviors. Consequently, the approach can extract more representative features that can enhance its ability to detect and classify malware. A summary of the extracted features, quantities, Volatility plugins, and information about the targeted behavior is presented in Tab. 1.

3.5 Dataset

One challenge in malware research is the lack of a referenced malware dataset [15]. To the best of our knowledge, there is no memory-based dataset available for malware researchers. Currently, a few malware datasets have been published that have been created based on either static or behavior analysis. However, the datasets suffer from the same limitations of the analysis technique. A dataset containing weak and irrelated features could negatively impact the training of machine learning classifiers. Besides, it makes the approach easily manipulated by malware evasion techniques. Further, such a dataset would probably reduce the ability to be generalized on detecting new malware samples [11]. As a result, it is typical for researchers to download live malicious files from anti-virus agents, such as VirusTotal, VirusShare, and VX_heaven, when performing malware analysis and classification.

Therefore, one of the research objectives is to create a comprehensive dataset based on memory features. Hence, to make our dataset available for other researchers, we have published the dataset online to be used for research purposes. The dataset can be found on the GitHub platform [https://github.com/sihwail/malware-memory-dataset].

Table 1: Memory features and Volatility commands

sFeature	Quantity	Command	Description
API	5567	Impscan	Malware use API calls to communicate with the operating system.
DLL	3271	Dlllist	Malware inject DLLs aiming to insert malicious code into a legitimate process.
Process Handle	29	Handles	Give information about malware behaviors, such as reading from a file, writing to a file, and accessing a registry key.
Privilege	29	Privs	Permission is giving to a process to do specific tasks, like changing the time, loading kernel drive, and shutting down the computer.
Network	Yes/No	Netscan	It is used to communicate with the attacker to receive commands or send user information.
Code Injection	Yes/No	Malfind	The malware attempts to inject its malicious code into another legitimate process to force the latest to run code on its behalf.

The first row in the dataset indicates the feature name starting with API features, followed by DLL and the rest of the features. However, starting from the second row onwards, each row represents one sample, either malware or benign, as a binary vector where (1) indicates the fact that the feature in the first row exists in the sample and (0) if it is not. Further, samples are labeled in the last column with the letter 'B' for benign and 'm' for malware. An example of the dataset is shown in Fig. 3.

AbortDoc	*AcceptEx*	*AddAce*	*AddAtomW*	*AddFormW*	*Arc*	*ArcTo*	*HNDL-Timer*	*HNDL-Token*	...	*Netscan*	*Malfind*	*Label*
0	1	0	0	0	1	0	0	0		0	0	m
0	0	0	0	1	0	0	1	1		1	0	m
1	0	1	0	0	0	1	0	0		1	0	m
0	1	0	1	0	1	1	0	0		1	1	m
						...						
1	0	1	0	1	1	0	1	1		0	0	B
0	0	1	1	0	0	1	1	0		0	1	B
0	1	0	0	0	0	0	0	1		1	0	B

Figure 3: A sample of the created dataset. The first row represents the attributes, while the rest of the rows represent the local vectors

To distinguish between feature types, a prefix is added to the first row features in the dataset as follows: MRY to API features, HNDL for handle features, and PRIV for privilege features. For

example, an API feature "CreateFileW" is saved as "MRY-CreateFileW" in the dataset. We did not add a prefix for DLL features as the feature name ends with the extension (.dll). Finally, there is only one Network feature, namely Netscan, and one code injection feature, namely Malfind, following the Volatility command names. A summary of the dataset is presented in Tab. 2.

Table 2: A general description of the created dataset

Dataset name	# Malware	# Benign	# Classes	# Malware Families
Memory- Malware	2502	966	2	5
# Total Features	# API	# DLL	# Handles	# Privilege
8898	5567	3271	29	29
# Networking	# Code Injection	Dataset URL		
1	1	https://github.com/sihwail/malware-memory-dataset		

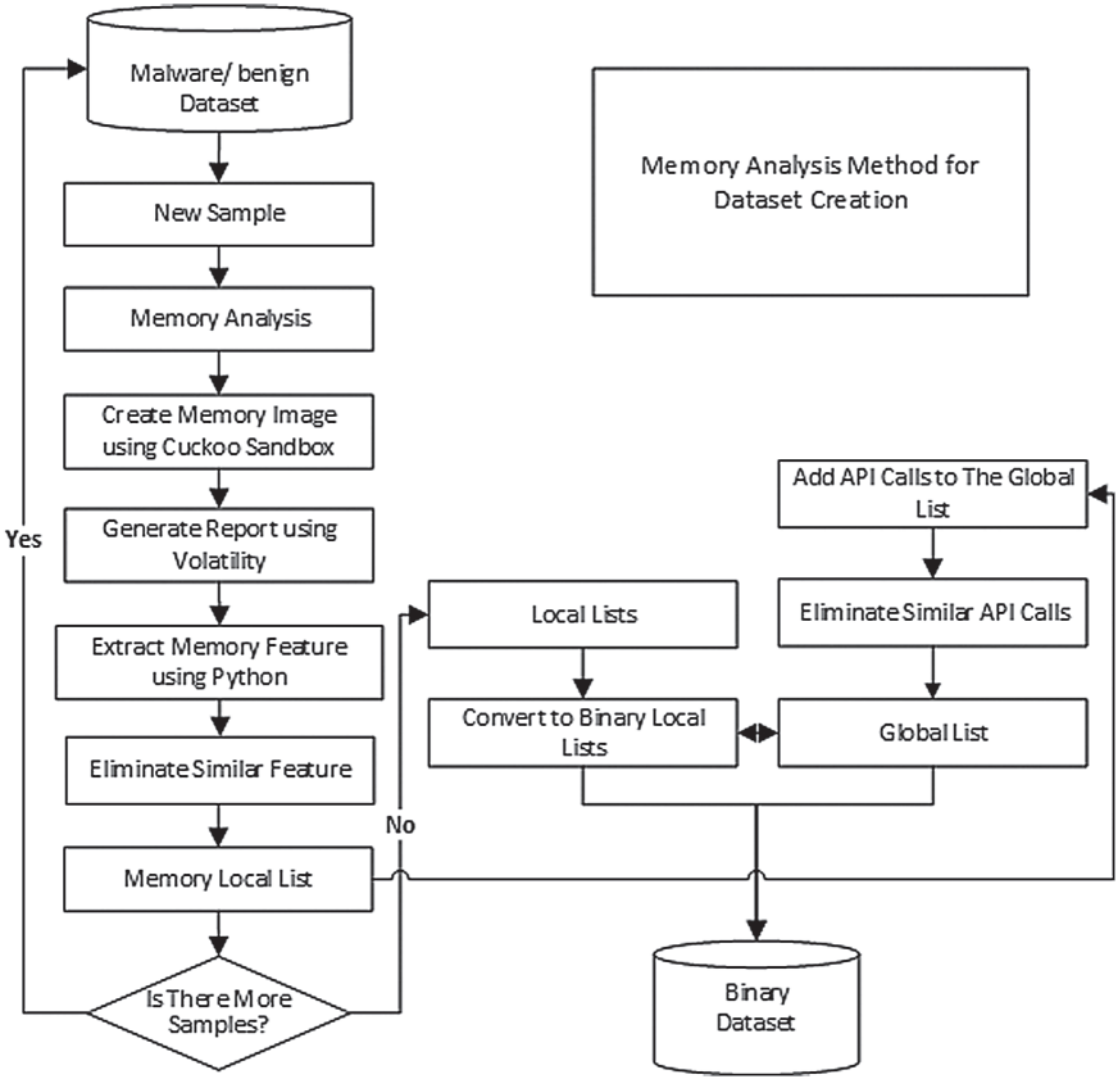

Figure 4: The general flowchart for extracting memory features to create the dataset

The flowchart for extracting memory features to create the binary dataset has passed through several steps, as illustrated in Fig. 4.

3.6 Feature Selection

In machine learning, the domain of features (attributes) have expanded from tens to thousands in the last few years. High dimensionality features have posed a severe challenge for machine learning classifiers. Feature selection helps create an accurate predictive model by selecting features that maintain good accuracy or improve it [38]. Feature selection methods are used to eliminate irrelevant, unnecessary, and redundant features [39]. However, many features create more challenges to the feature selection process because the number of features is exponentially related to the search domain's size. For example, when the number of attributes (domains) in the dataset is n attributes, there exist 2^n solutions [40].

Feature selection was used for two reasons; measure the features' effectiveness by calculating the features' weights and decrease the approach's time complexity by reducing the number of selected features used in the classification process.

4 Results and Discussion

This research aims to identify malware behaviors through close observation of their features in memory. The effectiveness of memory-based features in malware detection and classification has been validated by conducting two experiments. In the first experiment, we combined the six types of memory-based features. While in the second one, we evaluated each type individually.

The dataset was split into ten groups using 10-fold cross-validation and, therefore, the classifier is trained on data that is completely separated from the data used in the testing phase. The following measures are used to evaluate the results:

True Positive (TP): Number of positive examples that have been correctly identified.

False Positive (FP): Number of negative examples that have been incorrectly identified.

False Negative (FN): Number positive examples that have been incorrectly identified.

True Negative (TN): Number of negative examples that have been correctly identified.

$$\text{False Positive Rate (FPR)} = \frac{\text{F.P.}}{\text{FP} + \text{TN}} \tag{1}$$

$$\text{True Positive rate (Recall)} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2}$$

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4}$$

$$\text{F-Score} = \frac{2 * (\text{Precision} * \text{Recall})}{\text{Precision} + \text{Recall}} \tag{5}$$

4.1 All-In-One Features

In this experiment, all memory features, which consist of six feature types, were placed together into one global dataset used in training, testing, and evaluating the classifiers. Five

classifiers were involved, and the results of classifying six types with a total of 8988 features extracted from 3468 samples are displayed in Tab. 3.

Table 3: Performance of classifying memory features

Classifier	Recall	FPR	F-Score	Accuracy
Naïve Bayes	93.17%	1.72%	95.28%	96.86%
SVM	96.75%	1.24%	96.39%	98.50%
KNN (K=5)	92.86%	1.88%	94.74%	96.65%
Decision Tree	88.41%	1.56%	94.34%	96.37%
Random Forest	87.68%	2.04%	94.12%	95.94%

In the experiment, the results of measuring the accuracy and FPR of the classifiers were satisfied except for the decision tree, as shown in Tab. 3. However, the highest accuracy equals to 98.5% gained by applying the SVM classifier. The SVM achieved the lowest FPR as well, with 1.24% only. With accuracy equals 96.86% and FPR slightly higher than 1.7%, the Naïve Bayes came second in a row. The KNN classifier (K=5) came in third place with 96.65% and 1.88% accuracy and FPR, respectively.

We also calculated the weights of the features to determine the importance of features in the dataset. The weight was calculated based on information gain (IG) and Correlation feature selection techniques. Tab. 4 shows the 12 highest weighted features based on the two techniques.

Table 4: Weight calculation based on IG and Correlation techniques

Feature Name	Type	IG	Correlation
comctl32.dll	DLL	0.357	0.677
rasadhlp.dll	DLL	0.334	0.620
urlmon.dll	DLL	0.316	0.677
iertutil.dll	DLL	0.315	0.661
WININET.dll	DLL	0.285	0.623
ieframe.dll	DLL	0.284	0.584
NLAapi.dll	DLL	0.280	–
Malfind	Injection	0.279	0.621
mswsock.dll	DLL	0.266	–
HNDL-ALPC	Handle	0.266	0.607
HNDL-ALPC Port	Handle	–	0.607
HNDL-Port	Handle	–	0.607

The features and their frequencies are illustrated in Fig. 5. The most-weighted features are not necessary to be the most frequent features used by malicious or benign files. However, they are most likely the features that can distinguish between the two classes in the machine learning classification process.

Note that most of the features in Tab. 4 belong to the DLL features type. The DLL feature comctl32.dll has the highest score based on both IG and correlation weight-scale techniques, and

it was found in 2189 malware samples and 102 benign files. Two handle-features have excellent weight based on the correlation technique but less critical based on the IG technique. Finally, the injection feature Malfind has a significant weight based on both techniques.

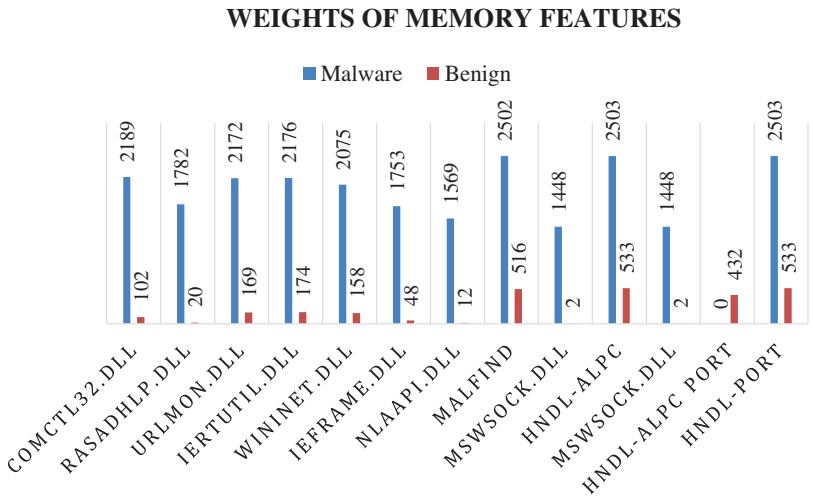

Figure 5: Distribution of the most weighted features over malware and benign samples

The DLL files are typically useful and contain functions that can support running programs and operating systems. For example, the comctl32.dll previously mentioned is responsible for dialog box functions, such as an open dialog box. However, based on the experiment, malware files exploit the DLL files in many ways:

- Downloading the DLL file that has the same name as a legitimate file but in a different folder.
- Downloading the DLL file that has almost a similar name as a legitimate one.
- Injecting DLL file and force it to perform malicious activities.

The results show that using feature selection did not significantly improve SVM and KNN classification accuracy on the dataset. However, a good accuracy of 97.35% achieved by selecting 1000 features using the IG technique and the SVM classifier. In contrast, a slight increase to 96.91% was achieved for KNN accuracy using 500 features selected based on the IG technique. On the other hand, the execution time is a crucial factor in measuring the approach's complexity. It is noticeable that the time is considerably decreased in both classifiers when reducing the number of selected features. The accuracy is also dropping down when the number of selected features is equals or less than 300. The relation between classification accuracy and the number of selected features is illustrated in Fig. 6.

Applying feature selection techniques helps create an accurate and predictive model by selecting features that maintain good accuracy or even improve it. It works by eliminating irrelevant, unnecessary, and redundant features in the dataset. Therefore, we applied IG and correlation feature selection and classified the outcomes using SVM and KNN classifiers. The classification accuracy and the consumed time are measured as presented in Tab. 5.

Several researchers applied memory analysis to extract features that are useful in detecting malware. However, the related work approaches focused on specific features like API call, DLL,

and registry. In contrast, our approach is more comprehensive as it focuses on studying malware behaviors in memory rather than collecting the available features. Besides extracting the standard memory features, such as API and DLL, our approach examines other features that can reveal hidden malware behaviors, including injecting DLLs and other processes, communicating with external sources, and attempting to gain higher privileges. A comparison with the related works is shown in Tab. 6.

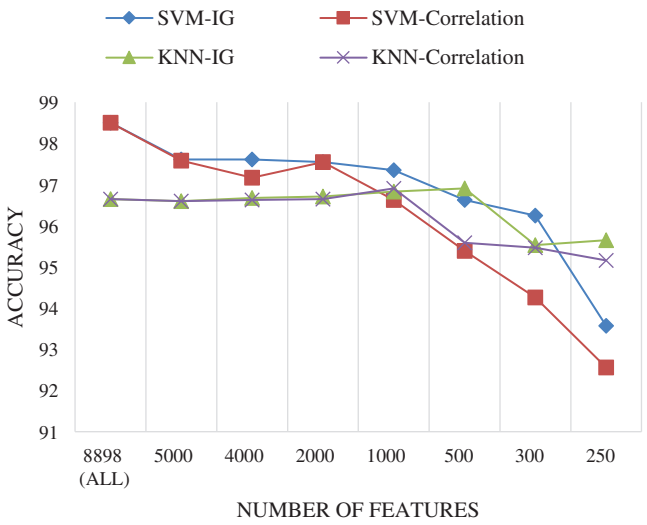

Figure 6: The relation between classification accuracy and feature selection for All-in-One features

Table 5: Accuracy evaluation using feature selection

Feature Number	SVM			KNN		
	IG	Correlation	Time (Sec.)	IG	Correlation	Time (Sec.)
8898 (All)	98.50%	98.50%	180	98.50%	98.50%	110
5000	97.61%	97.58%	95	96.60%	96.60%	70
4000	97.61%	97.17%	82	96.68%	96.63%	58
2000	97.55%	97.55%	66	96.71%	96.65%	35
1000	97.35%	96.63%	54	96.83%	96.91%	20
500	96.63%	95.39%	32	96.91%	95.59%	16
300	96.25%	94.26%	22	95.53%	95.47%	13
250	93.57%	92.56%	22	95.65%	95.16%	6

The proposed approach has outperformed the other related works by achieving the highest accuracy and lowest FPR with 98.5% and 1.24%, respectively. Aghaeikheirabady's approach focused on extracting API and DLL features from memory and achieved 98% accuracy, which is only 0.5 less than our approach. However, the approach suffers from very high FPR, which came as high as 16%.

Mosli introduced two approaches. While the first approach extracted API call, registry, and DLL features and used each feature individually to train the classifiers, the second approach

relied on the used process handles to detect and classify malware. Similarly, Duan's approach depended on a single type of feature, DLL, or API features. Lastly, Dai's approach extracted information about processes from memory images such as API sequence, disk reading, and registry modification. A comparison with the related works in terms of classification accuracy and FPR is presented in Fig. 7.

Table 6: Comparing experimental results with memory-based related work

Author/year	Extracted features	Malware/benign	Accuracy	FPR
Aghaei./2014	DLL, API calls, Registry	350/200	98%	16%
Duan/2015	DLL	3,390/230	90%	6%
Mosli/2016	Registry, DLLs, API calls	400/100	96%	5%
Mosli/2017	process handles	3,130/1,157	91.4%	5%
Dai/2018	Process info. (registry, API)	1984/-	95.2%	5%
Proposed method/2020	API, DLL, handles, privileges, network, code injection	2,502/966	98.5%	1.24%

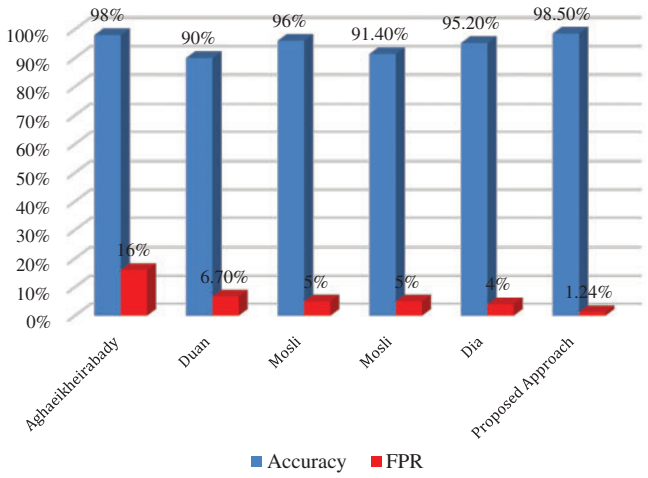

Figure 7: A comparison between the proposed method and the related work methods

We also performed the unpaired t-Test as a statistical test. We compared our approach against the other memory-based approaches in the related work. With P-values less than 0.05 for classification and FPR, the results have confirmed that the proposed detection approach has a significant contribution. The P-values results are shown in Tab. 7. In the next subsection, we studied feature's effectiveness and the weight of the prominent features on malware detection and classification processes.

4.2 Individual Features

This section analyzes the six types of features individually and measures their effectiveness on the classification process. Five classifiers were used to evaluate the accuracy and FPR for the feature types. While the accuracy is shown in Tab. 8, calculating the FPR is displayed in Tab. 9.

Table 7: Unpaired T-test

Value	Description
Accuracy	0.040258221
FPR	0.037247124

Table 8: Accuracy evaluation for individual feature type

Feature type	Number of features	Naïve Bayes	SVM	KNN	Decision tree	Random forest
API	5567	93.02%	93.74%	93.83%	89.50%	80.16%
DLL	3271	88.41%	92.39%	95.79%	92.56%	84.54%
Process Handle	29	85.24%	84.63%	88.93%	90.71%	89.85%
Privilege	29	72.09%	77.31%	73.24%	78.40%	78.14%
Network	1	66.81%	72.15%	72.15%	72.15%	72.15%
Code Injection	1	85.12%	85.12%	85.12%	85.12%	85.12%

Table 9: FPR evaluation for individual feature type

Feature type	Number of features	Naïve Bayes	SVM	KNN	Decision tree	Random forest
API	5567	1.24%	5.23%	1.84%	0.72%	0.04%
DLL	3271	15.91%	10.35%	2.04%	8.19%	0.20%
Process Handle	29	2.16%	9.03%	0.48%	2.28%	2.48%
Privilege	29	2.12%	11.47%	0.08%	11.87%	11.83%
Network	1	44.80%	–	–	–	–
Code Injection	1	–	–	–	–	–

From the last two tables, Tabs. 8 and 9, it can be noticed that the DLL feature has the highest accuracy rate among other memory-based features with 95.8%, followed by API and process handles features, 93.8%, and 90.7%, respectively. However, none of the memory features could individually reach the highest accuracy of 98.5%, which was achieved based on classifying all memory features together (all-in-one). Therefore, joining the features has added more value to the classification process, which is logically justified as adding more relevant features to the detection approach has enhanced its capabilities in distinguishing between the two classes. Further, the proposed approach has been proved to be able to identify different activities performed by malware, such as interacting maliciously with the operating system, attempting to inject malicious code into another process or DLL file, communicating with a suspicious external source, or trying to gain a higher privilege to perform a specific task. Furthermore, based on an intensive study of malware behaviors in memory, the proposed approach has utilized the six types of memory-based features to improve its ability to detect and classifying malware. Lastly, we calculated the features' weights based on IG and correlation techniques to display prominent features. The features and frequencies for the malware and benign samples are demonstrated in Fig. 8.

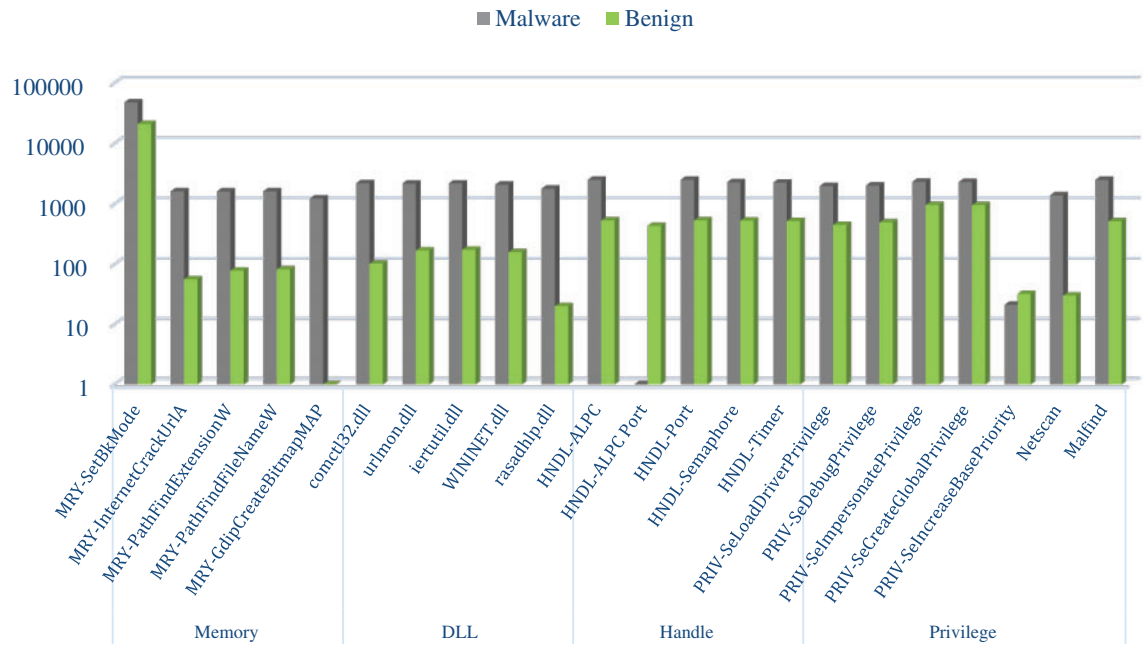

Figure 8: Distribution of the most weighted memory features over malware and benign samples

5 Conclusion

This paper proposed an approach that can detect and classify malware using memory-based features extracted from memory images. The approach represents malware behaviors and characteristics using six feature types; API calls, DLLs, process handles, privileges, network, and code injection. Memory features were extracted using the Volatility memory forensic tool. We have performed two experiments to evaluate the effectiveness of memory features. First, we combined the six features into one global vector and measured its performance in classifying malware samples. Second, memory features were evaluated individually, and the weight of each feature was calculated. The results showed that the proposed approach outperformed the related memory-based approaches with significant improvements in classification accuracy and FPR using the SVM classifier, 98.5% and 1.24, respectively.

We have also investigated the impact of reducing the number of selected features on classifier performance. The most weighted features were calculated using two feature selection techniques: Information Gain (IG) and Correlation. Although the accuracy was reduced when applying the feature selection techniques, the approach maintains a reasonable accuracy rate using 500 features selected by the IG technique. Further, it has been noticed that DLL features have the highest weights comparing to the other memory features.

Finally, we created a new dataset that consists of six types of memory-based features. It is the first memory-based dataset to be created and made available online for research purposes to the best of our knowledge. The dataset consists of 2502 malware and 966 benign samples forming a total of 8898 features. It includes nearly equal entries of Ransomware, Keylogger, Adware, Downloader, and Backdoor malware families.

The approach's limitation is time complexity due to the large number of features used in the classification process. Hence, a memory agent that can reduce the time is recommended. For future

work, we will investigate more possible memory features to increase the accuracy of the approach. In addition, memory dumps can be taken more than once during the execution based on process transactions to capture memory changes in different execution stages.

Funding Statement: This work was supported in part by Universiti Kebangsaan Malaysia (UKM) under Grant GUP-2019-062 and Grant GP-2019-K005539, and in part by the Ministry of Education Malaysia under Grant FRGS/1/2018/ICT04/UKM/02/3.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] AV-Test, "The AV-test security report," 2020. [Online]. Available: https://www.av-test.org/.

[2] R. Sihwail, K. Omar and K. A. Z. Ariffin, "A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 8, no. 4–2, pp. 1662–1671, 2018.

[3] A. Khanan, S. Abdullah, A. H. H. M. Mohamed, A. Mehmood and K. A. Z. Ariffin, "Big data security and privacy concerns: A review," in *Smart Technologies and Innovation for a Sustainable Future, Advances in Science, Technology & Innovation*, Cham: Springer, pp. 55–61, 2019.

[4] Y. Ye, T. Li, D. Adjeroh and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Compuing Survey*, vol. 50, no. 3, pp. 1–40, 2017.

[5] A. Mehmood, A. Khanan, M. M. Umar, S. Abdullah, K. A. Z. Ariffin *et al.,* " Secure knowledge and cluster-based intrusion detection mechanism for smart wireless sensor networks," *IEEE Access*, vol. 6, pp. 5688–5694, 2017.

[6] D. Sgandurra, L. Muñoz-González, R. Mohsen and E. C. Lupu, "Automated dynamic analysis of ransomware: Benefits, limitations and use for detection," arXiv preprint arXiv: 1609.03020, 2016.

[7] A. Afianian, S. Niksefat, B. Sadeghiyan and D. Baptiste, "Malware dynamic analysis evasion techniques: A survey," arXiv preprint arXiv: 1811.01190, 2018.

[8] S. Saad, F. Mahmood, W. Briguglio and H. Elmiligi, "JSLess: A tale of a fileless javascript memory-resident malware," in *Int. Conf. on Information Security Practice and Experience*, Ithaca, New York: Cornell University, pp. 113–131, 2019.

[9] C. Rathnayaka and A. Jamdagni, "An efficient approach for advanced malware analysis using memory forensic technique," in *2017 IEEE Trustcom/BigDataSE/ICESS*, Cham: Springer, pp. 1145–1150, 2017.

[10] R. Mosli, R. Li, B. Yuan and Y. Pan, "A behavior-based approach for malware detection," in *IFIP Advances in Information and Communication Technology*. Cham: Springer, pp. 187–201, 2017.

[11] Y. Dai, H. Li, Y. Qian and X. Lu, "A malware classification method based on memory dump grayscale image," *Digital Investigation*, vol. 27, pp. 30–37, 2018.

[12] C. W. Tien, J. W. Liao, S. C. Chang and S. Y. Kuo, "Memory forensics using virtual machine introspection for malware analysis," in *2017 IEEE Conf. on Dependable and Secure Computing*, Taipei, Taiwan, pp. 518–519, 2017.

[13] A. Case and G. G. Richard, "Memory forensics: The path forward," *Digital Investigation*, vol. 20, pp. 23–33, 2017.

[14] K. W. P. Choi, "Sang-Hoon and Yu-Seong Kim, Toward semantic gap-less memory dump for malware analysis," in In ICNGC Conference, Bangkok, Thailand, 2016.

[15] D. Ucci, L. Aniello and R. Baldoni, "Survey on the usage of machine learning techniques for malware analysis," *Computers & Security*, vol. 81, pp. 123–147, 2019.

[16] Z. Salehi, A. Sami and M. Ghiasi, "Using feature generation from API calls for malware detection," *Computer Fraud & Security*, vol. 2014, no. 9, pp. 9–18, 2014.

[17] Y. Cheng, W. Fan, W. Huang and J. An, "A shellcode detection method based on full native API sequence and support vector machine," *IOP Conference Series: Materials Science and Engineering*, Changsha, China, vol. 242, no. 1, pp. 12124, 2017.

[18] H. Hashemi and A. Hamzeh, "Visual malware detection using local malicious pattern," *Journal of Computer Virology and Hacking Techniques*, vol. 15, no. 1, pp. 1–14, 2019.

[19] S. Z. M. Shaid and M. A. Maarof, "Malware behaviour visualization," *Jurnal Teknologi*, vol. 70, no. 5, pp. 25–33, 2014.

[20] Z. Sun, Z. Rao, J. Chen, R. Xu, D. He *et al.,* "An OpCODE sequences analysis method for unknown malware detection," in *ACM Int. Conf. Proceeding Series*, Prague Czech Republic, pp. 15–19, 2019.

[21] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto *et al.,* "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *2018 26th European Signal Processing Conf.*, pp. 533–537, 2018.

[22] A. Mohaisen and O. Alrawi, "Unveiling Zeus: Automated classification of malware samples," in *Proceedings of the 22nd Int. Conf. on World Wide Web companion*, Rio de Janeiro Brazil, pp. 829–832, 2013.

[23] G. Liang, J. Pang and C. Dai, "A behavior-based malware variant classification technique," *International Journal of Information and Education Technology*, vol. 6, no. 4, pp. 291–295, 2016.

[24] H. S. Galal, Y. B. Mahdy and M. A. Atiea, "Behavior-based features model for malware detection," *Journal of Computing Virology and Hacking Techniques*, vol. 12, no. 2, pp. 59–67, 2016.

[25] Y. Ding, X. Xia, S. Chen and Y. Li, "A malware detection method based on family behavior graph," *Computers & Security*, vol. 73, pp. 73–86, 2018.

[26] T. Teller and A. Hayon, "Enhancing automated malware analysis machines with memory analysis report," Black Hat USA, 2014. [Online]. Available: https://www.blackhat.com/docs/us-14/materials/arsenal/us-14-Teller-Automated-Memory-Analysis-WP.pdf.

[27] S. Vömel and F. C. Freiling, "A survey of main memory acquisition and analysis techniques for the windows operating system," *Digital Investigation*, vol. 8, no. 1, pp. 3–22, 2011.

[28] A. Zaki and B. Humphrey, "Unveiling the kernel: Rootkit discovery using selective automated kernel memory differencing," in *Proc. of the 2014 Virus Bulletin Conf.*, Seattle, Washington, USA, pp. 239–256, 2014.

[29] M. Aghaeikheirabady, S. Farshchi and H. Shirazi, "A new approach to malware detection by comparative analysis of data structures in a memory image," in *International Congress on Technology, Communication and Knowledge*, pp. 1–4, 2014, 2014

[30] R. Mosli, R. Li, B. Yuan and Y. Pan, "Automated malware detection using artifacts in forensic memory images," in *2016 IEEE Symposium on Technologies for Homeland Security, HST 2016*, Waltham, Massachusetts, USA, pp. 1–6, 2016.

[31] Y. Duan, X. Fu, B. Luo, Z. Wang and J. Shi *et al.,* "Detective: Automatically identify and analyze malware processes in forensic scenarios via DLLs," *IEEE Int. Conf. on Communications*, London, UK, vol. 2015, pp. 5691–5696, 2015.

[32] R. Sihwail, K. Omar, K. Zainol Ariffin and S. Al Afghani, "Malware detection approach based on artifacts in memory image and dynamic analysis," *Applied Sciences*, vol. 9, no. 18, pp. 3680, 2019.

[33] R. Islam, R. Tian, L. M. Batten and S. Versteeg, "Classification of malware based on integrated static and dynamic features," *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 646–656, 2013.

[34] S. Kim, J. Park, K. Lee, I. You and K. Yim, "A brief survey on rootkit techniques in malicious codes," *Journal of Internet Services and Information Security*, vol. 3, no. 4, pp. 134–147, 2012.

[35] A. Hosseini, "Ten process injection techniques: A technical survey of common and trending process injection techniques," 2017. [Online]. Available: https://www.endgame.com/blog/technical-blog/ten-process-injection-techniques-technical-survey-common-and-trending-process.

[36] A. Ligh, M. Hale, A. Case, J. Levy, Walters *et al.,* "Processes, handles, and tokens," in *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and MAC Memory*. 1st ed., vol. 1. NJ, USA: John Wiley & Sons, pp. 149–188, 2014.

[37] B. Dolan-Gavitt, "The VAD tree: A process-eye view of physical memory," *Digital Investigation*, vol. 4, pp. 62–64, 2007.

[38] R. Tian, "An integrated malware detection and classification system," Ph. D. dissertation. Deakin University, Australia, 2011.

[39] A. Feizollah, N. B. Anuar, R. Salleh and A. W. A. Wahab, "A review on feature selection in mobile malware detection," *Digital Investigation*, vol. 13, pp. 22–37, 2015.

[40] R. Sihwail, K. Omar, K. Akram and Z. Ariffin, "Improved Harris hawks optimization using elite opposition-based learning and novel search mechanism for feature selection," *IEEE Access*, vol. 8, pp. 121127–121145, 2020.