applied sciences

MDPI

Article

Malware Detection Approach Based on Artifacts in Memory Image and Dynamic Analysis

Rami Sihwail [1,*], Khairuddin Omar [2], Khairul Akram Zainol Ariffin [2] and Sanad Al Afghani [1]

1 Supporting Studies Center, King Faisal University, Al Hasa 31982, Saudi Arabia
2 Faculty of Information Science & Technology, Universiti Kebangsaan Malaysia, Bangi 43600, Selangor, Malaysia
* Correspondence: rsihwail@kfu.edu.sa

Received: 21 July 2019; Accepted: 2 September 2019; Published: 5 September 2019

Abstract: The need to detect malware before it harms computers, mobile phones and other electronic devices has caught the attention of researchers and the anti-malware industry for many years. To protect users from malware attacks, anti-virus software products are downloaded on the computer. The anti-virus mainly uses signature-based techniques to detect malware. However, this technique fails to detect malware that uses packing, encryption or obfuscation techniques. It also fails to detect unseen (new) ones. This paper proposes an integrated malware detection approach that applies memory forensics to extract malicious artifacts from memory and combines them to features extracted during the execution of malware in a dynamic analysis. Pre-modeling techniques were also applied for feature engineering before training and testing the data set on the machine learning models. The experimental results show a significant improvement in both detection accuracy rate and false positive rate, 98.5% and 1.7% respectively, by applying the support vector machine. The results verify that our integrated analysis approach outperforms other analysis methods. In addition, the proposed approach overcomes the limitation of single path file execution in dynamic analysis by adding more relevant memory artifacts that can reveal the real intention of malicious files.

Keywords: malware; dynamic analysis; artifacts; memory analysis; feature engineering; machine learning

1. Introduction

The threat of malware attacks on computers, mobile phones and other devices is increasing one day after another. According to the latest report from AVTEST Security Institute, the number of malware captured in the first quarter of 2019 exceeded 889 million with 4.48 million new malware [1]. It is impossible to handle this large number of malware files manually. Therefore, anti-virus software products mainly adopted signature-based techniques to detect malware. In signature-based techniques, a unique sequence of bytes is extracted from the captured malware and used to detect similar malicious files. The detection process is fast with low false positive rates (FPR) [2]. However, attackers can easily modify malware signatures to avoid being detected by anti-virus software. In addition, the common malware use obfuscation techniques such as code manipulation, instruction substitution, register reassignment, and the dead code insertion to escape the detection process. Therefore, this approach is incapable of detecting unknown and new malware that have not been seen before, and it is easily deceived by malware that uses obfuscation, packing or encryption techniques [3].

In order to overcome the limitations of signature-based techniques, researchers have been concerned with studying and analyzing the different characteristics of malware such as behaviors, selected targets and evolution over time [4]. The techniques used for malware analysis can be categorized according to the way malware analysis is performed. Static analysis involves studying malicious files without executing them. However, in order to start a static analysis, a portable executable (PE) file must be first decompressed and then unpacked. In general, static analysis is inexpensive and is able to provide a view of all possible malware execution paths. However, it is common for current malware to use encryption and binary packers to avoid the detection and analysis process [5]. Dynamic analysis, also called behavior analysis, is another type of analysis where infected files are run in a controlled environment in order to be monitored and analyzed. Unlike static analysis, dynamic analysis does not need to disassemble or unpack a malicious file to examine it. Thus, dynamic analysis is more effective as malware that uses obfuscated and packed techniques cannot evade detection since obfuscation and packing do not change the behavior of the malware. However, dynamic analysis consumes considerable resources and requires intensive time [5]. It is possible for smart malware to change their behaviors or terminate themselves if they discover the existence of a controlled environment. [6]. Moreover, dynamic analysis can only provide one way of file execution, which is limiting its ability to study the general behaviors of a malicious file.

Memory analysis, on the other hand, is a promising technique that has become more and more popular in recent years, and has been proven to be efficient and accurate in studying malware behaviors. Analyzing memory is an excellent way to discover malicious activities in the system because volatile memory preserves its contents until it is powered off. Further, valuable live information which resides in the memory may include active processes, DLLs, registry keys, services, sockets and ports, and active network connections. Furthermore, malware hooks and malicious codes outrange the normal function scope which can be detected by performing a memory analysis. Additionally, information such as the operating system, running processes, and general state of the computer can be extracted as well [7]. Therefore, memory analysis provides a comprehensive analysis about the environment and the general view of malware. Memory analysis passes through two stages: Memory acquisition and memory analysis. In the first stage, target machine memory is dumped in order to get a full memory image. In the second stage, the memory image is analyzed searching for malicious activities using a memory forensic tool, like Volatility tool. This study proposes an integrated approach that can detect malware by combining malicious artifacts found in the memory and using memory forensics techniques, with features extracted while executing malware files using a dynamic analysis. The purpose of this research is to increase the accuracy of malware detection and reduce the false alarmed files. This study explained that adding more relevant features available in the memory to features extracted from the dynamic analysis increases the detection accuracy of our proposed approach. In addition, the approach overcomes the limitation of one path of file execution in dynamic analysis by looking for extra features that reside in the memory and do not appear in the dynamic analysis report. These features might reveal an unseen part of malware and, therefore, improve the overall performance of the proposed malware detection approach. Our work and experiments have focused on Windows operating system as it is the most targeted operating system by malware campaigns. The rest of the paper is organized as follows: The next section introduces the related work. In Section 3, an overview of the proposed approach is explained in detail. The experimental results and discussions are presented in Section 4. Finally, the conclusion and future works are described.

2. Related Work

Researchers have used different techniques to analyze and detect malware. Several researchers used static analysis to detect malware. Hashemi and Hamzeh [8] used machine learning to detect malware after extracting the unique opcode feature from executable files using static analysis. Salehi et al. [6] extracted the application programming interface (API) calls from each file and selected the most frequent APIs to learn the classifier. Similarly, Cheng et al. [9] used the WinDbg tool to extract native APIs and applied SVM to classify malware in the shellcode. Sun et al. [10] proposed a method that uses static analysis to extract opcode sequences from 32-bit and 64-bit malicious portable executable (PE) Windows files. The frequency for each opcode sequence is then calculated and used to distinguish the different types of malicious PE files. The experiment evaluated on more than 20,000 samples collected from the vx-heaven data set and Kaggle Microsoft Malware Classification Challenge (BIG 2015) data set. The results showed that the highest accuracy equaled to 98.57% by applying the Adaboost classifier. Further, Kolosnjaji et al. [11] investigated the vulnerability of malware detection methods that use deep networks to learn from row bytes and proposed a gradient-based attack which was able to evade detection by modifying a few bytes at the end of each malware file.

On the other hand, many authors used dynamic analysis in malware detection and classification. In [12], Egele et al. surveyed the dynamic analysis and feature extraction techniques along with the available tools that have been used in dynamic analysis. Mohaisen et al. [13] extracted a registry file system and network features and classified Zeus malware. Following this, Mohaisen et al. [14] introduced an automated and behavior-based malware analysis and labeling system (AMAL). The features, such as the file system, registry and network activity were used to study malware files. In [15], Liang et al. introduced a classification technique that calculates the similarity between malware variants based on the API dependency chain. Likewise, Galal et al. [16] captured information about API calls and its parameters by applying the API hook. The API calls that have mutual purposes were put into sequences. Similarly, Fan et al. [17] applied API hooking to trace hidden API calls and native APIs. Ki et al. [18] also extracted user level API call sequences, using the Detours tool, and applied a multiple sequence alignment algorithm (MSA) to match similar sequences. In [19], Ding et al. introduced a malware detection method that represents malware behaviors based on API as dependency graphs. To find the relationship between system calls, a dynamic taint analysis technique was used. The system call dependency graph was built according to the propagation of the taint data. They also proposed an algorithm that extracts behavioral features of a malware family. Finally, a malicious code was detected using a maximum weight subgraph. The results of the experiment show that the behavior graph API-based can represent a malware family with accuracy equal to 95.2%.

Some researchers chose to extract hybrid features from static and dynamic analyses in representing their malware detection approaches. Shijo and Salim [20] extracted printable strings information (PSI) features by performing a static analysis and API calls from a dynamic analysis. Likewise, Islam et al. [21] applied static analysis to extract the function length frequency (FLF) and printable string information (PSI) features, and used dynamic analysis to extract API calls and API parameters. In [22], Santos et al. presented a tool that is able to capture malware files (OPEM). Santos applied static analysis to extract opcode frequency and system calls, and dynamic analysis for operations and raised exceptions.

Several types of research concerning memory analysis have been proposed. In [23], Teller and Hayon introduced a memory dump approach that triggered based on the three events: Performance, API and instrumentation based. The approach dumps the memory based on the events rather than dumping the memory at the end of the execution process. Vömel and Freiling [24] surveyed the main memory acquisition and analysis techniques. Rathnayaka and Jamdagni [25] tested 200 malicious files using static analysis and memory forensics. They observed that successful malware infection leaves a footprint in the memory. Zaki and Humphrey [26] examined the artifacts left by rootkits in the memory kernel-level such as the drivers, modules, SSDT hooks, IDT hooks and callback functions. The experiment noticed that activities like modified drivers, callback functions and attached devices, are the major activities related to malware in the kernel-level. In [27], Aghaeikheirabady et al. introduced an analysis method for extracting features like DLLs, function calls and the registry from the memory. The method aimed to increase the information accuracy by comparing it with the information available in other memory structures. The features were then classified based on their frequency in the training data set, and the detection rate of 98% was achieved through Naïve Bayes. A significant drawback of this method is the high false positive rate. In the Aghaeikheirabady et al study, it exceeded 16%. Similarly, Mosli et al. [4] introduced an approach to detect malware based on three features extracted from the memory images, registry activities, API function calls, and imported libraries. The authors tested each feature individually, and the maximum accuracy equaled to 96% which was reached by applying SVM on the data from the registry activities. Afterward, in the next research, Mosli et al. [28]

introduced an approach that uses process handles to detect malicious files with accuracy slightly higher than 91.4%. The experiment observed that the main types of handles for the malicious files used were section handles, process handles and mutants. Likewise, Dai et al. [29] introduced a method for malware classification based on a memory dump. A file is extracted from the memory dump and then converted into an image with a gray scale. After that, the image is transformed into a fixed size. The API call features were extracted from the images and used to classify malware. The highest accuracy equaled to 93.9% which was achieved by applying the random forest. Despite the favorable results achieved by the above techniques, no other works attempt to integrate dynamic analysis with memory analysis to get the advantages of both analyses and overcome their limitations.

3. Methodology and Framework

In this section, the architecture of the proposed approach is discussed in detail. The data set consists of malware files that were collected from two sources in addition to a set of benign files. An analysis environment is then setup, which consists of all the necessary components to provide a suitable atmosphere to execute the samples and generate the required analysis reports. The feature extraction is then performed, and pre-modeling is applied. Finally, the data set is used to train and test the classification accuracy of several machine learning models. The overall framework is illustrated in Figure 1 and discussed in detail in the following subsections.

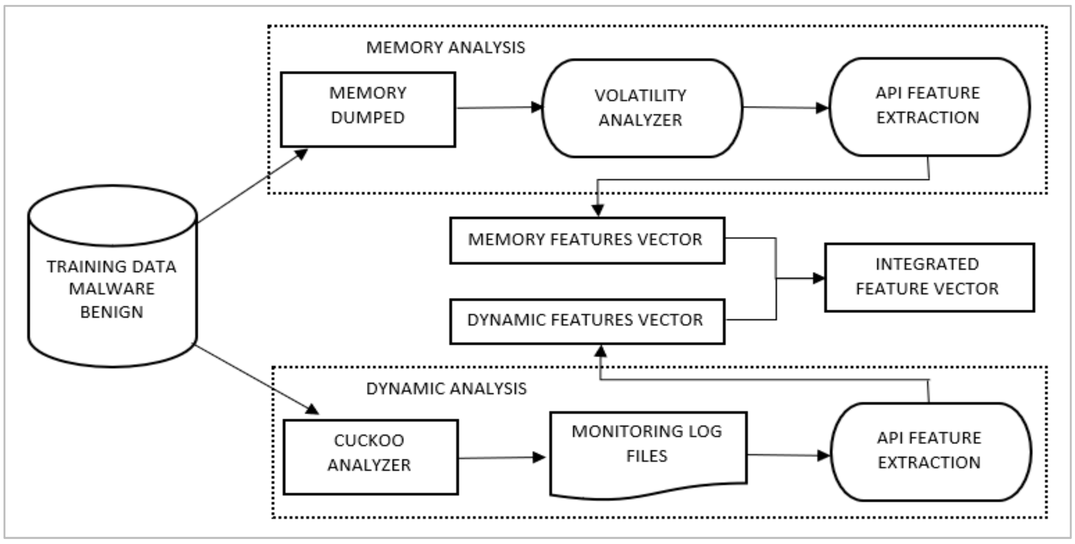


Figure 1. Framework of the proposed approach.

3.1. Data Set

The data set consists of 1200 portable executable (PE) malware and 400 benign samples. The malicious files were collected from two sources. The first source is the VirusTotal repository [30], where 900 malicious files were downloaded, which were captured between 2017 and 2019. The additional 300 malware samples captured between 2015 and 2017 were downloaded from Das Malwerk [31]. The malware data set contained an equal quantity of the following malware families: Adware, Ransomware, Keylogger, Downloader and Backdoor. The purpose of collecting malware from two sources in two different periods is to have a comprehensive and reliable data set as malware are changing their tactics and behaviors [21]. On the other hand, benign files were collected from Windows 7 operating system (Win 7 32-bit) files.

3.2. Environment Setup

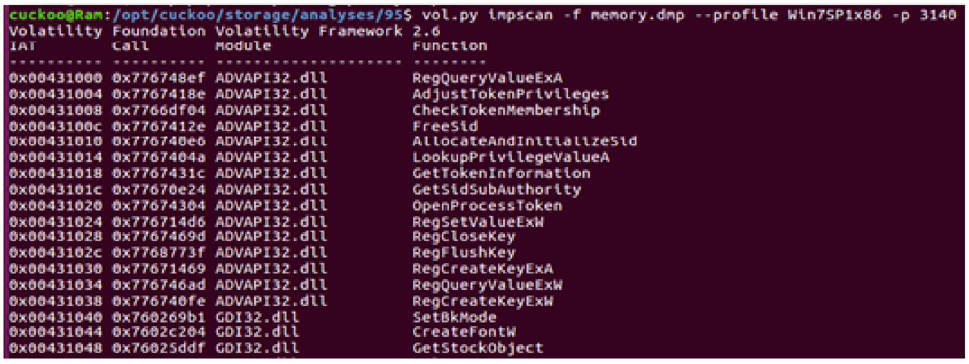
The analysis environment involved a machine running Ubuntu as a host operating system and Windows 7 as a guest system, running on one giga byte of RAM. The host system carried out the responsibility of running other necessary programs, such as the Cuckoo Sandbox, VirtualBox and Volatility tool. The Cuckoo Sandbox was first configured and then used for two purposes: Analyzing the behavior of the tested file and getting the memory image by dumping the memory at the end of each file execution.

3.3. Execution Process

The execution process consisted of two stages. (1) Dynamic analysis. The Cuckoo Sandbox was used to monitor the behavior of each file while it was running on the guest system, and then produced a behavior report with the Java server object notation (JSON) format after the process was completed. (2) Memory Analysis. At this stage, the Volatility tool was used to analyze each memory image and produce a memory analysis report with the JSON format. At the end of the execution process, two reports were generated: Behavior and memory reports for each tested sample. Thus, the total number of generated reports was 3200 for entire data set.

3.4. Feature Extraction

The API call feature was then extracted from the behavior and memory reports. The behavior report produced by the Cuckoo Sandbox contained the executed API calls in addition to other information (log file). Therefore, the API calls were directly extracted from each behavior report. However, in the memory, the API function calls existed in the import address table (IAT). Therefore, the Impscan command from the Volatility tool was applied to extract the API function calls from the memory image. The Impscan command scans the memory image looking for API calls in the IAT table plus other related information [32] (see Figure 2). While the number of features extracted from memory images was 4705, the number of features from dynamic analysis was 4280 features.

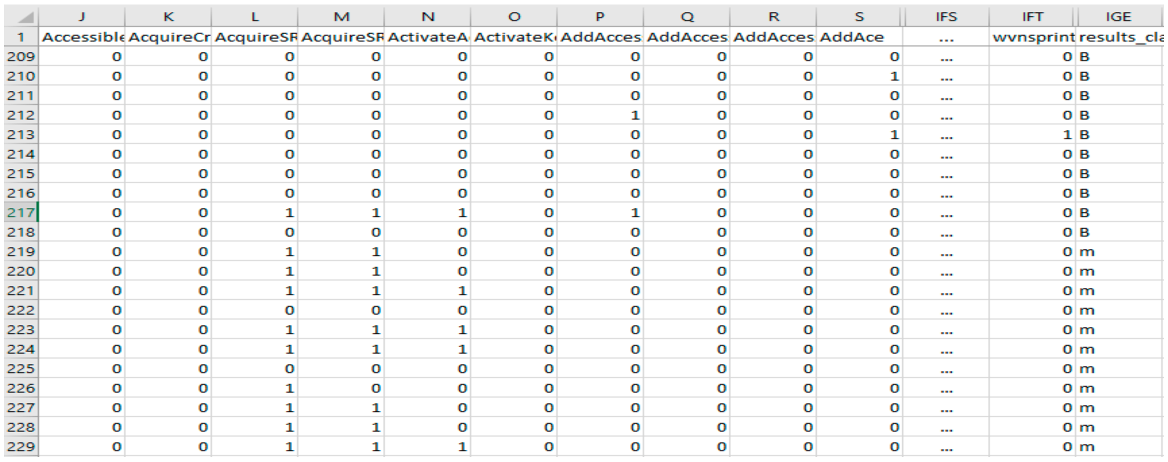


```
cuckoo@Ram:/opt/cuckoo/storage/analyses/95$ vol.py impscan -f memory.dmp --profile Win7SP1x86 -p 3140
Volatility Foundation Volatility Framework 2.6
IAT        Call       Module               Function
---------- ---------- -------------------- --------
0x00431000 0x776748ef ADVAPI32.dll         RegQueryValueExA
0x00431004 0x7767418e ADVAPI32.dll         AdjustTokenPrivileges
0x00431008 0x7766df04 ADVAPI32.dll         CheckTokenMembership
0x0043100c 0x7767412e ADVAPI32.dll         FreeSid
0x00431010 0x776740e6 ADVAPI32.dll         AllocateAndInitializeSid
0x00431014 0x7767404a ADVAPI32.dll         LookupPrivilegeValueA
0x00431018 0x7767431c ADVAPI32.dll         GetTokenInformation
0x0043101c 0x77670e24 ADVAPI32.dll         GetSidSubAuthority
0x00431020 0x77674304 ADVAPI32.dll         OpenProcessToken
0x00431024 0x776714d6 ADVAPI32.dll         RegSetValueExW
0x00431028 0x7767469d ADVAPI32.dll         RegCloseKey
0x0043102c 0x7768773f ADVAPI32.dll         RegFlushKey
0x00431030 0x77671469 ADVAPI32.dll         RegCreateKeyExA
0x00431034 0x776746ad ADVAPI32.dll         RegQueryValueExW
0x00431038 0x776740fe ADVAPI32.dll         RegCreateKeyExW
0x00431040 0x760269b1 GDI32.dll            SetBkMode
0x00431044 0x7602c204 GDI32.dll            CreateFontW
0x00431048 0x76025ddf GDI32.dll            GetStockObject
```

Figure 2. Extracting API calls from the memory image using the impscan command.

3.5. Pre-Modeling

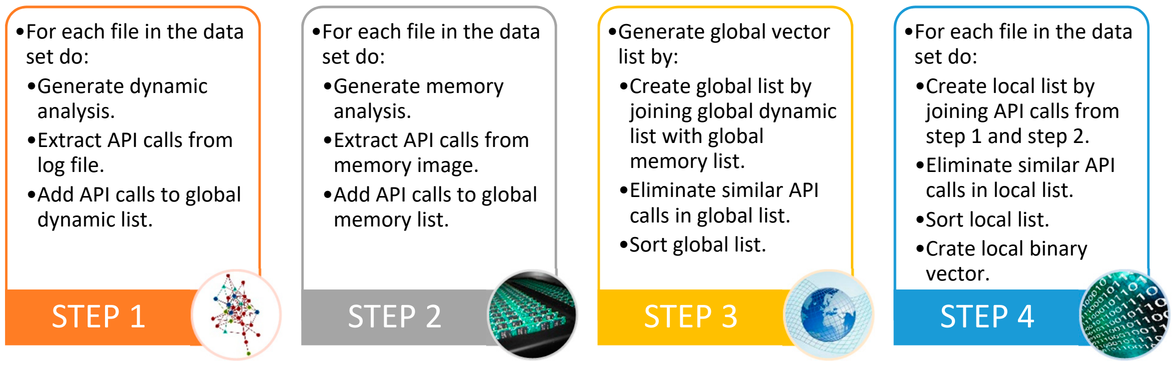
In this stage, feature engineering techniques were applied to improve the quality of the API features. The features extracted from both the memory and dynamic analyses were joined together. The total number of features was 8985 features. The duplicate features were then removed from the list. Thus, the total number of distinct features from both analyses was 6270 features. With simple calculations; the number of behavior features was 4280 and the total number of unduplicated features from both analyses was 8985. Therefore, that means there were at least 2715 independent features which have been found in the memory but not during the execution of malicious files in dynamic analysis. The independent features are a good sign of successful detection and the classification process. After that, the list was alphabetically reordered. The new list which contained the entire distinct features was considered as a global list (or global vector). Similarly, the same procedure was applied for each malware and benign file. However, the list for each file, which consisted of features extracted from both the memory and behavior analyses, were considered as a local list (or local vector). Thus, eventually, one global list and 1600 local lists were obtained. In the next step, each local list was converted to a binary vector where 1 represented the fact that the feature in a local list was presented in the global list and 0 if it was not. Similar samples (instances) were removed in order to increase the accuracy of the classification process. A sample of the global vector and binary local vector is shown in Figure 3.



	J	K	L	M	N	O	P	Q	R	S	IFS	IFT	IGE
1	Accessible	AcquireCr	AcquireSR	AcquireSR	ActivateA	ActivateK	AddAcces	AddAcces	AddAcces	AddAce	...	wvnsprint	results_cla
209	0	0	0	0	0	0	0	0	0	0	...	0	B
210	0	0	0	0	0	0	0	0	0	1	...	0	B
211	0	0	0	0	0	0	0	0	0	0	...	0	B
212	0	0	0	0	0	0	1	0	0	0	...	0	B
213	0	0	0	0	0	0	0	0	0	1	...	1	B
214	0	0	0	0	0	0	0	0	0	0	...	0	B
215	0	0	0	0	0	0	0	0	0	0	...	0	B
216	0	0	0	0	0	0	0	0	0	0	...	0	B
217	0	0	1	1	1	0	1	0	0	0	...	0	B
218	0	0	0	0	0	0	0	0	0	0	...	0	B
219	0	0	1	1	0	0	0	0	0	0	...	0	m
220	0	0	1	1	0	0	0	0	0	0	...	0	m
221	0	0	1	1	1	0	0	0	0	0	...	0	m
222	0	0	0	0	0	0	0	0	0	0	...	0	m
223	0	0	1	1	1	0	0	0	0	0	...	0	m
224	0	0	1	1	1	0	0	0	0	0	...	0	m
225	0	0	0	0	0	0	0	0	0	0	...	0	m
226	0	0	1	0	0	0	0	0	0	0	...	0	m
227	0	0	1	1	0	0	0	0	0	0	...	0	m
228	0	0	1	1	0	0	0	0	0	0	...	0	m
229	0	0	1	1	1	0	0	0	0	0	...	0	m

Figure 3. Generating global and local vectors for one file.

Moreover, a python script was developed to extract API calls from both the behavior reports and memory reports (feature extraction) and generate local vectors and global vector (pre-modeling). The script performs four major steps and is outlined in Figure 4.



- For each file in the data set do:
  - Generate dynamic analysis.
  - Extract API calls from log file.
  - Add API calls to global dynamic list.

STEP 1

- For each file in the data set do:
  - Generate memory analysis.
  - Extract API calls from memory image.
  - Add API calls to global memory list.

STEP 2

- Generate global vector list by:
  - Create global list by joining global dynamic list with global memory list.
  - Eliminate similar API calls in global list.
  - Sort global list.

STEP 3

- For each file in the data set do:
  - Create local list by joining API calls from step 1 and step 2.
  - Eliminate similar API calls in local list.
  - Sort local list.
  - Crate local binary vector.

STEP 4

Figure 4. Four steps performed by the python script for feature extraction and pre-modeling.

The effectiveness of the integrated vector is examined in the classification process, which is described in the next section.

4. Results and Discussion

4.1. Machine Learning Modeling

Several machine learning algorithms were applied on the integrated vector using the WEKA tool; the support vector machine (SVM), the decision tree, Naïve Bayes (NB), K-nearest neighbor (KNN) and the random forest. A 10-fold cross validation was applied to evaluate the classification models. Thus, the data set samples were shuffled and split into 10 groups and the data used in training the classifier was completely separated from the testing one. By using a 10-fold cross validation and by removing all the duplicates in the data set, the performance of a given model was estimated in a better way. Moreover, the following measures are used to evaluate the results:

- True Positive (TP): number of correctly classified malicious files.

- True Negative (TN): number of correctly classified benign files.
- False Positive (FP): number of benign files wrongly classified as malicious files.
- False Negative (FN): number of malicious files wrongly classified as benign files.
- Detection Rate (DR): = $\frac{TP}{TP+FN}$.
- False Positive Rate (FPR): = $\frac{FP}{FP+TN}$.
- Accuracy: = $\frac{TP+TN}{TP+TN+FP+FN}$.

In order to examine the accuracy of the proposed approach, three experiments were performed. In the first experiment, the API call features extracted from the memory analysis only were classified. The global vector and local vectors for this experiment were also constructed from the API features extracted from the memory analysis. The experiment result is shown in Table 1. This table shows that the K-nearest neighbor (KNN) machine learning algorithm performed better with accuracy close to 95% and a false positive rate equal to 5.1%.

Table 1. The result of classifying memory features.

Classifier	DR (%)	FPR (%)	Accuracy (%)	Time (ms)
Naïve Bayes	94.4	5	94.6	3.7
SVM	93.8	6.4	93.9	11.4
Decision Tree	88.2	13.3	88.2	3.4
Random Forest	93.8	6.7	93.8	27.2
KNN	94.7	5.1	94.8	1.0

In the next experiment, the API call features extracted during the dynamic analysis were only classified. Likewise, the global vector and local vectors were created from the API features obtained in dynamic analysis. The result is shown in Table 2. The performance of SVM is the highest among others with an accuracy rate of 97.4% and false positive rate of slightly less than 5%.

Table 2. The result of classifying dynamic features.

Classifier	DR (%)	FPR (%)	Accuracy (%)	Time (ms)
Naïve Bayes	80.8	8.2	88.6	9.2
SVM	97.4	4.9	97.4	22.8
Decision Tree	95.7	8.4	95.7	31.7
Random Forest	96.5	5.2	96.6	47.3
KNN	96.2	5.3	96.3	2.0

In the third experiment, the proposed integrated approach was examined. The combination of the API call features from the memory and dynamic analyses was used to create the global vector and local vectors, as discussed earlier in the feature extraction and pre-modeling subsections. The results (shown in Table 3) display a significant improvement in the accuracy rate as well as an impressive decrease in the false positive rate. The integrated approach applying the support vector machine (SVM) algorithm outperformed both the memory analysis and dynamic analysis with an accuracy rate equal to 98.5% and false positive rate as low as 1.7%. Figure 5 shows the evaluation rates for the malware and benign files when applying the SVM algorithm.

Table 3. The results of the classifying integrated features.

Classifier	DR (%)	FPR (%)	Accuracy (%)	Time (ms)
Naïve Bayes	87.4	6.3	90.7	12.3
SVM	98.4	1.7	98.5	15.1
Decision Tree	96	5.1	96.1	46.1
Random Forest	97.9	2	97.9	30.0
KNN	96.6	3.6	96.7	1.0

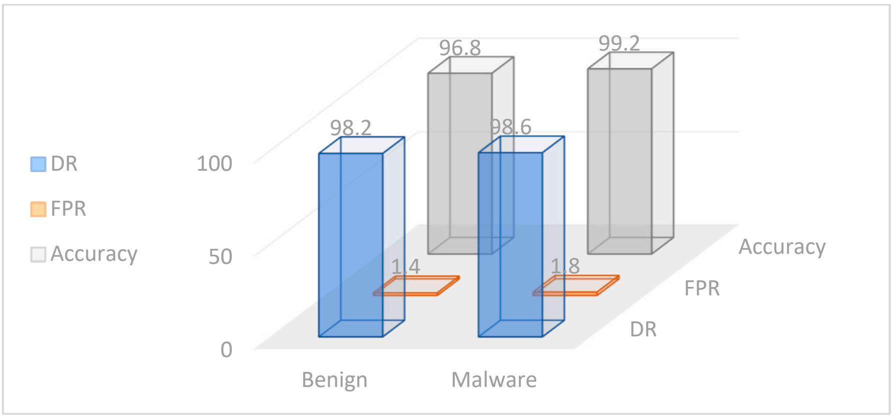


Figure 5. The evaluation rates applying the support vector machine (SVM) on the integrated features.

It is well known that smart malware change their behaviors or terminate themselves when they are placed in a controlled environment. In addition, dynamic analysis can only demonstrate one path of file execution. However, malware is loaded to the memory to be executed. Furthermore, it is preliminary for a malware file to be unpacked and decrypted to be executed. Moreover, malware artifacts remain in the memory for a while, even after the process is terminated [3]. Therefore, the proposed approach, which combines features from both the dynamic and memory analyses better, presents malware behaviors and intentions that are reflected in higher accuracy detection and lower false positive rates.

The proposed approach outperforms the other two analyses in terms of accuracy and false positive rates. The highest accuracy rate achieved by the proposed approach was equal to 98.5% compared to 97.4% and 94.8% reached by the dynamic analysis and memory analysis respectively. Moreover, the integrated approach impressively decreases false positive rates, which come as low as 1.7% compared to 4.9% for the dynamic analysis and 5.1% for the memory analysis. A comparison between the evaluation rates from the memory, dynamic and integrated analyses is illustrated in Figure 6.

The classification time was also measured for each algorithm in the three experiments. The classification time for the integrated approach applying SVM was 15.1 milliseconds. The time reduced after eliminating all the duplicates from the original data set.

Our experiment was compared with other memory-based related works. The optimal result of the accuracy and false positive rates for the proposed method was compared to the results of the related experiments, which used Windows as the operating system and the memory analysis as the detection method (see Table 4). From the table, it can be seen that the proposed method has a higher detection rate and a lower false positive rate, and from the comparison of the experimental results, the proposed method outperforms other related works. Although the proposed method has a slightly higher detection accuracy than Aghaeikheirabady [27], there is a significant decrease in the false positive rate between the two methods from 16% to only 1.7%.

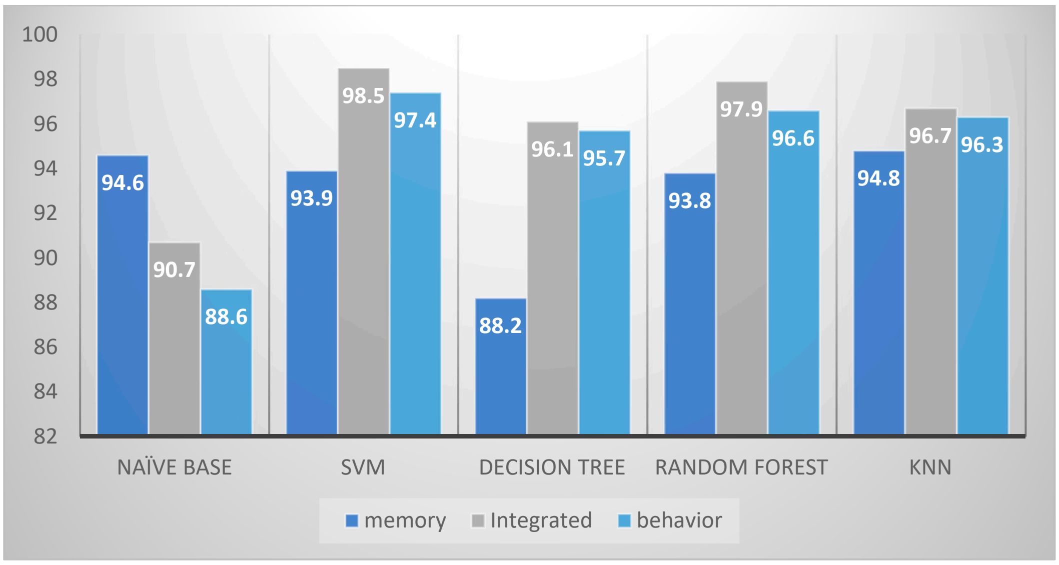


Figure 6. A comparison between the evaluation rates from the memory, dynamic and integrated analyses.

Table 4. Comparing experimental results with other memory related work.

Author	**Year**	**Extracted Feature (s)**	**Malware (M)/Benign (B)**	**Accuracy**	**False Positive**
Mosli [4]	2016	Registry, DLLs, API calls.	M = 400 B = 100	96%	5%
Mosli [28]	2017	Number of handles.	M = 3130 B = 1157	91.4%	-
Dai [29]	2018	API calls.	M = 1984	95.2%	-
Aghaeikheirabady [27]	2014	DLL, API calls, Registry.	-	98%	16%
Proposed method	2019	API calls.	M = 1200 B = 400	98.5%	1.7%

Memory analysis is an effective way to investigate malware goals and activities in memory. However, malware authors try to escape memory analysis by using anti-memory forensics, such as hiding the infected memory regions or preventing analysts from extracting memory information [12]. Additionally, malware try to hide any evidence that uncover its malicious activities seeking to be untraceable and invisible as possible. Therefore, combining behavior analysis and memory analysis overwhelms the memory analysis limitations and better detects malware activities. Further, the combination helps in adding extra related features and, therefore, increases the detection accuracy and lowers the wrongly classified files in the proposed approach.

4.2. Features in Memory and Dynamic Analysis

In order to expand our study of malware behaviors in memory, the API features extracted from the memory analysis with features extracted from the dynamic analysis for each file were compared. There are features found in the memory analysis that have not been seen in the dynamic analysis report. This means that the malware did not execute those APIs. It is known that dynamic analysis is limited to a single view of path execution and, therefore, unexecuted API calls do not appear in the behavior report. However, unexecuted APIs could reveal another side of malware behavior [3]. Table 5 shows the 20 most prominent features that appear in the memory analysis but not in the dynamic analysis with their frequencies and percentages. To exemplify, the RtlEncodePointer and RtlDecodePointer features were found in 630 files in the memory analysis, in 52.5% of the total data set, but not seen during monitoring the execution of that file (dynamic analysis). The TerminateProcess feature was not found 314 times through the execution of 1200 malicious samples. Therefore, the authors believe that memory forensic analysis overwhelms the shortage of single path execution in dynamic analysis.

Table 5. The 20 most prominent features appearing in the memory analysis.

#	Feature	Freq	%	#	Feature	Freq	%
1	RtlEncodePointer	630	52.5	11	UnhandledExceptionFilter	319	26.6
2	RtlDecodePointer	630	52.5	12	GetCurrentProcessId	317	26.4
3	RtlExitUserThread	573	47.8	13	GetDC	314	26.2
4	GetLastError	360	30	14	TerminateProcess	314	26.2
5	GetProcAddress	348	29	15	HeapAlloc	313	26.1
6	FreeLibrary	345	28.8	16	HeapFree	313	26.1
7	GetCurrentProcess	337	28.1	17	QueryPerformanceCounter	313	26.1
8	WideCharToMultiByte	337	28.1	18	EnterCriticalSection	312	26
9	MultiByteToWideChar	327	27.3	19	LeaveCriticalSection	312	26
10	GetCurrentThreadId	323	26.9	20	WriteFile	312	26

In addition to the features in Table 5, there are more than 2821 distinct features that have not been seen in the behavior analysis reports. There are two assumptions for the unexecuted features—either the malware did not execute these features yet, or it did not want to execute them. It is possible that the malware did not perform some of its activities because of the time limit of dynamic analysis, which is commonly 2–5 min. The second assumption is that the malware is trying to hide its activities or part of it, which is also common for smart malware to act normally when they are placed under analysis in order not to show any malicious actions. However, in both cases, the inspecting memory can reveal unrecorded behaviors of the malware and can also help in extracting additional features. Therefore, the memory analysis and behavior analysis were integrated in order to find malware hidden features and improve the classification of the proposed approach.

5. Conclusions

This study developed an effective and reliable approach that can be implemented in classifying malicious files at the end-user computer. The purpose of this research is to increase the detection accuracy of malware and reduce false positive rates. Based on the experiments, it was proved that adding more related features from the memory to features extracted from dynamic analysis enhances the detection accuracy. In addition, the proposed approach overcomes the limitation of one path of execution in dynamic analysis by looking for extra features in the memory that did not appear in the dynamic analysis, which might reveal a hidden part of malware and, therefore, improves the performance of the proposed detection approach. Our work and experiments have focused on Windows operating system as it is the most targeted operating system by malware campaigns. The proposed approach is comprehensive and consistent. It collects memory artifacts using memory forensics tools in addition to features extracted during the execution of malicious files through dynamic analysis. The malware data set consisted of malware downloaded from two sources and they were captured in different periods of time. Moreover, the data set contained an equal quantity of malware that represent Adware, Ransomware, Keylogger, Downloader and Backdoor families. After the completion of the features engineering process, the features were used in training and testing the classifier. A 10-fold cross validation was applied to evaluate the proposed approach by applying the following machine learning algorithms: The support vector machine (SVM), the decision tree, Naïve Bayes, K-nearest neighbor (KNN) and the random forest. The results show a significant improvement using SVM in the detection accuracy rate and a noteworthy decrease in false positive rates of 98.5% and 1.7% respectively.

In this work, only API call features were extracted from the memory images and dynamic analysis, and used in the classification process. Although API calls achieved a high classification result. However, other features, like the registry and networking features, are to be considered for improving the proposed approach in future work. The proposed approach can be improved by increasing the number of malware in the data set as well. Furthermore, more works are required to enhance the abilities of the sandbox environments against malware anti-forensics and anti-virtualization techniques.

Author Contributions: Conceptualization, R.S., K.O. and K.A.Z.A.; Formal analysis, R.S., K.O. and K.A.Z.A.; Methodology, R.S., K.O., K.A.Z.A. and S.A.A.; Resources, R.S.; Supervision, K.O. and K.A.Z.A.; Visualization, R.S.; Writing—original draft, R.S.; Writing—review & editing, R.S., K.O., K.A.Z.A. and S.A.A.; Project administration, R.S., K.O. and K.A.Z.A.

Funding: This research was funded by Deanship of Scientific Research at King Faisal University.

Acknowledgments: The authors acknowledge the Deanship of Scientific Research at King Faisal University for the financial support under Nasher Track (Grant No. 186256).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. AV-TEST. AVTEST The Independent ITSecurity Institute. 2019. Available online: https://www.av-test.org/ (accessed on 30 May 2019).
2. Damodaran, A.; di Troia, F.; Visaggio, C.A.; Austin, T.H.; Stamp, M. A comparison of static, dynamic, and hybrid analysis for malware detection. *J. Comput. Virol. Hacking Tech.* **2017**, *13*, 1–12. [CrossRef]
3. Sihwail, R.; Omar, K.; Ariffin, K.A.Z. A Survey on Malware Analysis Techniques: Static. *Dyn. Hybrid Mem. Anal.* **2018**, *8*, 1662–1671.
4. Mosli, R.; Li, R.; Yuan, B.; Pan, Y. Automated malware detection using artifacts in forensic memory images. In Proceedings of the 2016 IEEE Symposium on Technologies for Homeland Security, HST, Waltham, MA, USA, 10–11 May 2016; Volume 2016, pp. 1–6.
5. Ye, Y.; Li, T.; Adjeroh, D.; Iyengar, S.S. A Survey on Malware Detection Using Data Mining Techniques. *ACM Comput. Surv.* **2017**, *50*, 41. [CrossRef]
6. Salehi, Z.; Sami, A.; Ghiasi, M. Using feature generation from API calls for malware detection. *Comput. Fraud Secur.* **2014**, *2014*, 9–18. [CrossRef]
7. Okolica, J.; Peterson, G. A compiled memory analysis tool. In *IFIP Advances in Information and Communication Technology*; Springer: Berlin/Heidelberg, Germany, 2010; Volume 337, pp. 195–204.
8. Hashemi, H.; Hamzeh, A. Visual malware detection using local malicious pattern. *J. Comput. Virol. Hacking Tech.* **2018**, *15*, 1–14. [CrossRef]
9. Cheng, Y.; Fan, W.; Huang, W.; An, J. A Shellcode Detection Method Based on Full Native API Sequence and Support Vector Machine. In *IOP Conference Series: Materials Science and Engineering*; IOP Publishing: Bristol, UK, 2017; Volume 242, pp. 1–7.
10. Sun, Z.; Rao, Z.; Chen, J.; Xu, R.; He, D.; Yang, H.; Liu, J. An Opcode Sequences Analysis Method For Unknown Malware Detection. In Proceedings of the 2019 2nd International Conference on Geoinformatics and Data Analysis, Prague, Czech Republic, 15–17 March 2019; ACM: New York, NY, USA, 2019.
11. Kolosnjaji, B.; Demontis, A.; Biggio, B.; Maiorca, D.; Giacinto, G.; Eckert, C.; Roli, F. Adversarial malware binaries: Evading deep learning for malware detection in executables. In Proceedings of the 2018 26th European Signal Processing Conference (EUSIPCO), Rome, Italy, 3–7 September 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 533–537.
12. Egele, M.; Scholte, T.; Kirda, E.; Kruegel, C. A survey on automated dynamic malware-analysis techniques and tools. *ACM Comput. Surv.* **2012**, *44*, 6. [CrossRef]
13. Mohaisen, A.; Alrawi, O. Unveiling Zeus Automated Classification of Malware Samples. In Proceedings of the 22nd International Conference on World Wide Web Companion, Rio de Janeiro, Brazil, 13–17 May 2013; pp. 829–832.
14. Mohaisen, A.; Alrawi, O.; Mohaisen, M. AMAL: High-fidelity, behavior-based automated malware analysis and classification. *Comput. Secur.* **2015**, *52*, 251–266. [CrossRef]
15. Liang, G.; Pang, J.; Dai, C. A Behavior-Based Malware Variant Classification Technique. *Int. J. Inf. Educ. Technol.* **2016**, *6*, 291–295. [CrossRef]
16. Galal, H.S.; Mahdy, Y.B.; Atiea, M.A. Behavior-based features model for malware detection. *J. Comput. Virol. Hacking Tech.* **2016**, *12*, 59–67. [CrossRef]

17. Fan, C.-I.; Hsiao, H.-W.; Chou, C.-H.; Tseng, Y.-F. Malware Detection Systems Based on API Log Data Mining. In Proceedings of the 2015 IEEE 39th Annual Computer Software and Applications Conference, Los Alamitos, CA, USA, 10 June 2015; pp. 255–260.
18. Ki, Y.; Kim, E.; Kim, H.K. A novel approach to detect malware based on API call sequence analysis. *Int. J. Distrib. Sens. Netw.* **2015**, *11*, 659101. [CrossRef]
19. Ding, Y.; Xia, X.; Chen, S.; Li, Y. A malware detection method based on family behavior graph. *Comput. Secur.* **2018**, *73*, 73–86. [CrossRef]
20. Shijo, P.V.; Salim, A. Integrated static and dynamic analysis for malware detection. *Procedia Comput. Sci.* **2015**, *46*, 804–811. [CrossRef]
21. Islam, R.; Tian, R.; Batten, L.M.; Versteeg, S. Classification of malware based on integrated static and dynamic features. *J. Netw. Comput. Appl.* **2013**, *36*, 646–656. [CrossRef]
22. Santos, I.; Devesa, J.; Brezo, F.; Nieves, J.; Bringas, P.G. AISC 189-OPEM: A Static-Dynamic Approach for Machine-Learning-Based Malware Detection. In *Advances in Intelligent Systems and Computing*; AISC: Chicago, IL, USA, 2013; Volume 189, pp. 271–280.
23. Teller, T.; Hayon, A. *Enhancing Automated Malware Analysis Machines with Memory Analysis Report*; Black Hat USA: Las Vegas, NV, USA, 2014.
24. Vömel, S.; Freiling, F.C. A survey of main memory acquisition and analysis techniques for the windows operating system. *Digit. Investig.* **2011**, *8*, 3–22. [CrossRef]
25. Rathnayaka, M.C.; Jamdagni, A. An Efficient Approach for Advanced Malware Analysis using Memory Forensic Technique. In Proceedings of the 2017 IEEE Trustcom/BigDataSE/ICESS, Sydney, NSW, Australia, 1–4 August 2017; pp. 1145–1150.
26. Zaki, A.; Humphrey, B. "Unveiling the kernel: Rootkit discovery using selective automated kernel memory differencing," Virus Bull., no. September 2014; pp. 239–256.
27. Aghaeikheirabady, M.; Farshchi, S.M.R.; Shirazi, H. A new approach to malware detection by comparative analysis of data structures in a memory image. In Proceedings of the 2014 International Congress on Technology, Communication and Knowledge (ICTCK), Mashhad, Iran, 26–27 November 2014; Volume 1, pp. 273–278.
28. Mosli, R.; Li, R.; Yuan, B.; Pan, Y. A behavior-based approach for malware detection. In *IFIP Advances in Information and Communication Technology*; Springer: Cham, Switzerland, 2017; Volume 511, pp. 187–201.
29. Dai, Y.; Li, H.; Qian, Y.; Lu, X. A malware classification method based on memory dump grayscale image. *Digit. Investig.* **2018**, *27*, 30–37. [CrossRef]
30. Virus Total. 2019. Available online: https://www.virustotal.com/#/home (accessed on 1 April 2019).
31. DAS MALWERK. 2019. Available online: https://dasmalwerk.eu/ (accessed on 1 April 2019).
32. Gleeda, Command Reference Mal. 2017. Available online: https://github.com/volatilityfoundation/volatility/wiki/Command-Reference-Mal#impscan (accessed on 1 Jun 2019).