A Survey on Malware Analysis Techniques: Static, Dynamic, Hybrid and Memory Analysis

Article *in* International Journal on Advanced Science Engineering and Information Technology · September 2018

DOI: 10.18517/ijaseit.8.4-2.6827

3 authors:

Rami Sihwail
Universiti Kebangsaan Malaysia
3 PUBLICATIONS **24** CITATIONS
SEE PROFILE

Khairuddin Omar
Universiti Kebangsaan Malaysia
175 PUBLICATIONS **1,580** CITATIONS
SEE PROFILE

Khairul Akram Zainol Ariffin
Universiti Kebangsaan Malaysia
20 PUBLICATIONS **75** CITATIONS
SEE PROFILE

International Journal on Advanced Science Engineering Information Technology

Vol.8 (2018) No. 4-2
ISSN: 2088-5334

A Survey on Malware Analysis Techniques: Static, Dynamic, Hybrid and Memory Analysis

Rami Sihwail[#], Khairuddin Omar[*], K. A. Z. Ariffin[*]

[#] *Supporting Studies Centre, King Faisal University, Al-Hasa, Saudi Arabia*
E-mail: rsihwail@kfu.edu.sa

**Faculty of Information Science & Technology, Universiti Kebangsaan Malaysia, Selangor, Malaysia*
E-mail: ko@ukm.edu.my, k.akram@ukm.edu.my

***Abstract*— The threats malware pose to the people around the world are increasing rapidly. A software that sneaks to your computer system without your knowledge with a harmful intent to disrupt your computer operations. Due to the vast number of malware, it is impossible to handle malware by human engineers. Therefore, security researchers are taking great efforts to develop accurate and effective techniques to detect malware. This paper offers an overall view and detailed survey for malware detection methods like signature-based and heuristic-based. The Signature-based is largely used today by anti-virus software to detect malware. It is fast and capable to detect known malware. However, it is not effective in detecting zero-day malware and is easily defeated by malware that use obfuscation techniques. Likewise, a considerable amount of legitimate files that are incorrectly classified as malware (false positive) and long scanning time are the major limitations of heuristic-based. Alternatively, memory-based analysis is a promising technique that gives a comprehensive view of malware and it is expected to become more popular in malware detection. This paper mainly focuses on the following areas: (1) providing an overview of malware types and malware detection methods, (2) discussing current malware analysis techniques, their findings and limitations, (3) studying the malware obfuscation, attacking and anti-analysis techniques, and (4) exploring the structure of memory-based analysis in malware detection. The methods of malware detection are compared with each other according to their techniques, selected features, accuracy rates, and their advantages and disadvantages. This paper aims to help the readers to have a comprehensive view of malware detection and discuss the importance of memory-based analysis in malware detection.**

***Keywords*— malicious; malware detection method; feature; behaviour-based; memory analysis; security.**

I. INTRODUCTION

The threat that malware (short for malicious software) cause to the computing world is growing rapidly. According to the AV-TEST institute, 48 million various malware samples were developed in the first quarter of 2017 [1].
Due to the vast number of malware, it is impossible to handle malware by human engineers. Thus, security researchers use malware detection systems to detect malware. Detection systems includes two stages: analysis and detection. Anti-virus software commonly use signature-based approach to detect malware. This approach is fast and capable to detect known malware with minimal false positive rate. However, signature-based fails to discover unknown malware and is easily defeated by malware that uses obfuscation techniques. On the other hand, behavior-based is another approach that is used in malware detection where suspicious files are executed in a controlled environment, monitored, and marked as malicious if their behaviors match with known malware behavior. Behavior-based is able to detect unknown malware and malware that use obfuscation techniques, but it is time consuming with considerable false positive rate [2].

Alternatively, memory-based is another approach that is becoming more popular in malware detection lately due to the wealth of information found in the dumped memory that can be used in investigating malicious activities [3].

The paper is organized as follows: In the next section, under material and method, we explain malware types, detection methods, analysis techniques, and an overview of related works, Section III discusses the future direction of

malware and the main sources of malware dataset. Finally, the conclusion of the survey.

II. Material and Method

This section provides an overview of malware types, malware detection methods, and analysis techniques.

A. Malware Types

Malware is a software that is inserted into the system without user knowlege. It can harm the computer system by compromising computer functions, stealing data or evading access controls. The following list presents the common categories of malware:

- Virus: A malicious software that duplicates itself by injecting its code into other programs. Virus can spread from one program to another and from one computer to another [4].
- Worms: Are malicious programs that replicate themselves in a computer and destroy the files and data on it. Worms might also encrypt files or send junk emails. Unlike viruses, worms carry themselves in their own containers [5].
- Trojan horse: While acting as a legitimate programs, Trojans perform unknown and unwanted activities [4]. Trojans allow attackers to gain access to the effective computer and extract user confidential information like password and banking details.
- Spyware: Spyware is a software that continuously spies on the users activities. It is used to gather information about the users like webpages regularly visited and credit card number without their knowledge, then sends that information back to the attackers [6].
- Rootkit: Rootkit is a collection of malicious software that is programmed to access a computer system and allow other types of malware to get into the system [7].
- Ransomware: A harmful software that allows the hacker to lock the computer and restrict the victim access to the vital information. Ransomware encrypts the important data on the infected computer or network then asks for payment to lift the restriction [8].
- Adware: Advertising-supported software is a type of malware that continuously brings advertisements to the computer. Usually adware is bundled with free downloaded software and applications like free playing games [9].
- Botnet: A malware that remotely controls a group of devices like PCs, smart phones and internet of things devices are infected and controlled by a cybercriminal. Botnet is typically used for spam emails campaigns or denial of service attacks. Users are often unaware that their systems are infected by a botnet malware [10].

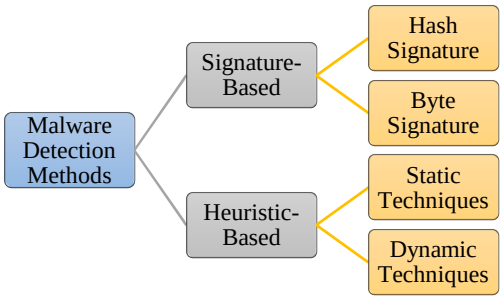

Fig. 1 Methods of Malware detection

B. Malware Detection Methods

Malware detection methods are categorized in several ways from different point of view. In this section, we discuss the main methods of malware detection: Signature-based and heuristic-based. Figure 1, shows the main malware detection methods.

1) Signature-Based Detection

Majority of available antivirus software use signature-based approach. This approach extracts unique signature from captured malware file and use this signature to detect similar malware. A signature is a sequence of bytes or a file hash that can be used to identify specific malware [11]. Therefore, this method has small false positive (FP) rate [14].. However, it is not difficult for attackers to change malware signature to evade being detected by antivirus software. Signature-based is very effective and fast in detecting known malware, but it is incapable to capture new released malware [13]. Signature-based approach depends on implementing static analysis to extract exceptional byte sequences known as marks [12]Figure 2 shows the signature-based general procedure for malware detection.

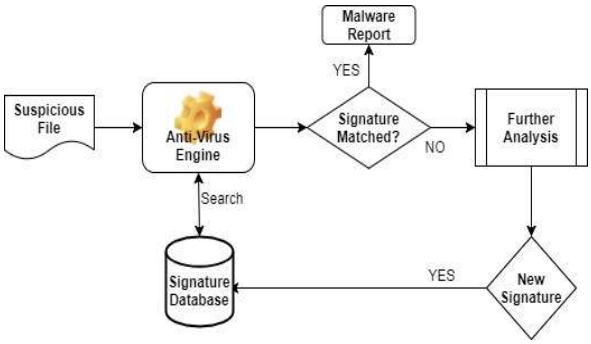

Fig. 2 Signature-based general flow

Malware authors have created another challenge for signature-based approach by using obfuscation techniques. This techniques include dead code insertion, register reassignment, instruction substitution, and code manipulation [15]. In the following we briefly explain each technique.

- Dead-Code insertion: This simple code obfuscation technique adds some NOP (No operation Performed) instructions or inserts ineffective PUSH/ POP statements to a program to change its look, but keep its same behavior.

- Register Reassignment: This technique works by switching registers or by reassigning the value of one register to unused one. For example, EAX is reassigned to EBX register.
- Subroutine Reordering: Subroutine is a group of program operations that do a specific task. This technique changes the subroutines order randomly in the program.
- Instruction Substitution: In this technique, original instructions that perform the same function are replaced by equivalent ones, such as replacing MOV instruction with PUSH instruction.
- Code Integration: A malware that embedded itself to another legal program. It was first found in Zmist malware. To apply this technique, malware decompiles its targeted program and adds itself in between its source code [16]. Code integration is considered as one of the most sophisticated obfuscation techniques that allows malware to evade detection.

2) Heuristic-Based Detection

Heuristic-based is also known as anomaly or behavior-based detection. In this detection, the activities performed by malware during runtime are analyzed in a training (learning) phase. After that, the file is labelled as malicious or legitimate file during a testing (monitoring) phase based on a pattern extracted during the training test [11].

Unlike signature-based, behavior-based approach is capable to detect both unknown malware and malware that uses obfuscation techniques. However, the major drawbacks of behavior-based are a considerable false positive rate (FP) and excessive monitoring time [14]. Further, the reduction of thousands of extracted features, evaluate similarities between them, and monitoring malware activities are directly effecting the ability of detecting zero-day malware attacks [17], [18].

Heuristic-based commonly depends on data mining techniques in order to understand the behaviors of running files, such techniques include Support Vector Machine, Naïve Bayes, Decision Tree and Random Forest.

C. Malware Analysis Techniques

Malware analysis concerns studying malicious files with the aim of having better understanding about several aspects of malware like malware behavior, evolution over time, and their selected targets [19]. The outcome of malware analysis should allow security firms to strengthen their defence strategies against malware attacks.

Techniques used for malware analysis mainly categorized into three parts: Static, Dynamic, and Hybrid analysis. In addition, memory-based analysis is another technique that is very useful in malware analysis. Figure 3, shows malware analysis techniques and their common features.

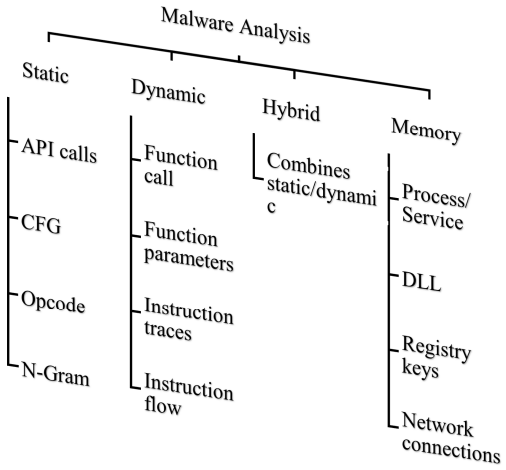

Fig. 3 Malware analysis techniques and features

1) Static Analysis

This technique refers to analyzing the Portable Executable files (PE files) without running them. Malware commonly uses binary packer, such as UPX and ASP Pack Shell, to avoid being analyzed [6]. A PE file needs to be unpacked and decompressed before being analyzed. To decompile windows executable file a disassembler tool can be used, such as IDA Pro and OlleyDbg that display assembly instructions, provide information about the malware, and extract pattern to identify the attacker.

The detection pattern can be extracted in static analysis like Windows API calls, string signature, control flow graph (CFG), opcode (operation codes) frequency and byte sequence n-grams [20]. In the following, we explain the main features in static analysis.

Almost all programs use Windows API (short for Application Programming Interface) calls to communicate with the operating system. For example, the "OpenFileW" is a Windows API in "Kernel32.dll" that creates a new file or opens an existing one. Therefore, API calls reveal the behavior of programs and could be considered as an essential mark in malware detection. For instance, the Windows API calls "WriteProcessMemory", "LoadLibrary" and "CreateRemoteThread" are a suspected behavior used by malware for DLL injection into a process, while rarely come together in a legitimate set. DLL injection is discussed in memory analysis section.

Strings are good indicator of malicious existence. Strings reveal the attacker's intent and goals since they often hold critical semantic information [6]. For example, the following string “This program cannot be run in DOS mode” indicates malicious file when it is found outside of the typical PE header, which is a common feature of droppers and installers.

Control Flow Graph (CFG): A CFG is a directed graph that demonstrates the control flow of a program, where blocks of code are presented by nodes and control flow paths by edges. In malware detection, CFG can be used to capture the behavior of a PE file and extract the program structure [19].

Opcodes is the first part of a machine code instruction (also called machine language) that identifies what operation to be executed by the CPU. A full machine language instruction composed of opcode and, optionally, one or more operands (e.g., "mov eax 7", "add eax ecx" and "sub ebx 1"). Opcode can be employed as a feature in malware detection by testing opcode frequency or calculating the similarity between opcode sequences.

N-grams are all of contiguous subsequences of a sequence of a length N [21]. For example, the word "MALWARE" is a sequence of letters of length 7, it can be segmented into 3-grams as: "MAL", "ALW", "LWA", "WAR" and "ARE". N-Grams have been applied with various detection features like API calls and opcodes.

Beside the previous features, there are other features that have been used in static analysis like file size and function length. Networking features like TCP/ UDP ports, destination IP and HTTP request are also features in static analysis [19].

One of the most significant research on malware signature evasion techniques has been done by Kirat and Vigna [22]. They were able to extract techniques from 2810 malware samples and group them into 78 similar evasion signature techniques.

Hashemi and Hamzeh presented a new approach that extracts unique opcode from the executable file and converts them into digital image. Visual features are then extracted from the image using Local Binary Pattern (LBP), which is one of the most famous texture extraction method in image processing. Finally, machine-learning methods are used to detect malware. The proposed detection technique obtained accuracy rate of 91.9% [23]. Shaid and Maarof also suggested displaying malware in the form of images. Their technique captures API calls of malware and converts them into visual cues or images. These images are used to identify malware variants [24].

On the other hand, both Salehi et al. [25] and Han et al. [26] built their techniques based on extracted API calls. Salehi et al. extracted API calls from each binary files and used API frequencies to learn the classifier. Then, three feature sets were generated ‘API calls list’, 'API arguments' and ‘API and arguments list’, and each set has been tested separately. Results showed that API arguments list is better compared to the other two sets with accuracy of 98.4% and false positive rate around 3%. In the same way, Han et al. extracted APIs from the IAT table (import Address Table) using static analysis. They compared the extracted API sequence with another sequence and calculated the similarity between them to classify malware family. Han found that malware within the same family are about 40% similar and false positive rate calculated 16%. Likewise, Cheng et al. [27] analyzed native APIs sequences using WinDbg tool and applied Support Vector Machine to detect shellcode malware. They used a too small training set, and were able to achieve 94.37% accuracy rate. However, false negative rate accounted as high as 44.44%.

Table I, shows the results of surveyed papers that applied static analysis in their malware detection approaches.

TABLE I
SURVEYED PAPERS THAT APPLY STATIC ANALYSIS

Author Year	Static feature	Classifier	Dataset Malware/ Benign	Acc	FP
Hashemi 2018 [23]	Opcode	KNN	M=3,100 B=3,100	91.9%	-
Salehi 2014 [25]	API, arguments	ROT-F, RF, DT, J48, NB	M=826 B=395	98.4%	3%
Han 2012 [26]	APIs sequence	-	M=545	40%*	16%
Santos 2013 [28]	Opcode sequence	DT, KNN, BN, SVM	M=1,000 B=1,000	97.5%	6%
Cheng 2017 [27]	Native APIs sequence	SVM	M=18/ B=72	94.4%	1.4%

* Similarity within the same family

2) Dynamic Analysis

It is also called behavior analysis. In this analysis, suspicious files are executed and monitored in a controlled environment like VM, emulator or simulator [9]. The infected files need to be analyzed in invisible environment for simple reason that some malware are supported with anti-virtual machine and anti-emulator techniques. Malware behave normally when they detect such environment and do not show any malicious activity.

Compared to static analysis, dynamic analysis is more effective as there is no need to disassemble the infected file to analyze it. In addition, dynamic analysis is able to detect known and unknown malware. Furthermore, obfuscated and polymorphic malware cannot evade dynamic detection. However, dynamic analysis is time intensive and resource consuming [6].

TABLE II
SURVEYED PAPERS THAT APPLY DYNAMIC ANALYSIS

Author Year	Dynamic feature	Classifier	Dataset Malware/ Benign	Acc	FP
Liang 2016 [32]	API calls	DT, ANN, SVM	M=12,199	91.3%	-
Mohaisen 2013 [29]	file system, registry, network	SVM, DT, KNN	M=1,980	95%	5%
Mohaisen 2015 [30]	file system, registry, network	SVM, DT, KNN	M=115,000	99%	-
Galal 2017 [33]	APIs sequence	DT, RF, SVM	M=2,000/ B=2,000	97.2%	-
Ki 2015 [34]	APIs sequence	-	M=23,080	99.8%	0%
Fan 2015 [35]	User API, native API	J48, NB, SVM	M=773/ B=253	95.9%	5%

Various techniques can be used with dynamic analysis, such as function call monitoring, function parameter analysis, instruction traces, and information flow tracking [20]. Reviewing the surveyed papers, API and system calls are largely employed in malware dynamic analysis as well as file system, Windows registry and network features.

Mohaisen et al. tried to classify Zeus malware using several machine learning techniques. Artifacts like registry, file system, and network features were used to learn the

classifier [29]. The dataset consisted of 1980 samples of Zeus Banking Trojan and accuracy achieved close to 95%. Afterward, in the next work, Mohaisen et al. proposed AMAL, an automated and behavior-based malware analysis and labeling system. AMAL consists of two components: AutoMal and AutoLabel. Automal uses file system, network activity logging, and registry monitoring features to analyze malware samples. Further, AutoLabel classifies malware samples into their families based on their behavior. AMAL used more than 115,000 malware samples and achieved detection rate around 99% [30].

In their work [31], Chen and Bridges studied WannaCry Ransomware features from system logs, which is produced using Cuckoo Sandbox. TF-IDF approach, shorts for term frequency–inverse document frequency, has been used to calculate frequent terms with high weights in the system logs.

Most of the dynamic techniques focused on API calls to represent malware behaviors (e.g. [32], [33], [34]–[35]). Liang et al. [32] introduced a behavior-based malware variant classification technique that captures API calls of running malware, then creates multilayer dependency chain based on the dependency relationship of the API calls. The technique is able to measure the degree of similarity between malware variants. Galal et al. also applied API hook to capture information about API calls and their parameters. Then, related API calls that share common semantic purposes are set together into sequences. Their highest accuracy was 97.19% achieved using Decision Tree [33]. Likewise, Ki et al. [34] proposed an approach that extracts user level API call sequences by using, Microsoft supported tool, Detours and apply Multiple Sequence Alignment algorithm (MSA), which is one of the most popular algorithms used in DNA sequence alignment. After that, Ki et al. applied Longest Common Subsequence algorithm (LCS) to match similar sequences. The approach achieved 99.8% accuracy and zero (0) false positive. Further, Fan et al. [35] used API hooking to trace APIs that malware try to hide. The technique monitors both regular APIs and native APIs like undocumented and low-level APIs. In the experiment, only 80 APIs were selected and detection rate reached 95% using Decision Tree and Naive Bayesian algorithms.

Table II, shows the results of surveyed papers that applied dynamic analysis in their malware detection approaches.

In dynamic analysis, malware are executed in a controlled environment to examine the live behavior of malicious files without being harmed by them. There are several types of control environment like emulators, debuggers, simulators and virtual machines. Next, we present each type and explain the strategies malware use in order to detect the existence of controlled environment.

Emulator is a controlled environment that is used to control the execution of a malicious program. A full emulation system controls the CPU, hard disk and resources. Emulators are distinguished based on the controlled part of the running environment. TEMU, which is part of BitBlaze project, introduced in 2008 by Sont et al. [36] as a full emulation system that supports dynamic binary analysis by monitoring features like network activities, memory locations, function calls, processes, modules and API calls. TTAnaylze [37] is another type of emulators that works on QEMU, which is an open source machine emulator, and provides automatic malware analysis module that records windows APIs and native APIs. However, majority of malware are able to detect emulated environment. In case of partial emulation system, malware can perform operation that works outside the emulated environment to detect whether it is running inside a controlled environment. Further, malware can still detect the characteristics and side effects of full environment system like detecting imperfect CPU features and comparing system properties (i.e. currently logged-in user) [38].

Debugger is another type of controlled environment, which is a program that observes and examines the execution of other binary programs. WinDbg, OllyDbg and GDB are debuggers that can be used to monitor the execution behavior of suspected binaries at the instruction level. Unlike OllyDbg, WinDbg also supports kernel debugging. Further, IDA Pro is a static analysis tool that has less capable built-in debugger. Though, The use of Windows API is the most straightforward technique malware use to determine that it is being debugged. API functions that can be used for anti-debugging include “IsDebuggerPresent”, “CheckRemoteDebuggerPresent” and “OutputDebugString”. Another technique performed by malware is to look for signs of installing debugging tool on the system such as searching registry keys, files and directories. Further, malware can use several techniques like exceptions and interrupts to disrupt the execution of a program only if it is being debugged [40].

Another environment is simulator, which is a program that simulates operation in order to be observed by user without actually performing that operation. Simulator tools such as CWSandbox, Norman sandbox and Detours allow malware to execute in a controlled virtual environment and record its behavior. Detours is used to intercept function calls made by a process to any DLL (DLL injection), while CWSandbox performs API hooking to capture Windows API calls invoked by a malware. On the other hand, Norman sandbox simulates Windows operating system, LAN and Internet connectivity on the host machine [38]. For anti-simulation, Malware checks for registry, files or processes to determine the existence of certain sandbox product. The execution time is another technique to detect sandbox and virtual environment as executing instruction under controlled environment requires longer time than a real one [41].

The most common controlled environment is virtual machine (VM). VM is a computer software that runs an operating system and applications. These applications are isolated from the host system. Thus, running file or software inside a virtual machine cannot interfere with the host machine. Virtual machine applications include VirtualBox, Parallels and VMware. A virtual machine monitor (VMM) is a software that creates, runs and manages virtual machine

[39]. Furthermore, it is also responsible for assigning hardware to virtual machine. However, Malware examines the existence of virtual machine (VM) on a system by searching for artifacts that installed VM tools leave in the file system, registry and process listing. Malware can also look for certain instructions that can be invoked by user mode such as "sidt", "sgdt", and "sldt" to observe the presence of VM tools [40]. Furthermore, Hardware characteristics and features may lead to the presents of virtual machine. For example, CPUID hypervisor bit is set to zero in the real system and malware, therefore, can test this bit to determine if they are running inside a virtual machine. In addition, most debuggers and Virtual Machines create files and drivers that belong to that particular tool, malware can look for these artifacts to discover the presence of virtual machines or debuggers [41].

3) *Hybrid Analysis*

Hybrid analysis gather information about malware from static analysis and dynamic analysis. By using hybrid analysis, security researchers gain the benefits of both analyses, static and dynamic. Therefore, increasing the ability of detecting malicious programs correctly [42]. Both analyses have their own advantages and limitations. Static analysis is cheap, fast and safer compared to dynamic analysis. However, malware evade it by using obfuscation techniques. On the other hand, dynamic analysis is reliable and can beats obfuscation techniques. Furthermore, it is able to recognize malware variants and unknown malware families. However it is time intensive and resource consuming [6].

Shijo and Salim [43] proposed an integrated technique to detect and classify unknown files. Printable strings information (PSI) feature was extracted by performing static analysis. Besides, using dynamic analysis to extract API calls. Experiment showed detection rate of 95.8% applying static, 97.1% applying dynamic and 98.7% for hybrid analysis. Their highest accuracy was achieved using SVM technique. Islam et al. [44] extracted two features from static analysis Function Length Frequency (FLF) and Printable String information (PSI) and API calls and parameters during dynamic analysis. Based on the results, Random Forest machine learning technique showed the highest result in classifying the data. In addition, they have found that applying the approach on old malware samples has better accuracy compared to new samples, with accuracy of 99.8% and 97.1% respectively. Further, Ma et al. [45] introduced a method to reduce false positive in malware classification called Ensemble that combined static and dynamic classifier into one classifier. The method uses multi features include static import functions and dynamic call functions to improve the accuracy and reduce false positive. Furthermore, Santos et al. [46] introduced OPEM, a tool to detect unknown malicious files by combining opcode frequency obtained during static analysis and system calls, operations and raised exceptions during dynamic analysis. OPEM showed accuracy of 95.9% from static analysis, 77.26% using dynamic and 96.6% using hybrid analysis with SVM.

Table III, shows the results of surveyed papers that applied hybrid analysis in their malware detection approaches.

TABLE III
SURVEYED PAPERS THAT APPLY HYBRID ANALYSIS

Author Year	Feature Static/ Dynamic	Classifier	Dataset Malware/ Benign	Acc	FP
Shijo 2016 [43]	PSI/ API calls	RF, SVM	M=1,368 B=456	98.7%	-
Islam 2013 [44]	Function length, PSI/ API	DT, SVM, RF, IB1	M=2,939	97%	5.1%
Ma 2016 [45]	Import functions/ call functions	DT, NB, SVM	M=279	-	-
Santos 2013 [46]	Opcode/ system calls, operations, and exceptions	DT, KNN, NB, SVM	M=13,189 B=13,000	96.6%	3%

4) *Memory Analysis*

Memory analysis has become a popular technique, in the recent years, proven to be efficient and accurate in malware analysis. Memory analysis attracts malware analysts as it gives comprehensive analysis of malware [47] since it is able to examine malware hooks and code outside the function normal scope [48]. It uses memory image to analyze information about running programs, operating system, and the general state of the computer.

Memory forensics investigations pass through two steps: memory acquisition and memory analysis. In the memory acquisition, the memory of the target machine is dumped to obtain a memory image using tools such as Memoryze, FastDump and DumpIt. The memory analysis step is to analyze the memory image looking for malicious activities using tools like Volatility and Rekall.

A number of researches related to memory forensics techniques have been proposed. Teller and Hayon [50] proposed a trigger-based memory analysis approach that triggers memory dumps based on the following events: API-based, performance-based and instrumentation-based in order to know what happens during the execution of the malware file and not only at the end of it. Further, Choi et al. [51] introduced a modification to Teller and Hayon's technique in [50]. By implementing API trigger-based memory dump technology for Cuckoo sandbox, which makes Cuckoo capable to dump the memory at every wanted API call.

In their research, Mosli et al. [3] investigated three features extracted from memory images: imported libraries, registry activity, and API function calls to detect malware. Their highest accuracy was about 96% using SVM with data from registry activity. Afterward, in the next research, Mosli et al. [52] proposed a technique that uses the process handles to detect whether the suspected sample is malicious or benign. The experiment have spotted the light on the main types of handles that malicious process commonly use, such

as section handles, process handles and mutants. On the other hands, Zaki and Humphrey [7] studied the memory artifacts left by rootkits in kernel-level such as: driver, module, SSDT hook, IDT hook and callback. The experiment has proven that callback functions, modified drivers, and attached devices are the most suspicious activities in the kernel-level. Table IV, shows the results of surveyed papers that applied memory analysis in their malware detection approaches.

TABLE IV
SURVEYED PAPERS THAT APPLY MEMORY ANALYSIS

Author Year	Feature	Classifier	Dataset Malware/ Benign	Acc	FP
Mosli 2016 [3]	Registry, imported libraries, API calls	SVM, SGD, DT, RF, KNN	M=400 B=100	96%	-
Mosli 2017 [52]	Number of opened handles	KNN, SVM, RF	M=3,130 B=1,157	91.4%	-
Zaki 2014 [7]	Driver, module, hooks, callback	-	-	-	-

In addition, Memory forensic techniques are able to monitor malware behaviors like API hooking, DLL injection and Hidden processes [49]. In the following, we discuss each behavior and malware anti-forensics techniques.

Windows API is generally used to communicate with system resources like files, processes, registry and network [53]. Malware use a technique called API hooking to interrupt the function calls. In other words, it can change the behavior and flow of API calls. Following is the most common types of API hooks [54] [55]:

- IAT hooks: A PE file stores the address of API functions in the Import Address Table (IAT). Malware overwrites the location of API in the IAT, thus forcing the process to call an attacker function instead of the original API. Many well-known malware are using IAT hook like Zeus, FinFisher and Stuxnet.
- Inline API hooks: Also called trampoline or detours hooks. Inline hooks require writing in few places in process's memory. For example, Malware adds jump instruction (JMP) into a legitimate function prologue to move the flow to a different memory location that is occupied by a rootkit.
- IDT hook: The Interrupt Descriptor Table (IDT) stores functions for handling interrupts and exceptions. Malware changes the value in 0x2E entry in the IDT and gain control when a call to a kernel mode API function is executed.
- SSDT hooks: The System Service Descriptor Table (SSDT) is a table containing pointers to kernel mode functions. Malware uses SSDT to protect and hide themselves. For example, by hooking the SSDT malware can negate the call to "NTOpenProcess" and, therefore, no program will be able to kill the malicious process.
- IRP hook: Applications in Windows communicate with drivers through Input/ Output Request Packet (IRP). Malware uses IRP hook to do malicious actions such as keylogging and disk access filtering.

DLL injection is a technique aiming to insert a malicious code into a legitimate process. Once the malicious DLL is injected, the execution flow is transferred to the malicious memory space [56]. The DLL injection can be categorized into the following main techniques:

- Classic DLL injection using remote thread: DLL injection is a common technique malware use to inject malicious code into another process. Malware write the Dynamic Link Library (DLL) path in the virtual address space of the target process, and creates remote thread in the target process to make sure that it loads the malicious DLL. The malware first calls the API "VirtualAllocEx" to assign memory into the address space of the victim, 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 [57].
- Injecting via registry modification: Hackers modify the value of "Appinit_DLL" registry key that is located at "HKLM\Software\Microsoft\WindowsNT\CurrentVersion\Windows\Appinit_Dlls", to conduct DLL injection. Then, the location of the malware library is added to "Appinit_DLL" to make another process load their library. To apply this, malware calls "RegCreateKeyEx" to open the "Appinit_Dlls" registry key, and then calls "RegSetValueEx" to modify its values. Additional, rebooting the system is needed to apply the modifications in the registry values to the system [58].
- Injection using window hooking function: In Windows application, program code is executed based on events. Malware can load their malicious DLL whenever certain event is triggered. The "SetWindowsHookEx" function is used to install a hook routine into the hook chain. Thus, the malicious action inputted into the "SetWindowsHookEx" function is called whenever a particular event is triggered such as mouse move or key press [56].

The existence of hidden processes, files or network connections is a good indicator of a successful malicious attack [59]. Therefore, attackers try to hide their malicious artifacts. Hiding a process is typically accomplished by rootkit called stealth rootkit, which modifies program binaries. Another method is to hook the call path between applications and the kernel by modifying system call tables, dynamic link structures, libraries, or operating system functions [60]. Further, some rootkits modify the kernel data structures using direct kernel object manipulation (DKOM) leading to incorrect user requests and, therefore, unrevealing the existence of malicious process [61].

In order to prevent memory forensics or make it unworkable, malware use anti-forensic techniques. The following are some of anti-forensic techniques:

- Memory hiding: Malware tries to hide malicious memory region form memory analysis tools.

- Memory acquisition failure: By terminating certain processes or limiting driver loading to thwart memory acquisition process.
- Anti-Carving: Malware attempts to prevent memory analysts from extracting kernel information by manipulating the kernel data structure field in order to make the memory analysis tools mislead the field [62].
- Increase timing: Malware creates fake objects in the kernel data structure to increase analysis time.

III. Results and Discussion

A. Future Direction

Many security researchers believe that the future of malware still ambiguous. There are a number of challenges in the future of malware development in which, we believe, security firms and researchers should consider. First, one worry is the automation of creating malware variants. Studying the latest malware detection methods and using machine learning, attackers can develop automated tools that are able to produce thousands of different malware samples every day. Second, malware groups may offer those malware automation tools for rental or sale, giving the chance to low-skilled groups and amateur hackers to enter malware world. Third, Malware are rapidly change in terms of structure and functionality. Most of the surveyed techniques used one malware dataset to learn and test the behaviors (the classifier). Although they have got a high detection rate, but results would be different when applying the techniques on a new released malware. Finally, malware are expected to become more complicated in the future. Attackers might use a new encryption methods or obfuscation techniques to make malware detection and analysis an impossible job.

The traditional way anti-virus software use to capture malware is by searching for known signature. Unfortunately, this technique can easily be evaded by simple obfuscation technique [63]. Static and dynamic analyses have their limitations as well. Alternatively, memory analysis gives comprehensive analysis of malware. Malware can hide its code in the computer system effectively. However, malware must execute its code in the memory to perform its tasks eventually. Volatile memory (RAM) keeps its contents until it is powered off. Therefore, analyzing the RAM can tell us about the activities which is happening in the system. Valuable live Information that resides in memory include running process, Dynamic Link Library (DLL), files, registry keys, services, sockets and ports, and active network connections [64]. Thus, memory analysis is a promising technique that is expected to become more popular, together with data mining and machine learning techniques, in malware detection.

B. Dataset

In order to study malware techniques and tricks, it is very important for researchers to collect malware samples. One way to collect samples is by using honeypots, which is a dedicated machine deployed to attract attackers to learn their attacking techniques [65]. Researchers can also use known malicious URLs. In addition, malware dataset can be downloaded from anti-malware agents' websites such as Malware DB, Malwr, MalShare, VX Heaven, theZoo and VirusShare malware repository. Furthermore, some specialized companies and research project groups occasionally share their collection of malware datasets. In 2015, Microsoft provided 500 GB dataset of known malware files in the big challenge competition [66]. Recently, Endgame is sharing “ember” project with 600 thousands malicious files to address the lack of open-source datasets in the domain of static detection malware [67].

IV. Conclusion

Malware is causing a critical threat to our computer systems, internet and data. The challenges that malware authors pose by developing complicated malware that frequently changes their signature to evade detection, and by releasing more sophisticated versions of malware that use new obfuscation techniques, have brought many issues to anti-virus software and security researchers. In this paper, we briefly surveyed malware types and malware detection methods. We have also reviewed three types of malware analysis techniques: static, dynamic and hybrid. We also gave a discussion on the use of memory forensics in finding malware artifacts. In addition, we discussed the future of memory-based analysis in malware detection. Techniques used by malware to evade detection such as obfuscation, attacking and anti-analysis techniques have been reviewed as well. Finally, the future direction of malware development and the main sources of malware dataset have been studied in this paper.

Acknowledgment

The authors would like to thank the Ministry of Education Malaysia for supporting this work under grant FRGS/1/2016/ICT02/UKM/01/1. Also, would like to thank Universiti Kebangsaan Malaysia (UKM) for supporting this work under grant GGPM-2017-026.

References

[1] AV-TEST, “The AV-TEST Security Report,” 2017. [Online]. Available: https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2016-2017.pdf.

[2] C. T. Lin, N. J. Wang, H. Xiao, and C. Eckert, “Feature selection and extraction for malware classification,” J. Inf. Sci. Eng., vol. 31, no. 3, pp. 965–992, 2015.

[3] 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, 2016, pp. 1–6.

[4] M. Karresand, “Separating Trojan horses, viruses, and worms - A proposed taxonomy of software weapons,” in IEEE Systems, Man and Cybernetics Society Information Assurance Workshop, 2003, pp. 127–134.

[5] X. Wang, W. Yu, A. Champion, X. Fu, and D. Xuan, “Detecting worms via mining dynamic program execution,” in Proceedings of the 3rd International Conference on Security and Privacy in Communication Networks, SecureComm, 2007, pp. 412–421.

[6] Y. Ye, T. Li, D. Adjeroh, and S. S. Iyengar, “A Survey on Malware Detection Using Data Mining Techniques,” ACM Comput. Surv.,

vol. 50, no. 3, pp. 1–40, 2017.
[7] A. Zaki and B. Humphrey, “Unveiling the kernel : Rootkit discovery using selective automated kernel memory differencing,” Virus Bull., no. September, pp. 239–256, 2014.
[8] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, “CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data,” in Proceedings - International Conference on Distributed Computing Systems, 2016, vol. 2016–Augus, pp. 303–312.
[9] G. A. N. Mohamed and N. B. Ithnin, “Survey on Representation Techniques for Malware Detection System,” Am. J. Appl. Sci., vol. 14, no. 11, pp. 1049–1069, 2017.
[10] M. Chowdhury and A. Rahman, “Malware Analysis and Detection Using Data Mining and Machine Learning Classificatio,” in International Conference on Applications and Techniques in Cyber Security and Intelligence, 2018, vol. 580, pp. 266–274.
[11] A. Damodaran, F. Di Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, “A comparison of static, dynamic, and hybrid analysis for malware detection,” J. Comput. Virol. Hacking Tech., vol. 13, no. 1, pp. 1–12, 2017.
[12] A. Souri and R. Hosseini, “A state-of-the-art survey of malware detection approaches using data mining techniques,” Human-centric Computing and Information Sciences, vol. 8, no. 1. 2018.
[13] M. Alazab, S. Venkataraman, and P. Watters, “Towards understanding malware behaviour by the extraction of API calls,” Proc. - 2nd Cybercrime Trust. Comput. Work. CTC 2010, no. July 2009, pp. 52–59, 2010.
[14] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh, “A survey on heuristic malware detection techniques,” in IKT 2013 - 2013 5th Conference on Information and Knowledge Technology, 2013, pp. 113–120.
[15] I. You and K. Yim, “Malware obfuscation techniques: A brief survey,” in Proceedings - 2010 International Conference on Broadband, Wireless Computing Communication and Applications, BWCCA 2010, 2010, pp. 297–300.
[16] W. Wong and M. Stamp, “Hunting for metamorphic engines,” J. Comput. Virol., vol. 2, no. 3, pp. 211–229, 2006.
[17] M. Hafiz, M. Yusof, and M. R. Mokhtar, “A Review of Predictive Analytic Applications of Bayesian Network,” Int. J. Adv. Sci. Eng. Inf. Technol., vol. 6, no. 6, pp. 857–867, 2016.
[18] S. N. Das, M. Mathew, and P. K. Vijayaraghavan, “An Approach for Optimal Feature Subset Selection using a New Term Weighting Scheme and Mutual Information,” Int. J. Adv. Sci. Eng. Inf. Technol., vol. 1, no. 3, pp. 273–278, 2011.
[19] D. Ucci, L. Aniello, and R. Baldoni, “Survey on the Usage of Machine Learning Techniques for Malware Analysis,” arXiv Prepr. arXiv1710.08189, pp. 1–67, 2018.
[20] E. Gandotra, D. Bansal, and S. Sofat, “Malware Analysis and Classification: A Survey,” J. Inf. Secur., vol. 05, no. 02, pp. 56–64, 2014.
[21] T. Abou-Assaleh, N. Cercone, V. Keselj, and R. Sweidan, “N-gram-based detection of new malicious code,” Proc. 28th Annu. Int. Comput. Softw. Appl. Conf. 2004. COMPSAC 2004., vol. 2, no. 1, pp. 41–42, 2004.
[22] D. Kirat and G. Vigna, “MalGene,” in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS ’15, 2015, pp. 769–780.
[23] H. Hashemi and A. Hamzeh, “Visual malware detection using local malicious pattern,” Journal of Computer Virology and Hacking Techniques, pp. 1–14, 2018.
[24] S. Z. M. Shaid and M. A. Maarof, “Malware behaviour visualization,” J. Teknol., vol. 70, no. 5, pp. 25–33, 2014.
[25] Z. Salehi, A. Sami, and M. Ghiasi, “Using feature generation from API calls for malware detection,” Comput. Fraud Secur., vol. 2014, no. 9, pp. 9–18, 2014.
[26] K. S. Han, I. K. Kim, and E. G. Im, “Malware classification methods using API sequence characteristics,” in Lecture Notes in Electrical Engineering, 2012, vol. 120 LNEE, pp. 613–626.
[27] Y. Cheng, W. Fan, W. Huang, and J. An, “A Shellcode Detection Method Based on Full Native API Sequence and Support Vector Machine,” in IOP Conference Series: Materials Science and Engineering, 2017, vol. 242, no. 1, pp. 1–7.
[28] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, “Opcode sequences as representation of executables for data-mining-based unknown malware detection,” Inf. Sci. (Ny)., vol. 231, pp. 64–82, 2013.
[29] A. Mohaisen and O. Alrawi, “Unveiling Zeus: automated classification of malware samples,” Proc. 22nd Int. Conf. World Wide Web companion, pp. 829–832, 2013.
[30] A. Mohaisen, O. Alrawi, and M. Mohaisen, “AMAL: High-fidelity, behavior-based automated malware analysis and classification,” Comput. Secur., vol. 52, pp. 251–266, 2015.
[31] Q. Chen and R. A. Bridges, “Automated Behavioral Analysis of Malware A Case Study of WannaCry Ransomware,” arXiv Prepr. arXiv1709.08753, pp. 1–9, 2017.
[32] G. Liang, J. Pang, and C. Dai, “A Behavior-Based Malware Variant Classification Technique,” Int. J. Inf. Educ. Technol., vol. 6, pp. 291–295, 2016.
[33] H. S. Galal, Y. B. Mahdy, and M. A. Atiea, “Behavior-based features model for malware detection,” J. Comput. Virol. Hacking Tech., vol. 12, no. 2, pp. 59–67, 2016.
[34] Y. Ki, E. Kim, and H. K. Kim, “A novel approach to detect malware based on API call sequence analysis,” Int. J. Distrib. Sens. Networks, vol. 2015, no. 6: 659101, pp. 1–9, 2015.
[35] C.-I. Fan, H.-W. Hsiao, C.-H. Chou, and Y.-F. Tseng, “Malware Detection Systems Based on API Log Data Mining,” in 2015 IEEE 39th Annual Computer Software and Applications Conference, 2015, pp. 255–260.
[36] D. Song et al., “BitBlaze: A new approach to computer security via binary analysis,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2008, vol. 5352 LNCS, pp. 1–25.
[37] U. Bayer et al., “Dynamic analysis of malicious code,” J Comput Virol, vol. 2, pp. 67–77, 2006.
[38] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, “A survey on automated dynamic malware-analysis techniques and tools,” ACM Comput. Surv., vol. 44, no. 2, pp. 1–42, 2012.
[39] Microsoft Azure, “What is a virtual machine?,” 2018. [Online]. Available: https://azure.microsoft.com/en-in/overview/what-is-a-virtual-machine/.
[40] M. Sikorski and A. Honig, Practical malware analysis: the hands-on guide to dissecting malicious software. no starch press. 2012.
[41] X. Chen, J. Andersen, Z. Morley Mao, M. Bailey, and J. Nazario, “Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware,” in Proceedings of the International Conference on Dependable Systems and Networks, 2008, pp. 177–186.
[42] M. Eskandari, Z. Khorshidpour, and S. Hashemi, “HDM-Analyser: a hybrid analysis approach based on data mining techniques for malware detection,” J. Comput. Virol. Hacking Tech., vol. 9, no. 2, pp. 77–93, 2013.
[43] P. V. Shijo and A. Salim, “Integrated static and dynamic analysis for malware detection,” in Procedia Computer Science, 2015, vol. 46, pp. 804–811.
[44] 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.
[45] X. Ma, Q. Biao, W. Yang, and J. Jiang, “Using multi-features to reduce false positive in malware classification,” in Proceedings of 2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference, ITNEC 2016, 2016, vol. 3, pp. 361–365.
[46] I. Santos, J. Devesa, F. Brezo, J. Nieves, and P. G. Bringas, “OPEM: A static-dynamic approach for machine-learning-based malware detection,” in Advances in Intelligent Systems and Computing, 2013, vol. 189 AISC, pp. 271–280.
[47] C. Rathnayaka and A. Jamdagni, “An efficient approach for advanced malware analysis using memory forensic technique,” Proc. - 16th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. 11th IEEE Int. Conf. Big Data Sci. Eng. 14th IEEE Int. Conf. Embed. Softw. Syst., pp. 1145–1150, 2017.
[48] J. Stüttgen and M. Cohen, “Anti-forensic resilient memory acquisition,” in Digital Investigation, 2013, vol. 10, no. SUPPL., pp. 105–115.
[49] 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 Conference on Dependable and Secure Computing, 2017, pp. 518–519.
[50] T. Teller and A. Hayon, “Enhancing Automated Malware Analysis Machines with Memory Analysis Report,” Black Hat USA, 2014.
[51] and K.-W. P. Choi, Sang-Hoon, Yu-Seong Kim, “Toward Semantic Gap-less Memory Dump for Malware Analysis,” ICNGC Conf., pp. 1–4, 2016.

[52] R. Mosli, R. Li, B. Yuan, and Y. Pan, “A behavior-based approach for malware detection,” in IFIP Advances in Information and Communication Technology, 2017, vol. 511, pp. 187–201.
[53] G. Willems, T. Holz, and F. Freiling, “Toward automated dynamic malware analysis using CWSandbox,” IEEE Security and Privacy, vol. 5, no. 2. pp. 32–39, 2007.
[54] M. H. Ligh, S. Adair, B. Hartstein, and M. Richard, Malware analyst’s cookbook and DVD: tools and techniques for fighting malicious code. Wiley Publishing, 2011.
[55] Adlice Software, “Rootkits hooks,” 2014. [Online]. Available: https://www.adlice.com/.
[56] S. Kim, J. Park, K. Lee, I. You, and K. Yim, “A Brief Survey on Rootkit Techniques in Malicious Codes,” J. Internet Serv. Inf. Secur., vol. 3, no. 4, pp. 134–147, 2012.
[57] 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.
[58] J. Berdajs and Z. Bosnic, “Extending applications using an advanced approach to DLL injection and API hooking,” Softw. - Pract. Exp., vol. 40, no. 7, pp. 567–584, 2010.
[59] J. Butler, J. L. Undercoffer, and J. Pinkston, “Hidden processes: The implication for intrusion detection,” in IEEE Systems, Man and Cybernetics Society Information Assurance Workshop, 2003, pp. 116–121.
[60] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, “VMM-based hidden process detection and identification using Lycosid,” in Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments - VEE ’08, 2008, pp. 91–100.
[61] A. Schuster, “Searching for processes and threads in Microsoft Windows memory dumps,” Digit. Investig., vol. 3, no. SUPPL., pp. 10–16, 2006.
[62] K. Lee, H. Hwang, K. Kim, and B. N. Noh, “Robust bootstrapping memory analysis against anti-forensics,” Digit. Investig., vol. 18, pp. S23–S32, 2016.
[63] A. Moser, C. Kruegel, and E. Kirda, “Limits of static analysis for malware detection,” in Proceedings - Annual Computer Security Applications Conference, ACSAC, 2007, pp. 421–430.
[64] J. Okolica and G. Peterson, “A compiled memory analysis tool,” in IFIP Advances in Information and Communication Technology, 2010, vol. 337 AICT, pp. 195–204.
[65] V. ATLURI, Anoop Chowdary; TRAN, Botnets threat analysis and detection. Cham, 2017.
[66] Endgame, “Ember,” 2018. [Online]. Available: https://www.endgame.com/blog/technical-blog/introducing-ember-open-source-classifier-and-dataset.
[67] Microsoft, “Microsoft Malware Classification Challenge (BIG 2015),” 2015. [Online]. Available: https://www.kaggle.com/c/malware-classification.