Tools & Techniques for Malware Analysis and Classification

Ekta Gandotra, Divya Bansal
and
Sanjeev Sofat
PEC University of Technology, Chandigarh, India

Ever-evolving malware continues to flood the Internet at an alarming rate. This makes it challenging for security organizations and anti-malware vendors to devise effective solutions. It is, therefore, imperative to study automated tools and techniques for quick detection of malware, possibly limiting or preventing any impact on the target. The code or behavioural patterns obtained from malware analysis can be used to classify new malware samples into their existing families and recognize those which possess unknown behaviour and thus need a closer manual inspection. This paper provides a comprehensive review of techniques and tools currently employed for malware analysis and classification. It includes the comparison of tools and techniques for collecting malware, analyzing them statically and dynamically for extracting features and finally classifying these using machine learning methods. It also provides the examples from the literature that analyze executables for extracting useful features and apply machine learning for discriminating malicious software from benign ones.

Keywords: Malware Analysis, Classification, Machine Learning, Automated Analysis.

1. INTRODUCTION

In this era, the Internet is an indispensable part of day-to-day life. People have become habitual of services being provided by the Internet which includes online banking, shopping, social networking, finance, education etc. At the same time, there exist some people who use the Internet with profane intentions and endeavor to augment themselves through malevolent activities which are achieved using malicious programs called malware. These are becoming sophisticated, persistent and unknown day by day. The security systems like IDS/IPS and antivirus (AV) use signatures for detecting malware which are created manually and require the malware samples to be analyzed minutely. McAfee[1] catalogs about 69 new malware samples every minute. According to Symantec Threat Report (2016)[2], 430 million new malicious samples are discovered in the year 2015, which is about 36% more than those discovered in 2014. A report by Trendmicro[3] says that the malware specimens are growing exponentially in volume (growing threat landscape), variety (innovative malicious methods) and velocity (fluidity of threats) . These are evolving, becoming more sophisticated and using new ways to target computers and mobile devices. It has now become possible for malware authors to create new malicious specimens within seconds through the readily available online tools.

According to Bayer et al. [2006], a malware is a piece of software program that accomplishes the destructive intent of an attacker. There are different classes of malware depending upon their characteristics like the ways they infect the systems, get propagated etc. The most common classes of malware are Worm, Virus, Trojan-horse, Spyware, Backdoor, Rootkit, Botnet and Adware etc. The most popular ways through which a system is infected by malware are: Drive-by Download, Social Engineering and Exploiting Network Services Vulnerabilities. Advanced

[1]http://www.scmagazine.com/the-state-of-malware-2013/slideshow/1255
[2]https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf
[3]http://www.trendmicro.com/cloud-content/us/pdfs/business/white-papers/wp_addressing-big-data-security-challenges.pdf

Persistent Threats (APTs) violation against social, political, economic and military networks necessitates the continual insight to mitigate risks and ensure resilience for securing the nation. In order to detect the signs of malware on the network, it is essential to understand the ways by which it gets into the network, spreads within, and as a final point starts moving data out of it. The whole Cyber-attack sequence typically follows a kill chain cycle (Figure 1) as published by Cloppert [2009].

—Reconnaissance: In this step, the attacker tries to understand the internal structure of the target organization. He tries to collect the information that can be used for social engineering attacks.

—Weaponization: The attacker makes efforts to find vulnerabilities in devices of target organization and the ways to exploit them, so that, he can get control of those devices.

—Delivery: After finding vulnerabilities and the ways to exploit these, the payload is delivered to the target user by making her to click on a link or visit a website containing malware.

—Exploitation: In this step, the exploit code is executed on the target organization by the attacker to get the control. This process may be a multistage process in which first a downloader gains control of the machine of target organization and then downloads other exploit code.

—Command and Control (C&C): In this step, the compromised device establishes contact with its control network to receive and execute further instructions.

—Exfiltration: This is the step in which the attacker starts to pick the stolen data.

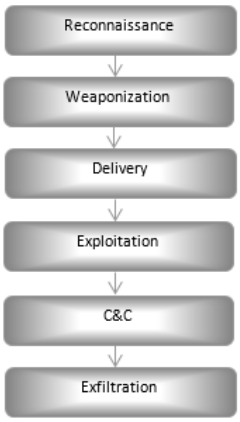

Figure 1. Cyber Attack Progression

In the initial phases of the kill chain (in Reconnaissance and Weaponization phases), the issues are difficult to observe, whereas the events that occur in the subsequent phases (starting from delivery phase) are easy to identify and analyze because the evidences reside on the system that has been compromised. These evidences are, in fact, very useful in detecting zero-day malware. The process of malware detection involves 3 major steps:

—Malware Acquisition

—Malware Analysis

—Malware Classification

This paper aims to provide an overview of tools and techniques for capturing malware, analyzing these statically or dynamically for extracting features and finally classifying them using machine learning algorithms. It also provides the examples from the literature that analyze executables for extracting useful features and apply machine learning for discriminating malicious software from benign ones. An overview of the tools and techniques described in the paper in context to malware detection is shown in figure 2.

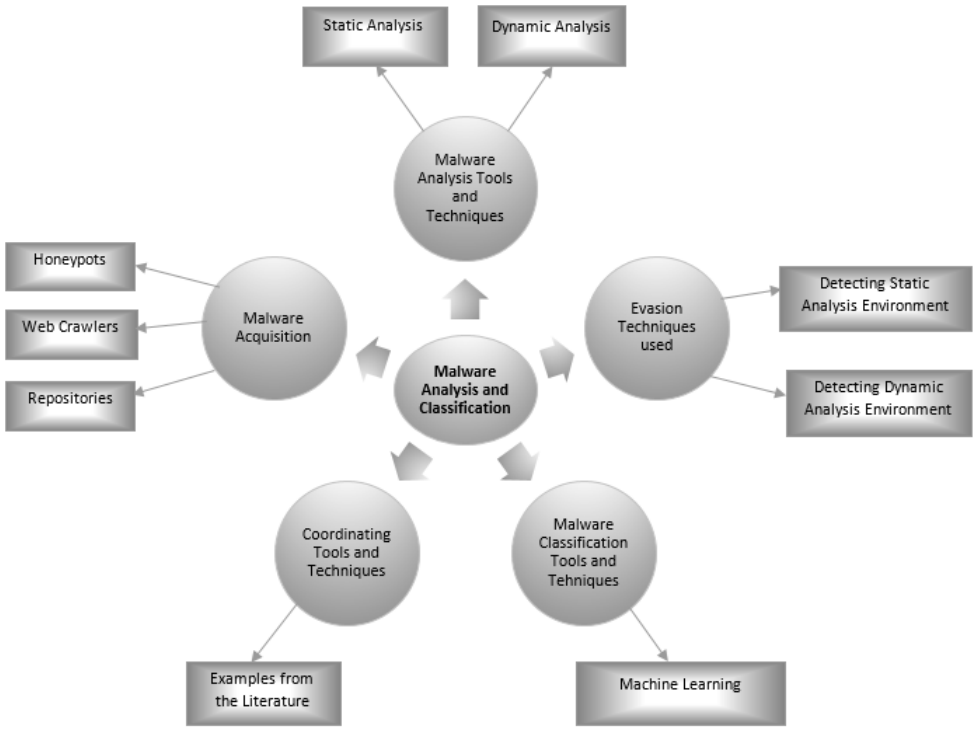

Figure 2. Outline of Tools and Techniques Discussed in the Paper

2. MALWARE ACQUISITION

The initial step for performing malware analysis is to collect specimens of malicious programs. There are various tools which can be used for collecting malware samples. Most of these rely on honeypots (such as Mwcollect[4], Nepenthes explained in Baecher et al. [2006], Honeytrap project[5], and Billy Goat discussed in Riordan et al. [2006] etc.) and spam traps described in Prince et al. [2005]. A discussion on honeypots based tools can be found in Zhuge et al. [2007]. Another method for collecting malware samples is to use web crawlers or by using an executable sniffer, deployed at the network edge for sniffing the PE files passing through the network. In addition to these methods, there are some sources[6] which provide free or after registered access to the malware samples contained on their websites.

Clean files can be collected from the system itself. For instance, the files from the directories of

[4]http://alliance.mwcollect.org

[5]https://sourceforge.net/projects/honeytrap

[6]http://zeltser.com/combating-malicious-software/malware-sample-sources.html

Windows Operating System (OS) can be used as clean files. These can also be collected from http://download.cnet.com/windows/ where various software programs are available for downloading.

3. MALWARE ANALYSIS

It is the ability to cut-apart the malevolent code for understanding its working and behaviour, so that, it can be identified and defeated. The malicious programs and their competence can be examined using two broad categories of malware analysis: Static Analysis (inspecting the code statically) and Dynamic Analysis (executing malware binaries in a virtual environment and monitoring their behaviour) as shown in figure 3. This section provides the description of Static and Dynamic malware analysis techniques and tools (shown in figure 4).

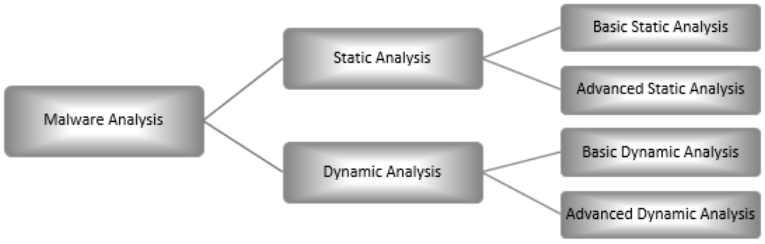

Figure 3. Malware Analysis Techniques

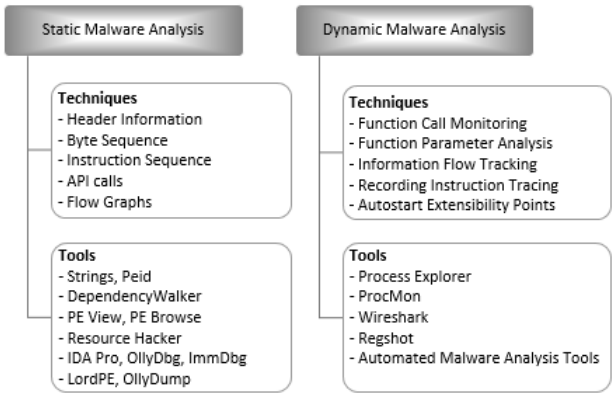

Figure 4. Tools and Techniques for Malware Analysis

3.1 Static Malware Analysis

Investigating malicious programs without executing them is known as static malware analysis. Basic static analysis inspects the malicious program without viewing the actual instructions. Technical gauges gathered with basic static analysis include file information (like name, size, and type) and the corresponding hash value which are recognized by AV detection tools. Advanced static analysis loads malware into a disassembler to reverse engineer and analyze its instructions and functionality. This type of analysis provides additional details about malware that are generally not revealed through basic static analysis.

3.1.1 *Existing Techniques.* Static malware analysis is usually conducted manually and can be applied on different representations. Various techniques can be used for performing static malware analysis. The most widely used are as follows.

PE Header Information: The PE header contains the information of great value to the malware analyst. It includes information about the import/export functions, required library functions (dynamic or static), type of application, time of its compilation etc. These are important components of a program which provides an insight into its actions. Many AV vendors use this information for detecting suspicious binaries.

Sequence: One of the features that can be extracted from an executable is the raw byte level content. It is most widely used by AV vendors for detecting malware. The simplest method is hashing the contents of the file and comparing it against a database of blacklisted hashes. Another method is to find static string signatures that represent the patterns in a malware's raw content to uniquely identify it. This method is efficient and more effective than matching hash values. Regular expressions are the extensions of string matching technique and are able to detect malware variants. Though it is an efficient and fast method, yet, is unable to detect polymorphic malware variants.

Instruction Sequence: The instruction contents of a program can be more resilient than the byte level content, if these are considered by their type or mnemonics. The malware needs to be disassembled before finding the instructions and opcodes.

API Calls: These are the application programming interface programs which are invoked to interact with the underlying OS & libraries and can be identified statically to have an insight into the programs intent.

Flow Graph: Static analysis can be done using various types of graphs representing the control flow, execution path flow and data flow of a program. A code sequence (at byte level or instruction level) without any control transfer is called a basic block. The basic blocks along with their dependencies can be grouped together to construct a directed control flow graph. In these graphs, the nodes act as the basic blocks and edges as the control flow of the concerned procedures. The control flow graph for different procedures can be combined together to represent the complete program. These can have an additional structure that models the possible execution paths called the call graphs. A program's data flow represents the set of possible data values that a program can assume. Data flow analysis methods look at the specific value of the data at various program points.

3.1.2 *Tools for Static Malware Analysis.* Most of the malicious programs are very large and complex. Generally, the key features are focused rather than understanding every detail of such programs. Different tools and approaches can be used for analyzing malware. These may have sometimes overlapping functionalities.

A program titled as Strings[7] can quickly examine ASCII and Unicode strings in binary files, but it can easily be thwarted by using packing and string obfuscation techniques. This is accomplished by using packer tools. After getting packed, the program binary looks very much different and its logic along with metadata is hard to recover. PEiD is used to detect the packed files. The executable needs to be unpacked before performing static analysis. The disassembler and debugger tools like IDA Pro[8] and OllyDbg[9] present a binary program as Intel X86 assembly instructions, which provide insight into the malicious activities of the program. Memory dumper tools like OllyDump[10] and LordPE[11] are used to investigate packed binaries which are tricky to disassemble. PEframe[12] is a tool developed using python that helps to perform static analysis

[7]http://bit.ly/ic4plL
[8]https://www.hexrays.com/products/ida/support/download_freeware.shtml
[9]http://www.ollydbg.de/
[10]http://www.woodmann.com/collaborative/tools/index.php/OllyDump
[11]http://www.woodmann.com/collaborative/tools/index.php/LordPE
[12]https://github.com/jt6211/peframe

on malware. In addition to performing some useful analysis on the PE file, it also helps in understanding if the malware is having any anti-debug function that blocks execution during dynamic analysis. There are many other static analysis tools like PE View, Dependency Walker, PE Browse etc. Their details can be found in Sikorski and Honig [2012]. The tools for performing static malware analysis are summarized in table I on the basis of their functionality.

Tool Name	**Functionality**
Strings	To search an executable for strings.
PEid	Detecting packed files. It can detect the type of packer/compiler used to build an application.
Dependency Walker	Lists dynamically linked libraries and functions imported by an executable.
PE View, PE Browse, PE Explorer	Give details of the PE files structure. PE header, individual sections and import/export tables can be viewed using these.
Resource Hacker	Display information about PE file sections: code, data and resource sections.
IDA Pro	Interactive X86 disassembler, providing functionality as function discovery, stack analysis and local variable identification.
LordPE, OllyDump	Memory Dumper tools. Very useful when analyzing packed executables, which encode or encrypt their instructions.
OllyDbg, ImmDbg	Debugger for malware analysis. ImmDbg is derived from OllyDbg

Table I: Static Analysis Tools with their basic Functionalities

An outstanding tool available online for performing static malware analysis is VirusTotal[13]. It was launched in the year 2004 and attained by Google in 2012. It provides the detailed static properties of malicious files within no time.

3.1.3 *Negative Aspects of Static Malware Analysis.* Most Malware authors are smart enough to thwart static analysis by making use of obfuscation techniques. These techniques transmute the malicious binaries into unique structured and compressed files which make the static analysis process expensive and untrustworthy. Usually, the source code of malicious programs is not available. When using binary executables, the information like variables or data structure gets lost, thus, complicating the static analysis of malware as specified by Egele et al. [2012].
Through their work, Moser et al. [2007b] reveal the fact that static analysis alone is not sufficient for classification of malware but the dynamic analysis is an indispensable complement to it because it is less susceptible to code obfuscation translation.

3.2 Dynamic Malware Analysis

Monitoring the actions performed by a program while it is being run in a safe environment is called dynamic analysis. It doesn't necessitate the malicious binary to get disassembled before performing the analysis. It actually monitors the behaviour of malware during its execution which is more flexible as compared to static analysis. The basic dynamic analysis reveals the information like domain names, IP address, registry keys, file path locations, additional files located on the system or network. In some cases, the basic dynamic analysis is not able to provide productive results as the malware may require some additional information to run. So, advanced dynamic analysis is performed which uses the debugger or some other special tools to extract detailed information from the malware file while it is being executed.

[13]https://www.virustotal.com//

3.2.1 *Existing Techniques.* There are several approaches which are used for dynamic malware analysis. The most common are introduced in this sub-section.

Function Call Monitoring: The semantic knowledge of a program can be gained by tracking the functions called by it. Tracking functions called by a program is done by using a process known as hooking. The executable being analyzed is maneuvered in such a way that a hook function is called along with the intended functions, which logs the invocation details. The OS provides many application programming interfaces that are used by applications for performing several tasks like file manipulation, network communication etc. It provides the system call interface API, which helps a user-mode application to carry out tasks on its behalf by switching to the kernel mode. On the invocation of a system call, the system first changes to the kernel-mode and performs the desired actions after verifying the access rights. Like non-malicious programs, the malicious programs (executing in the user mode) also need to invoke the system calls. Most of the malware are able to gain access to the kernel-mode and don't require a system call interface, thus, elude the malware analysis.

Function Parameter Analysis: It makes use of the actual values passed while invoking a function. Grouping of the actual values passed and returned by function into joint sets give an insight into the behaviour of the program.

Information Flow Tracking: The Information flow tracking highlights the dissemination of data right through the system while it is being manipulated. The data under observation is particularly marked with a label. The marked label is propagated while it is being processed by the application.

Recording Instruction Tracing: These are the machine instruction sequences that the program executed while it is being analyzed. These are very troublesome to read and interpret but contain significant information which is not revealed in a high-level abstraction like reports generated by function calls.

Autostart Extensibility Points (ASEPs): It is the mechanism that permit a program to get invoked automatically when an application is launched or the OS is booted. It is interesting to keep an eye on these ASEPs when a malicious program is analyzed as some of their components attempt to accumulate with one of the ASEPs.

3.2.2 *Tools for Dynamic Malware Analysis.* This sub-section provides an outline of the tools that are used to analyze the malicious programs. The outcome of these tools provides a valuable insight into the behaviour of malware. Different monitoring tools like Process Explorer[14], Process Hackerreplace[15], Process Monitor[16], Capture BAT[17], Wireshark[18] and Regshot[19] are installed and activated before executing the malicious programs. Different information such as file attributes, domains & protocols, registry settings, API calls, command & control and so on is gathered while analyzing malware using these tools. It is required to set up a laboratory system that can be used for malware analysis without affecting the production environment.

Process Explorer: It is a free tool by Microsoft, which gives information pertaining to the active processes and the various properties associated with those processes. It consists of two windows. One showing a list of active processes, and the other displaying the information based upon the mode of process explorer. If it is in the handle mode, the 2nd window displays the handles that the selected process has opened. If it is in the DLL mode it displays the DLL and memory-mapped files that the selected process has loaded. This tool is useful for tracking DLL-version problems due to its unique capabilities.

ProcMon: It is a sophisticated monitoring tool for Windows which brings together and en-

[14]https://technet.microsoft.com/en-us/sysinternals/bb896653.aspx
[15]http://processhacker.sourceforge.net/
[16]https://technet.microsoft.com/en-us/sysinternals/bb896645.aspx
[17]https://www.honeynet.org/node/315
[18]http://www.wireshark.org/
[19]http://sourceforge.net/projects/regshot/

hances the features of two utilities i.e Filemon and Regmon and is discussed in Sikorski and Honig [2012]. It provides the real-time information about the file system, registry and process activities. Though procmon is a useful tool, it is not able to capture the user-mode component activities of device driver and certain GUI calls like SetWindowsHookEx. This tool displays various columns which contain events information including the time stamp, sequence number, and the name of the process etc. This detailed information is too long to understand and analyze. It provides the filtering capability which can be used on individual system calls for the better analysis of malware behaviour.

WireShark: It is an open source packet-sniffing tool that provides packet flow analysis and detailed analysis of individual packets. It gives an insight into the network behaviour of malware by sniffing packets as malware communicates.

Regshot: It is a light weight open source tool that compares the key changes that a program makes to the file system and the registry.

3.2.3 *Negative Aspects of Dynamic Malware Analysis.* The major drawback of dynamic analysis is that it is time exhaustive as every sample needs to be executed for few minutes to observe its behaviour. Thus it leads to uplifting the scalability concerns. A controlled environment is required to be set up before performing dynamic analysis. This environment is completely different from the actual one. The malicious programs may be designed to behave in different ways in such environments resulting in synthetic behaviour rather than the real one. Moreover, some of the malicious programs are designed to execute under specific circumstances (like on execution of a specific command or on a specific date etc.) thus, thwarting the dynamic analysis.

3.2.4 *Automated Malware Analysis Tools.* Automated malware analysis is the simplest way to analyze malicious programs. Various commercialized and open source automated tools have been designed and developed with the intention of resolving the challenges raised by the large volume of malware samples. These systems seem to be similar at first glance, but use different technologies at the backend. The reports generated by automated systems provide information about the malware including both static and dynamic attributes like meta data of the file, mutex creation, file system activities, registry activities and network activities etc. Such automated analysis systems can help in providing the feature vectors or similarity measures for the process of discriminating malicious files from the clean ones using machine learning techniques. This subsection discusses some of the popular free and open source automated malware analysis tools.

Anubis: Anubis (Analysis of unknown binaries) project[20] has evolved from TTAnalyzer developed by Bayer et al. [2006]. It executes the malware binary (to be analyzed) in an emulated OS environment (Qemu Bellard [2005]) and generates a report containing enough information to have an idea about the purpose of the binary. It monitors the Window Native API calls, Window system API calls and system service calls (along with the parameters passed). It is also able to monitor all processes created by observing the API and system calls responsible for the same.

Multiple-Path Exploration: Dynamic analysis tools observe only a single program execution. However, some malware trigger their malicious behaviour only under certain circumstances. In order to address this problem, Moser et al. [2007a] presented a tool (by extending Anubis) possessing a capability to look at multiple execution paths and recognize the malicious behaviour that is revealed only under specific circumstances. Whenever, the tool detects the branch points for which the return value of a system call is responsible for control flow decision, a snapshot of the running process is taken. After the process terminates, the system is reset to the earlier stage snapshot and the value responsible for control flow decision is manipulated resulting in the execution of alternate path.

CWSandbox: It is developed by Willems et al. [2007] for executing the samples either natively or in a virtual windows environment. It captures the behaviour of malware samples pertaining to file system, network communication, registry changes and OS interaction. It uses rootkit

[20]http://anubis.iseclab.org/

techniques to hide system stuff that could reveal the presence of the analysis environment. It also uses hooking functions to keep a check on API level.

Norman Sandbox[21]: It runs the executables in a virtual environment simulating a Windows OS. It simulates a host computer and the attached local area network. This simulated system provides support for all OS mechanisms. Packed and obfuscated samples are not able to evade the analysis as these are executed in a simulated environment. Its main focus is to detect Worm (that spreads via email or P2P network) and Virus (that gets replicated using the network). It is able to log the information regarding function calls and their arguments. It also keeps a check on autostart extensibility points that some malware use to ensure automatic invocation after reboot.

Joebox: It is designed by Buehlmann and Liebchen [2010] to run on a real hardware instead of relying on emulation or virtualization. It uses a client-server model where client machines perform malware analysis and the server machine coordinates multiple clients and collect the analysis data. It provides various activities (like the file system, registry activities etc.) performed by the sample under analysis in the form of log files.

HookFinder: Some malicious programs make use of hooking techniques to get notified on the occurrence of some specific events. For instance, Keylogger may create a hook using API call SetWindowHookEx and get notified when a key is pressed. It is possible to detect these hooking techniques and generate the reports pertaining to where these hooks are and how these are entrenched into the system.

Ether: Dinaburg et al. [2008] pointed out that the analysis tools are usually detected by malicious code being monitored. In order to address this issue, they proposed a transparent malware analysis framework called "Ether". It is based on hardware virtualization extensions and is implemented in a hypervisor. It has the capability to monitor memory writes, executed instructions and system calls.

Table II shows an overview of the type of features that can be captured by using different automated tools for analyzing malicious programs.

Functionalities \ Sandbox	Anubis	Multipath	CWSandbox	Norman Sandbox	JoeBox	Hookfinder	Ether
API Calls	✓	✓	✓	✓	✓	✓	✗
System Calls	✓	✓	✓	✓	✓	✓	✓
Function Parameters	✓	✓	✓	✓	✓	✓	✗
File System Operation	✓	✓	✓	✓	✓	✓	✗
Process Creation	✓	✓	✓	✓	✓	✓	✗
Instruction Trace	✗	✓	✗	✗	✗	✓	✓
Info. Flow Tracking	✗	✓	✗	✗	✗	✗	✗
Multipath Exploration	✗	✓	✗	✗	✗	✗	✗
ASEP	✗	✗	✗	✓	✗	✓	✗

Table II: Comparison of various sandboxes

3.2.5 *Tools to deal with Packed Binaries.* Malware authors make use of packer tools which automatically transforms malicious executable into a representation which is equivalent semantically but differ in syntax. It is done by using obfuscation techniques. Using this method, the malware instances obfuscate the malicious part of their code at compile time and unpack them back into executable code at runtime. Analyzing these binaries dynamically is not different from those of non-packed binaries.

Royal et al. [2006] proposed a technique named as "PolyUnpack" which has the capability to

[21] http://download01.norman.no/product_sheets/eng/SandBox_analyzer.pdf

automatically extract the hidden code bodies of such malware. It combines the static-dynamic analysis method to recognize the execution of code generated at runtime. It is based on the two step algorithm. In the first step, binary is disassembled. In the second step, binary is executed in a virtual environment for monitoring instruction traces by performing in-memory disassembly for every executed instruction. If the instruction sequence obtained from the second step is not present in the disassembled binary obtained from the first step, it means that the code is dynamically generated and is about to execute. Another similar technique named as "OmniUnpack" was developed by Martignoni et al. [2007], which monitors the program in real time while it is being executed and detects when the program has detached from the several layers of packing. This tool reverses the packed binaries and scans the recovered code using AV software. Renovo, proposed by Kang et al. [2007] is another tool which is used to deal with packed binaries.

4. MALWARE ANALYSIS EVASION TECHNIQUES

Malware writers tend to design the programs which use stealth techniques and multiple defense mechanisms to evade detection. These include self-modification of malware binaries while propagating, dynamically generated code and the approaches that assist them to obscure their malicious behaviour while executing in instrumented environment. This section discusses the techniques that malware authors use to evade malware analysis. These may also be used by authors of legitimate programs to protect their programs from being analyzed.

4.1 Detecting Static Analysis Environment

In order to thwart static analysis, malware authors use self-modifying parts in the code. Recently developed packer tools obfuscate/encrypt the executable to a new executable, which is semantically equivalent to the original one. An unpacker routine deobfuscates/decrypts the binary to the original representation, which then performs the intentional malicious tasks. For example, Polymorphic and Metamorphic are two classes of malware which have the capability to change their code while propagating.

Polymorphic malware has two parts: one is encrypted malware body and second is decryption routine. Their variants can easily be created by using random keys for encryption. On launching an infected application the decryption routine decrypts the malware body back to its original form. The decryption routine remains the same which makes it a little easy for AV program to identify the malware. Metamorphic variants, on the other hand, also mutate the unpacking/decryption routine and every succeeding version of the malcode is different from the preceding one which in turn impedes its detection. You and Yim [2010] describes obfuscation techniques (like subroutine reordering, insertion of dead code, instruction substitution etc.) being used by malware authors for creating polymorphic and metamorphic malware.

4.2 Detecting Dynamic Analysis Environment

In dynamic analysis, the information features are collected when the program is executed in the virtual environment. It may miss some of the important information due to the design being used by malware authors for evading its detection. In this context, the main task of malware authors is to determine if the code is running in a real target machine or in a virtual environment for monitoring purpose. To this end, many techniques have been developed and being used by malware writers. Some of these are discussed as follows:

4.2.1 *Human Interaction .* Malware authors make use of the fact that sandboxes emulate physical systems without having any interaction with the user.

Mouse Click: Malware may start to perform malicious activities only after detecting a left mouse button click. For instance, Upclicker[22] from a Trojan family, analyzed in December, 2012

[22]https://www.fireeye.com/blog/threat-research/2012/12/dont-click-the-left-mouse-button-trojan-upclicker.html

used mouse click for detecting human activity. After six months of discovery of Upclicker, another malware Banechant[23] was analyzed which activates only after three mouse clicks.

Dialogue Boxes: Malware can identify a live target by presenting a dialog box which necessitates a respond from the user side. In order to create dialog boxes in EXE and DLL files, different APIs of windows like MessageBox and MessageBoxEx are used. In Singh and Bu [2013], the authors describe how a malware gets activated and start to show their actual behaviour only after the user click a button on the dialog box.

4.2.2 *Configuration Specific.* The virtual environments are configured so as to mimic the physical computers. Malware authors can make use of these configurations to evade malware automated analysis.

Sleep Calls: An extended sleep call is added to the code, so that, the malware refrains from any distrustful behaviour during analysis. For instance, a malware named as Trojan Nap[24], discovered in February, 2013, makes use of this approach. This malware used a sleepEx() method with a timeout of 10 minutes or an API method NtDelayExecution() to delay any suspicious action. Another API call Sleep() can also be used with time triggers to execute malware. By using the API method, it can find out the current date and time of local machine. If the file is designed to be executed on a specific date or time, it calls a sleep() method with parameter value equal to the time for which it wants to remain inactive. After that, it again checks the current date and time. If it still has not reached the explosion trigger, it again sleeps and repeats the loop, until detonation time has reached. For example, a malware called Hastati[25] (from Trojan family) attacked in South Korea in March, 2013, which shows the same behaviour to evade its analysis.

4.2.3 *Environment Specific .* Malware authors insert features into the malicious code that can check whether they are being executed in the virtual environment or not.

Version Check: Some files are designed to execute only on specific OS versions. If the required version is not installed on sandbox, the malicious file doesn't show its actual behaviour. For example, a PDF file can make use of JavaScript code which uses viewerVersion() API method to determine the version of Acrobat Reader installed.

DLL Loader Checks: In order to run a DLL file, normally a run32dll.exe is used or a DLL is loaded in a process that executes it. Malware authors may design the malicious code, which requires a particular loader to run the DLL. If the requisite loader is not present, the DLL doesn't get executed and hence is not detected by the sandbox.

Embedded iframes: Malware authors sometimes make use of legitimate files to evade detection by defenses. One of the approaches is embedding iframe in a non-executable file, like an Acrobat Flash. These files do not execute by themselves thus do not reveal any suspicious behaviour in the virtual environment. In fact, they conceal data, which is executed after getting unlocked by another file waiting for it on a compromised machine. Malware authors make use of distinctive configuration or environment of a virtual machine to evade malware detection. They use processes, services and communication port specific to the environment (VMware, VPC or VirtualBox) to evade malware analysis. For instance[26], a VMware makes use of VMX port to communicate with the virtual machine. The port is checked by malware code. If it is present, the malware plays dead to avoid detection . A commonly used method to check the presence of automated malware analysis systems such as CWSandbox and Anubis is to check for a unique product key.

[23]https://www.fireeye.com/blog/threat-research/2013/04/trojan-apt-banechant-in-memory-trojan-that-observes-for-multiple-mouse-clicks.html
[24]https://www.fireeye.com/blog/threat-research/2013/02/an-encounter-with-trojan-nap.html
[25]https://www.fireeye.com/blog/threat-research/2013/03/more-insights-on-the-recent-korean-cyber-attacks-trojan-hastati.html
[26]https://www.virusbulletin.com/virusbulletin/2013/02/techniques-evading-automated-analysis

5. MALWARE CLASSIFICATION

This section presents an overview of the machine learning techniques and tools which have been used in the literature to classify malware samples into their families or underline those that need a closer human analysis.

5.1 Machine Learning Techniques

Machine learning is a class of Artificial Intelligence that offers computers the ability to learn without being explicitly programmed. It is similar to the process of data mining which searches through the data to look for patterns. However, machine learning uses the detected patterns for adjusting the program actions accordingly.
In machine learning, classification is the process of identifying, to which category a new observation belongs on the basis of training data set containing examples whose category is known. The classification process consists of two-steps: Training and Classification. Training step involves the construction of classification model and classification step is used to foretell the class labels for the given data. If the training data is labelled, it is called Supervised Learning which contrasts with the Unsupervised Learning where it is not labelled. There is another class of machine learning techniques knows as Semi-Supervised Learning, which makes use of both labelled and unlabelled data. Labelled instances are used to learn the class models, whereas, unlabelled instances are used to refine the boundaries between classes. It is believed that the unlabelled data, when used in conjunction with a small amount of labelled data can produce substantial improvement in learning accuracy.

5.1.1 *Naïve Bayes (NB).* It is a simple Bayesian classifier which is based on Bayes theorem (from statistics) with strong (naïve) independent assumptions. It has the capability to predict the probabilities, with which a given tuple of a data set belongs to a specific class. Han et al. [2011] explains that a feature-values effect on a given class is independent of the values of the other features suggesting thereby that all present features contribute independently to calculate the probability for data classification. This model is useful for very large datasets and is easy to build. Let the dataset D has a feature vector $X = (x_1, x_2....x_n)$, then according to Bayes theorem, the posterior probability is computed by using equation (1).

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} \ldots\ldots (1)$$

where, $P(c|x)$ is the posterior probability of class c, given predictor attribute x . $P(x|c)$ is the probability of predictor attribute given class c. $P(c)$ is the prior probability of class and $P(x)$ is the prior probability of predictor attribute.

5.1.2 *Support Vector Machine (SVM).* This algorithm makes use of a nonlinear mapping to convert the training data into higher dimensions, where a linear optimal hyper-plane separating the data of one class from another is searched. The hyper-plane is established by using training tuples (support vectors) and margins defined by them. This algorithm looks for the hyper-plane with the largest margin. Sequential Minimal Optimization (SMO) algorithms are the fast implementation of SVM. They solve the problem pertaining to optimization which arises during the training phase.

5.1.3 *Artificial Neural Networks (ANN) .* It is a standard for information processing and is motivated by the system a human brain works. It consists of a number of processing elements (called neurons) which are organized in layers. Through input layer, the patterns are offered to the network and thus communicated to the hidden layers where actual processing is done using weighted connections. The disadvantage of ANN is that their operations/results can be unpredictable, unlike other conventional computers which use the cognitive approach to solve a problem. According to Han et al. [2011], a Multilayer Perceptron (MLP) is a feed forward ANN

model that maps sets of input data onto a set of appropriate outputs. It consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Except for the input nodes, each node is a neuron with a nonlinear activation function. MLP utilizes a supervised learning technique called back propagation for training the network.

5.1.4 *Instance Based Learners .* These are the learning algorithms that match new problem instances with those in training data. It is a kind of lazy learning that waits until the last minute before constructing classification model i.e. it simply stores the training data and waits until it is given a test data. On getting the test data, it constructs the classification model based on the matching to the stored training data instances. These learners, however, support incremental learning. These learners are computationally expensive. These necessitate proficient storage techniques and are appropriate for implementation on parallel hardware. Examples of this type of learning are k-Nearest Neighbor (kNN) and IB1 algorithm and are explained in Han et al. [2011].

5.1.5 *Boosted Classifiers .* It is an ensemble method which combines the different classifiers into a composite model and is more accurate than their component classifiers. This method consists of producing a set of weighted models by learning it iteratively with the help of a weighted data set. Then it is evaluated and new weights are assigned to data set tuples based on its performance. These weights and models are then used for predicting the class with the highest weight as expained in Maloof [2006]. For example, Bagging (Bootstrap Aggregation) is a meta-algorithm which combines the output of different models to make the final decision more reliable by reducing the variance error.

5.1.6 *Decision Tree.* It is a rooted tree with each internal node considered as the test on an attribute, each branch, as a result of the test and each leaf node holding a class label. Attributes and corresponding values are used to traverse the tree from root to leaf node and predict its class label. The tree is built by selecting the attributes and their values for best splitting the training instances into their appropriate classes. In this way, the nodes, branches and leaves are created for various attributes and their values. The process repeats itself until a node contains instances of the same class as discussed in Maloof [2006]. An example of Decision Tree is J48 (a version of C4.5 Decision Tree), which builds the decision tree based on the labelled training data using the concept of entropy.

5.1.7 *Clustering .* It is an unsupervised machine learning technique which intends to divide a given data set into significant groups called clusters. The objects within one cluster are similar to each other and dissimilar from objects contained in other clusters. Clustering helps to discover the structure in the unknown data and can be employed using different methods. The major fundamental clustering methods can be categorized into three classes: Partitioning, Hierarchical and Density based methods as explained in Han et al. [2011]. Partitioning Method is a distance-based technique which finds mutually exclusive clusters of spherical shape. These are effective for small to medium size datasets. Hierarchical method generates a hierarchical decomposition of the given data set objects and cant undo the erroneous merges. Density-based methods create the arbitrary shaped clusters of dense regions. It has the capability to filter out the outliers.

5.2 Data Mining and Machine Learning Tools

Various data mining and machine learning tools including WEKA by Hall et al. [2009], Orange[27], RapidMiner[28], and KNIME[29] etc. can be used for data analysis along with visualization. (Comparative analysis of these tools can be found in Christa et al. [2012]). Among these tools, WEKA (Waikato Environment for Knowledge Analysis) is the most widely used in literature for classifica-

[27]http://orange.biolab.si/
[28]http://rapidminer.com/
[29]http://www.knime.org/

tion and clustering in different areas including malware classification. It is a java implementation of various data mining and machine learning programs. It is freely available and developed at the University of Waikato, New Zealand. It has visualization capabilities and contains various algorithms for data analysis along with GUIs for providing easy access to different functionalities.

6. COORDINATING TOOLS AND TECHNIQUES FOR MALWARE ANALYSIS AND CLASSIFICATION

This section offers the instances from the literature which analyze binary files statically or dynamically or both and applies machine learning for discriminating malicious software from benign ones.

Schultz et al. [2001] and Kolter and Maloof [2004] were among the first few researchers who used the concept of data mining and machine learning for detecting and classifying malware. Schultz et al. [2001] used three static features (PE features, strings and byte sequence) as a basis for malware classification using Naïve Bayes and Multinomial Naïve Bayes. In Kolter and Maloof [2004], authors extracted byte sequences from executable's hexadecimal code and converted them into n-grams (continuous sequence of n items), which is then analyzed for making a base as a feature for malware detection and classification using classifiers.

In Saini et al. [2014], the authors presented a scalable method of discriminating malicious files from the clean ones. They used suspicious section count and function call frequency as the static features of malware and used machine learning algorithms of WEKA for classification purpose. The experimental results conducted provide an accuracy of 98.35%.

Kong and Yan [2013] used structural information (function call graph) as a basis for malware classification. They apply discriminate distance metric learning for finding similarity between programs.

Nataraj et al. [2011] used image processing technique for malware visualization and K-nearest neighbor model with Euclidean distance method for malware classification.

Hu et al. [2016] proposed a design for a scalable malware classification system which uses multiple content features (like strings, section information etc.) and threat intelligence information from various sources. The evaluation is done on a dataset consisting of 21,741 malicious files to prove the efficiency of proposed algorithm.

The authors used function length frequency in Tian et al. [2008] and printable string information in Tian et al. [2009] to classify the malicious data using machine learning algorithms available in WEKA. For detecting worms in the wild, Siddiqui et al. [2009] used instruction sequences of variable length along with machine learning. In Bilar [2007], Santos et al. [2010] and Siddiqui et al. [2008], the authors used opcode distribution for detecting malware.

After collecting malware using HoneyClients and Amun, Zolkipli and Jantan [2011] identified their behaviour by executing them on both CWSandbox and Anubis. Their results were customized using human behaviour analysis for grouping them into their families. Rieck et al. [2011] also used dynamic analysis along with clustering for malware classification. Anderson et al. [2011] used the graphs constructed from dynamically collected instruction traces using Ether and used SVM for malware classification.

Bayer et al. [2009] proposed a system relying on Anubis with tainted propagation capabilities for generating execution traces of all the samples. They used clustering for malware grouping. Tian et al. [2010] extracted API call sequences from executables and made use of WEKA tool for discriminating malware from clean files and for classifying malicious programs into their families.

In Cho et al. [2016], the authors presented a framework for malware classification which makes use of the sequence alignment method. API call sequences are logged while executing the malicious files in the virtual environment set up using Cuckoo[30] Sandbox. The common parts are identified from the API sequences and are used to find the similar behaviour of malware variants.

[30]https://cuckoosandbox.org/

The experiment is conducted on a dataset of 150 malware samples from 10 families and average accuracy is found to be 87%.

Imran et al. [2015] proposed a system for malware classification which makes use of a sequence of system calls obtained after executing the malicious files in CWSandbox along with Hidden Markov model. The proposed approach is validated on 964 malicious and 50 benign files which gives an average accuracy of 97.34%.

Bailey et al. [2007] expressed malware behaviour using system state changes. After running the binaries in sandboxed environment, a behavioural fingerprint related to their activities like processes created, files written and network connection etc. is created and then a pairwise single linkage hierarchical clustering of these fingerprints is performed to cluster the malware.

Park et al. [2010] extracted system calls along with their parameters after running the binaries in a sandboxed environment and generated a directed graph from these. The maximal common subgraph is used to compute the similarity between the graphs of these programs. Firdausi et al. [2010] used Anubis for analyzing malware behaviour. The behavioural results thus gathered are preprocessed into sparse vector models for classification using machine learning algorithms available on WEKA.

Nari and Ghorbani [2013] also used WEKA for malware classification after extracting network behaviour from pcap files. Lee and Mody [2006] created a behavioural profile using the information recorded concerning sample's interaction with system environment like registry changes, file activities and network activities. Then they use K-nearest neighbor method for malware classification.

Authors of Santos et al. [2013], Islam et al. [2013], Anderson et al. [2012] and Gandotra et al. [2014a] pointed out that using only static or dynamic analysis features is not adequate for detecting malware accurately. Thus, they proposed hybrid techniques which are capable of including both static and dynamic features simultaneously. They extracted features of both static and dynamic analysis simultaneously and used machine learning algorithms for their classification. A detailed review of malware analysis and classification models that are used in the literature can be found in Gandotra et al. [2014b]. Table III summarizes the techniques used in literature to classify malware using various tools along with the dataset used and their Accuracy (Acc) or Area under the Curve (AUC).

Malware Analysis Technique	Features used for classification	Machine learning approach	Dataset			Acc/AUC	Reference
			Malicious	Benign	Total		
Static	program headers, strings and byte sequences	NB, Multi NB	3,265	1,001	4,266	Acc: 97.11%	Schultz et al. [2001]
	n-grams	NB, DT, SVM and boosting	1,651	1,971	3,622	AUC with a 95% confidence interval 0.9836±0.0095	Kolter and Maloof [2004]
	Suspicious Section Count and Function Call Frequency	NB, MLP and J48 tree of WEKA	2,460	627	3,087	Acc: 98.35%	Saini et al. [2014]
	Binary texture based on image processing	kNN	9,458	-	9,458	Acc: 98%	Nataraj et al. [2011]
	Function length frequency	Centroid Vector	721	-	721	Acc.: 87.7%	Tian et al. [2008]
	Printable string information	NB, SVM, RF, IB1, DT of WEKA library	1,206	161	1,367	Acc: 97%	Tian et al. [2009]
Dynamic	API call sequence	RF, DT, SMO, IB1 on WEKA library	1,368	456	1,824	Acc: 97%	Tian et al. [2010]
	API Sequence (Cuckoo Sandbox)	Sequence Alignment Method	150	-	150	Acc: 87%	Cho et al. [2016]
	Sequence of System Calls (CWSandbox)	Hidden Markov Model	964	50	1,014	Acc: 97.34%	Imran et al. [2015]
	Reports generated using Anubis are preprocessed into sparse vector models	kNN, NB, J48, DT, SVM and MLP on WEKA library	220	250	470	Acc: 96.8%	Firdausi et al. [2010]
	Flow Extraction and Behavioural Graph features (Automatic Experimentation System)	J48 and other classifiers (names not specified) of WEKA library	9,610	-	9,610	Acc: 94.57%	Nari and Ghorbani [2013]
Hybrid	Sequence of operational codes (Static) and system calls, operations and raised exceptions (Dynamic)	DT, kNN, BN and SVM on WEKA library	13,189	13,000	26,189	AUC: 0.98	Santos et al. [2013]
	Function length frequency and printable sting information (Static) and API function calls (dynamic)	SVM, IB1 and RF on WEKA library	2,398	541	2,939	Acc: 97.05%	Islam et al. [2013]
	Suspicious section count, Function call frequency (Static) and network activities & file activities (Dynamic)/Integrated	MLP, IB1, DT and RF on WEKA library	998	428	1,426	Acc: 99.58%	Gandotra et al. [2014a]

Table III: Examples from the Literature

7. INSIGHTS DRAWN AND FUTURE DIRECTIONS

While conducting the study on tools and techniques being used for malware analysis and classification process, a few of the lessons learned and the insights drawn out of these are discussed in this section.

—The first step involved in the process of malware analysis and classification is to acquire malicious specimens. Researchers working in this area, acquire these samples from different sources for conducting experiments to validate the proposed techniques. Typically, the accuracy of classification methods depends on the dataset selected. It may provide good results to a particular set of samples and may not give good results on some other dataset. Unfortunately, there doesn't exist any standard dataset of malware samples using which the researchers can compare their techniques. So, a dataset of malware samples (consisting of various types) should be created, so that, researchers can use that as a benchmark for conducting experiments to validate their techniques and to compare the results with existing techniques.

—The malware analysis techniques discussed in section 3 can be grouped into stages creating a pyramid that grows in the upward direction in terms of complexity and effort required to perform analysis (figure 5).

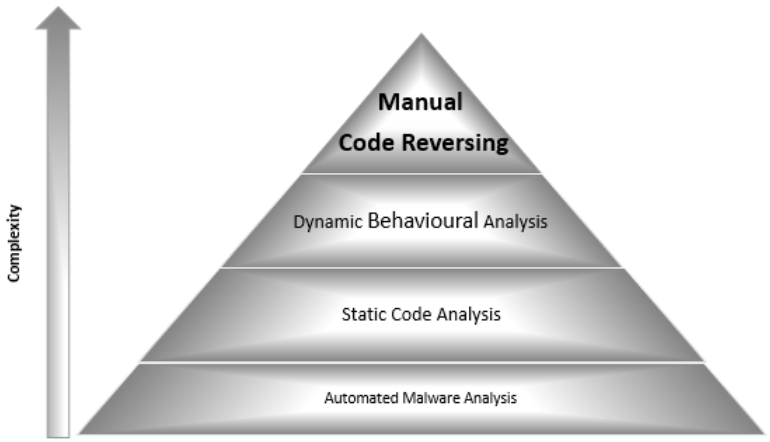

Figure 5. Phases of Malware Analysis Techniques with Complexity

The automated tools don't provide as much insight as an analyst would obtain while investigating the sample manually. However, the reports generated help the security analysts in quick incident response. It also helps in identifying the specimens which really need closer human attention and requires to move close to the top of the pyramid. Moreover, the automated malware analysis tools are being evaded by malware writers thus exposing the organizations and businesses to a series of new threats.

Basic static malware analysis can be performed quickly as compare to the dynamic behavioural analysis because these do not require executing the malicious files. Moreover, it provides the basic pointers of compromise which help in determining if the specimen is a potential candidate for closer analysis or not.

The dynamic behavioural analysis involves understanding malware's interaction with the system and its resources. It involves not only to observe the malware behaviour but also to interact with it for getting better insights. Thus the process is more complex as compared to the earlier techniques specified in the pyramid.

Manually reverse-engineering the malicious code adds valuable insights. It is a rigorous task as it involves the usage of the debugger, disassembler and other specialized tools. It takes a lot of time and requires the skilled analysts. All the stages shown in the pyramid, if considered as sequential steps, simplifies the process of malware analysis as the insights gain at the lower stage (less complex) can help to put efforts at the upper one (more complex).

—The evasion techniques being used by malware (as discussed in section 4), clears that an arms race has developed between malware authors and security researchers. As innovative malware analysis technique is proposed, malware authors react with newer techniques to frustrate malware analysis. So, there is a need to keep update regarding the new techniques used by malicious programs and get prepared to take preventive measures. This can be done through sharing malware intelligence information.

—From the open literature related to malware classification, it is evident that a single vision (either static or dynamic) is not appropriate for accurately classifying malicious programs because of the evasion techniques being used by malware authors. Performing malware classification using integrated attributes aids better recognizing the malware's intent that may be missed by static or dynamic analysis alone. So, the need is to adapt a hybrid approach which integrates both static and dynamic attributes simultaneously for better malware detection and classification.

—Today's threat landscape is full of volume, variety and velocity. It calls for the implementation of big data analytics to enhance malware detection and prediction accuracy. The continuously streaming network data necessitates new machine learning and data mining algorithms with novel methodologies and principles intended for renovating raw data into the constructive information.

In addition to the above insights drawn while conducting the study on malware analysis and classification tools and techniques, it is comprehended that there is a need to develop a model that facilitates insights on malware and is able to go beyond the accustomed gaps which remains owing to traditional malware analysis. This type of model should be able to provide information of value and interest with respect to the malware along with the intrinsic technical aspects. The insight so generated can be used for generating early warnings to defend against malware. The intelligent information obtained can be shared with CERTs (Computer Emergency Response Teams) and other stakeholders, who can take preventive measures to stop these threats before they actually cause damage or to mitigate the risk of their impacts on critical infrastructure.

8. CONCLUSION

Malware has consistently been marked as one of the key cyber threats to business, governments and individuals. For developing the countermeasures against malware, it is imperative to examine and understand their behaviour and the evasion techniques they might use to evade malware analysis. This paper presents a high-level description of various tools and techniques that assist an analyst in getting the quick and detailed knowledge on malware attributes and behaviour. Automated malware analysis, for instance, produces the reports containing such type of information which can further be used to cluster the malware having same behavioural profiles and to identify those which need closer manual inspection. The paper also highlights the lessons learned and insights drawn from this study which can help the researchers to innovate new techniques and design/develop better tools for malware detection and classification.

References

Anderson, B., Quist, D., Neil, J., Storlie, C., and Lane, T. 2011. Graph-based malware detection using dynamic analysis. *Journal in Computer Virology 7,* 4, 247–258.

Anderson, B., Storlie, C., and Lane, T. 2012. Improving malware classification: bridging the static/dynamic gap. In *Proceedings of the 5th ACM workshop on Security and artificial intelligence*. ACM, 3–14.

Baecher, P., Koetter, M., Holz, T., Dornseif, M., and Freiling, F. 2006. The nepenthes platform: An efficient approach to collect malware. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 165–184.

Bailey, M., Oberheide, J., Andersen, J., Mao, Z. M., Jahanian, F., and Nazario, J. 2007. Automated classification and analysis of internet malware. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 178–197.

Bayer, U., Comparetti, P. M., Hlauschek, C., Kruegel, C., and Kirda, E. 2009. Scalable, behavior-based malware clustering. In *NDSS*. Vol. 9. Citeseer, 8–11.

Bayer, U., Kruegel, C., and Kirda, E. 2006. *TTAnalyze: A tool for analyzing malware*. na.

Bayer, U., Moser, A., Kruegel, C., and Kirda, E. 2006. Dynamic analysis of malicious code. *Journal in Computer Virology 2,* 1, 67–77.

Bellard, F. 2005. Qemu, a fast and portable dynamic translator. In *USENIX Annual Technical Conference, FREENIX Track*. 41–46.

Bilar, D. 2007. Opcodes as predictor for malware. *International Journal of Electronic Security and Digital Forensics 1,* 2, 156–168.

Buehlmann, S. and Liebchen, C. 2010. Joebox: a secure sandbox application for windows to analyse the behaviour of malware.

Cho, I. K., Kim, T. G., Shim, Y. J., Ryu, M., and Im, E. G. 2016. Malware analysis and classification using sequence alignments. *Intelligent Automation & Soft Computing 22,* 3, 371–377.

Christa, S., Madhuri, K. L., and Suma, V. 2012. A comparative analysis of data mining tools in agent based systems. *arXiv preprint arXiv:1210.1040*.

Cloppert, M. 2009. Security intelligence: Attacking the kill chain. *Retrieved on June 1*, 2012.

Dinaburg, A., Royal, P., Sharif, M., and Lee, W. 2008. Ether: malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 51–62.

Egele, M., Scholte, T., Kirda, E., and Kruegel, C. 2012. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys (CSUR) 44,* 2, 6.

Firdausi, I., Erwin, A., Nugroho, A. S., et al. 2010. Analysis of machine learning techniques used in behavior-based malware detection. In *Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on*. IEEE, 201–203.

Gandotra, E., Bansal, D., and Sofat, S. 2014a. Integrated framework for classification of malwares. In *Proceedings of the 7th International Conference on Security of Information and Networks*. ACM, 417.

Gandotra, E., Bansal, D., and Sofat, S. 2014b. Malware analysis and classification: A survey. *Journal of Information Security 2014*.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. 2009. The weka data mining software: an update. *ACM SIGKDD explorations newsletter 11,* 1, 10–18.

Han, J., Pei, J., and Kamber, M. 2011. *Data mining: concepts and techniques*. Elsevier.

Hu, X., Jang, J., Wang, T., Ashraf, Z., Stoecklin, M. P., and Kirat, D. 2016. Scalable malware classification with multifaceted content features and threat intelligence. *IBM Journal of Research and Development 60,* 4, 6–1.

Imran, M., Afzal, M. T., and Qadir, M. A. 2015. Using hidden markov model for dynamic malware analysis: First impressions. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2015 12th International Conference on*. IEEE, 816–821.

Islam, R., Tian, R., Batten, L. M., and Versteeg, S. 2013. Classification of malware

based on integrated static and dynamic features. *Journal of network and computer applications 36,* 2, 646–656.

Kang, M. G., Poosankam, P., and Yin, H. 2007. Renovo: A hidden code extractor for packed executables. In *Proceedings of the 2007 ACM workshop on Recurring malcode*. ACM, 46–53.

Kolter, J. Z. and Maloof, M. A. 2004. Learning to detect malicious executables in the wild. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 470–478.

Kong, D. and Yan, G. 2013. Discriminant malware distance learning on structural information for automated malware classification. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1357–1365.

Lee, T. and Mody, J. J. 2006. Behavioral classification. In *EICAR Conference*. 1–17.

Maloof, M. 2006. *Machine Learning and Data Mining for Computer Security: Methods and Applications*. Advanced Information and Knowledge Processing. Springer.

Martignoni, L., Christodorescu, M., and Jha, S. 2007. Omniunpack: Fast, generic, and safe unpacking of malware. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*. IEEE, 431–441.

Moser, A., Kruegel, C., and Kirda, E. 2007a. Exploring multiple execution paths for malware analysis. In *2007 IEEE Symposium on Security and Privacy (SP'07)*. IEEE, 231–245.

Moser, A., Kruegel, C., and Kirda, E. 2007b. Limits of static analysis for malware detection. In *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*. IEEE, 421–430.

Nari, S. and Ghorbani, A. A. 2013. Automated malware classification based on network behavior. In *Computing, Networking and Communications (ICNC), 2013 International Conference on*. IEEE, 642–647.

Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B. 2011. Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security*. ACM, 4.

Park, Y., Reeves, D., Mulukutla, V., and Sundaravel, B. 2010. Fast malware classification by automated behavioral graph matching. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*. ACM, 45.

Prince, M. B., Dahl, B. M., Holloway, L., Keller, A. M., and Langheinrich, E. 2005. Understanding how spammers steal your e-mail address: An analysis of the first six months of data from project honey pot. In *CEAS*.

Rieck, K., Trinius, P., Willems, C., and Holz, T. 2011. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security 19,* 4, 639–668.

Riordan, J., Zamboni, D., and Duponchel, Y. 2006. Building and deploying billy goat, a worm detection system. In *Proceedings of the 18th annual FIRST conference*. Vol. 2006. 174.

Royal, P., Halpin, M., Dagon, D., Edmonds, R., and Lee, W. 2006. Mirrored by: www. siliconinvestigations. com for more information, call us-920-955-3693.

Saini, A., Gandotra, E., Bansal, D., and Sofat, S. 2014. Classification of pe files using static analysis. In *Proceedings of the 7th International Conference on Security of Information and Networks*. ACM, 429.

Santos, I., Brezo, F., Nieves, J., Penya, Y. K., Sanz, B., Laorden, C., and Bringas, P. G. 2010. Idea: Opcode-sequence-based malware detection. In *International Symposium on Engineering Secure Software and Systems*. Springer, 35–43.

Santos, I., Devesa, J., Brezo, F., Nieves, J., and Bringas, P. G. 2013. Opem: A static-dynamic approach for machine-learning-based malware detection. In *International Joint Conference CISIS12-ICEUTE´ 12-SOCO´ 12 Special Sessions*. Springer, 271–280.

Schultz, M. G., Eskin, E., Zadok, F., and Stolfo, S. J. 2001. Data mining methods

for detection of new malicious executables. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*. IEEE, 38–49.

Siddiqui, M., Wang, M. C., and Lee, J. 2008. Data mining methods for malware detection using instruction sequences. In *Artificial Intelligence and Applications*. 358–363.

Siddiqui, M., Wang, M. C., and Lee, J. 2009. Detecting internet worms using data mining techniques. *Journal of Systemics, Cybernetics and Informatics 6,* 6, 48–53.

Sikorski, M. and Honig, A. 2012. *Practical malware analysis: the hands-on guide to dissecting malicious software*. no starch press.

Singh, A. and Bu, Z. 2013. Hot knives through butter: Evading file-based sandboxes. *Threat Research Blog*.

Tian, R., Batten, L., Islam, R., and Versteeg, S. 2009. An automated classification system based on the strings of trojan and virus families. In *Malicious and Unwanted Software (MALWARE), 2009 4th International Conference on*. IEEE, 23–30.

Tian, R., Batten, L. M., and Versteeg, S. 2008. Function length as a tool for malware classification. In *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*. IEEE, 69–76.

Tian, R., Islam, R., Batten, L., and Versteeg, S. 2010. Differentiating malware from cleanware using behavioural analysis. In *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*. IEEE, 23–30.

Willems, C., Holz, T., and Freiling, F. 2007. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security and Privacy 5,* 2, 32–39.

You, I. and Yim, K. 2010. Malware obfuscation techniques: A brief survey. In *BWCCA*. Citeseer, 297–300.

Zhuge, J., Holz, T., Han, X., Song, C., and Zou, W. 2007. Collecting autonomous spreading malware using high-interaction honeypots. In *International Conference on Information and Communications Security*. Springer, 438–451.

Zolkipli, M. F. and Jantan, A. 2011. An approach for malware behavior identification and classification. In *Computer Research and Development (ICCRD), 2011 3rd International Conference on*. Vol. 1. IEEE, 191–194.

Ms. Ekta Gandotra is working as Assistant Professor in CURIN (Chitkara University Research and Innovation Network), Chitkara University, Punjab, India. Earlier she worked in software industry. Her research interests include cyber security, cyber threat intelligence, machine learning and big data analytics.

Dr. Divya Bansal is Associate Professor with PEC University of Technology, Chandigarh, India. She is also the Coordinator for Cyber security Research Centre, Chandigarh. She has completed her Ph.D. in the area of security in wireless networks from PEC. Her research interests include cloud security, cyber crime & investigations, cyber warfare & cryptology and wireless networks.

Dr. Sanjeev Sofat is a distinguished Professor and Head of the Department, with PEC University of Technology, Chandigarh. He is also the Coordinator of Cyber Security Research Centre, Chandigarh, India. His current areas of research are computer networks and security, image processing and biometric security.