Survey on Representation Techniques for Malware Detection System

Article *in* American Journal of Applied Sciences · November 2017

DOI: 10.3844/ajassp.2017.1049.1069

CITATIONS
12

READS
523

2 authors, including:

Gamal Abdel Nassir Awad Ali Mohamed
Muscat College, Oman - Affiliated to Stirling University - UK
7 PUBLICATIONS **16** CITATIONS

SEE PROFILE

American Journal of Applied Sciences

Review

Survey on Representation Techniques for Malware Detection System

[1,2]Gamal Abdel Nassir Mohamed and [1]Norafida Bte Ithnin

[1]Faculty of Computing, University Technology Malaysia, Skudai, Malaysia
[2]Department of Computing, Muscat College, Muscat, Sultanate of Oman, Oman

Article history
Received: 04-12-2016
Revised: 04-03-2017
Accepted: 08-07-2017

Corresponding Author:
Gamal Abdel Nassir Mohamed
Faculty of Computing,
University Technology
Malaysia, Skudai, Malaysia
Email: gamal.utm@gmail.com

Abstract: Malicious programs are malignant software's designed by hackers or cyber offenders with a harmful intent to disrupt computer operation. In various researches, we found that the balance between designing an accurate architecture that can detect the malware and track several advanced techniques that malware creators apply to get variants of malware are always a difficult line. Hence the study of malware detection techniques has become more important and challenging within the security field. This review paper provides a detailed discussion and full reviews for various types of malware, malware detection techniques, various researches on them, malware analysis methods and different dynamic programming-based tools that could be used to represent the malware sampled. We have provided a comprehensive bibliography in malware detection, its techniques and analysis methods for malware researchers.

Keywords: Malicious, Malware Representation, Detection Techniques, Analysis Methods, Dynamic Programming

Introduction

Representation of malware basically deals with how the collected malware samples are being transformed from specific format to another by applying certain techniques to represent them such as a system call representation technique (Mehdi *et al*., 2010) and Opcode sequences as representation of executable data-mining-based (Santos *et al*., 2013). In this study, besides providing the researchers with full and comprehensive literature on malware definitions, types, various detection techniques and methods, we aim at giving researchers an idea about various techniques that are used for representing the malware samples as we will conduct a deep survey on some representations that are based on the major malware detection techniques. We are proposing the string representation technique as a solution for some drawbacks/disadvantages caused by representation techniques been used by researchers in the recent years our survey will be focused or limited to analysing some techniques based on machine learning, data mining, API call graph and String (Signature based technique). The proposed string representation will be based on these representation techniques drawbacks/disadvantages. Our survey will touch upon some of the tools that are used in implementing some of these and other malware representation techniques as we will focus on the dynamic programming tools since they have been used in API call graph String and some other malware system designed recently by a considerable number of researchers. So with respect to various representation techniques, questions such as: How far these techniques or models have achieved largest detection rates? What is the best level of accuracy that the various detection models have reached? What types of programming tools available to carry out the detection system designed with these techniques? Are of various research challenges on the field of malware detection. With the increased concern towards the various vulnerabilities that cause unavailability of the network resources, malware is considered as one of the serious threats that violate confidentiality, integrity and availability of the system. It has tremendous negative impact on the computer security. The antivirus systems development is in increasing process as viruses are still considered to be of great threats and can harm our application programs and systems as well. With the complexity of the malicious software and its ability to harm and infect computer systems research has been conducted and the process is still going on by researchers on computer security field to deal with all the threats caused by this malicious software (Kumar *et al*., 2010). The software is set to be malicious as it disrupts computer operation, gather sensitive information, or

gain unauthorized access to a computer system (Santos *et al*., 2013). In the case of the malware infecting executable codes when this happen the file or files got infected will be residing in memory the moment the user executes them and hence they will infect any other files that the user may execute afterwards. It has a tremendous negative impact on the computer security (Mehdi *et al*., 2010). With the use of antivirus programs and firewall many malware could be defeated "to some extent" especially those been very active through the network, but at the same time, if we disable the use of antivirus and firewalls for a single day, this may show strong proof of the fast spread of this malicious software. The techniques that are being developed and applied by the researchers to detect the malware are clearly explained and known through the malware detectors. Of no doubt, malware has grown in volume and complexity and this has proven and experienced the danger of this malicious software. It is found that 1 in 8 legitimate Websites has a critical vulnerability, (Patil and Patil, 2015). Following a pull-based model, as the technique uses two categories of web infection by means of delivering malware into it as first various social engineering techniques are used by the attackers so that they can attract the visitors to download the malware. The second category involves the underhanded tactic of targeting various browser vulnerabilities to automatically download and run i.e., unknowingly to the visitor, the binary upon visiting a website (Mavrommatis and Monrose, 2008) last but not least, there exist a gap between the various detection techniques and the representation methods used. This is very clear as in the case of API call graph which has a major issue related to its NP-Complete problem because of its computational complexity and also during the construction phase of the graph it is difficult to build a precise call graph from information collected about malware samples (Patil and Patil, 2015).

Types of Malware

As stated above malware as a malicious software is a set of codes that are designed by hackers with an intent to harm others information or computer systems. With this background in mind and as it is shown in the above Fig. 1 we can classify the malware into different categories or types. In the above Fig. 1, the researchers have given a category for different types of malware in terms of their existence and spread and activation during the period that prior to the year 2011. From the percentages shown we can easily say that the Trojan horses followed by Virus have taken the maximum percentage whereas other malware has taken some different ranges of percentages.

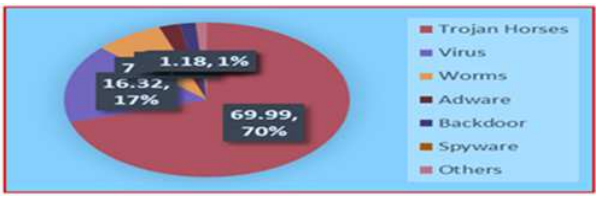

Fig. 1: Types of malware by categories (Piyanuntcharatsr *et al*., 2015)

Viruses

It has been defined by some researchers as pieces of code that insert itself into other program(s) known as virus host and get replicated. This host is very important and necessary for the virus so as to cause harm to the computer or data. As Idika and Mathhur (2007) emphasizes, a virus may use some utility software such as word processing application and attach itself to it, once the user lunched that utility software the virus will get activated and it may reach the level of disabling the malware detectors enabled in that particular computer system. There are viruses which can evolve into various types by duplicating itself. These types of viruses are known as metamorphic viruses. Unlike most of other malware the virus has disadvantage that it cannot be active unless it/its host been launched/executed, but same time the dangerous of it is driven by the fact that in order to get spread it needs a host so as to attach itself to and then do its harmful job, this host could be a useful file created by the user using any of the computer languages. Once the file is launched the virus will get activated and it can go up to the level of disabling any detection utility that has been installed in the computer system for the sake of detecting any malware. Hence basically a virus is a malware that does self-replication into other existing files or programs that could be executed and this action is repeated by the infected file for other uninfected files to make the virus spread within the same computer system or even can shift between various computers through usage of infected external media such as USB, CD/DVD, Floppy Disk and so on. It is also a well-known fact that the virus cannot simply spread through the network as this is done by another type of malware, discussed below, called worm. As mentioned above the two types of virus classification, in this case, can be classified as polymorphic and metamorphic malware.

Polymorphic Virus

In this type of virus, the morphing will be done for the code to decrypt malware after every infection. The functionality of the code will not be changed while morphing as only the internal structure of the code will get changed completely (Rad *et al*., 2012). In this case, by implementing signature based detection the encrypted malware could be recovered or resolved.

Metamorphic Virus

In this type of virus, the internal structure of the malware gets changed after every execution with its overall functionality remains the same, which could be considered an advanced version of the polymorphic virus (Lin and Stamp, 2011). For morphing the internal structure of the malware there are different approaches been followed, they could be discussed as in the case of 'Subroutines Permutation' where the subroutine definitions get arranged by malware writers within the code, as the time of maintaining the ordering of subroutine calls this result into generating codes that have equal functionality but different in their structure. Another approach is what is called 'Instruction Recording' where instructions are rearranged in a program to generate the morphed copies (Rad *et al*., 2012). Further, when an instruction or sequences of instructions are replaced by its functionality equivalent instruction in order to generate new variants of same code this is called as 'Instruction Equivalence' (Walenstein *et al*., 2007). Last but not least an approach some junk code or dead codes otherwise called do-nothing instructions which are basically inserted into software which will not affect the execution of the program, these dead codes are added to facilitate code obfuscation (Al Daoud *et al*., 2008). Certain algorithms have been used such as: Feng-Doolittle algorithm, a clustering algorithm, Prim's algorithm and MSA construction algorithm.

Worms

By exploiting operating systems vulnerabilities this type of malware spreads over computer networks. Although they are been classified as viruses, but the main different between them and the viruses is that they have the ability to self-replicate where viruses need human activity to spread in the computer system, in this sense, the propagation of computer worms is often based on the sending of mass emails with infected attachments to user's contacts (Rey, 2015). The worm replicates itself by executing its own code independent of any other program. The primary distinction between a virus and a worm is that a worm does not need a host to cause harm. Another distinction between viruses and worms is their propagation model. In general, viruses attempt to spread through programs/files on a single computer system. However, worms spread via network connections with the goal of infecting as many computer systems connected to the network as possible (Idika and Mathur, 2007).

Trojan Horses

This malicious software will be embedded by the one who has designed it i.e., hackers for example, in any kind of application or even a system that is intended and appears to perform some function or action which apparently useful like for example give the information about the local weather but in fact it is performing some other action for example collecting whatever possible information about the user who has used the application or the computer system and not only gazer them but also send them to a host which is reported to be malicious one, in this case, such type of Trojan horses could be classified as spyware as well (Landwehr *et al*., 1994).

Spyware

The personal information is not only gathered by the malicious software either by replicating or morphing or through the network, the term spyware could refer to such software which monitors and gathers personal information about the user as when he/she visited the particular page(s) frequently or accessing of email address or even the continuous usage of the credit card number or key pressed by user and so on. Moreover, it enters the system whenever free trial software is downloaded (Vinod *et al*., 2009).

Adware

It is again malicious software but unlike Spyware it appears whenever an advertisement is played or implemented by particular software automatically this could be referred to what we call Adware. By monitoring the Internet user's' activities, for example, many malware developers can add any Adware to any software that supports the add-ins, the most common adware programs according to Vinod are free games, peer-to-peer clients like kaZaa Bear Share etc (Vinod *et al*., 2009).

Botnet

As shown in the above Fig. 2, the fact behind this malicious software is that there will be number of computers (Botnets) get infected with a malware known as bots which will send orders through a Command and Control (C and C) to the botnets through its unique characteristic (Lee *et al*., 2010a). The usage of the botnet was just for the sake of mere vandalism and then it has been switched to financial revenue goals by criminals (Satrya *et al*., 2015). According to a survey done by Vinod P. and others they have clearly pointed out that the botnet is a remotely-controlled software-collection of autonomous software robots. It is also a fact to mention that botnet is usually a zombie program (Worms, Trojans) under common control on public and private network infrastructure, hence with this clear concept and remote technique of the botnets nature in sending spam/spyware. The bot looks for the communication with similar instances of bots awaiting instructions and doesn't sit on the infected machine and wait instruction from a third party. The survey has shown how the configuration is taking place and also the nature of it with respect to the bot as simplest bot configuration is where the bots are connected to the single central hub. This configuration does not scale

much because maintenance of various connections over single server is difficult. The next configuration is a hierarchical structure where bot master connects to hundreds of bots which in turn is connected to many bots. Thus this configuration could scale much larger extent (Vinod *et al*., 2009).

One of the most powerful ways to pursue any computationally challenging task is to leverage the untapped processing power of a very large number of everyday endpoints. This is the idea behind the modern botnet: A collection of compromised workstations and servers distributed over the public Internet, which jointly serve the agenda of a malicious or criminal entity. Once infiltrated with malware in a variety of ways, these compromised systems ("bots") typically link back to a Command and Control (C and C) server and wait for instructions. The botnet can then be used for tasks ranging from Distributed Denial of Service (DDOS) attacks to spam-marketing on a mass scale and collecting sensitive credit card/financial data leading in short order to identity theft and fraud (Lee *et al*., 2010b).

Ransomware

According to a research Pathak and Nanded (2016) they have defined Ransom ware as such kind of malware that attempts to extract money from a computer user by contaminating and taking control of the victim's machine or the files or documents stored on it as they use the encryption of all those user's files so as to prevent the user from using them unless a ransom is paid. They have preceded more in their introduction to explained that in general, the Ransomware will performs some actions to the user's computer or data as it either locks the computer to prevent normal usage or encrypts the documents and files on it to prevent access to the documents and files. The ransom demand is displayed, usually either via a text file as shown in the above Fig. 3 or as a web page in the web browser. This type of malware exploits the victim's embarrassment or fear to force them to pay the ransom demanded. (Pathak and Nanded, 2016).

Rootkit

With this type of malware the targeted computer is been accessed or controlled remotely without giving the user or other security program any chance to detect it. So because of their stealthy operation their prevention, detection and removal are difficult (Rey, 2015). It has been also defined as a malicious code that is designed to hide the presence of other malware. They are usually combined with other malware such as a backdoor, so that remote access could be performed by the attacker so that the detection of it becomes very difficult (Satrya *et al*., 2015).

Fig. 2: The structure of a botnet (Lee *et al*., 2010a)

Fig. 3: A sample of a typical ransom ware (Elhadi *et al*., 2012)

Backdoor

It is referred to the concept of the malware that installs itself into a computer enabling the attacker to access the computer. They allow an attacker to connect to a computer with little or no verification and execute a command on the local system (Satrya *et al*., 2015). It could be further defined as a mechanism which bypasses a normal security check.

Scareware

It is a kind of malicious software that is designed to fright the infected user to buy something it has an interface that makes it look like an antivirus. This malicious software informs the user that they are attacked by a virus and the only way to clean it is to buy their software (Satrya *et al*., 2015). More specifically it could be also defined as malicious computer programs intended to trick a user into buying and downloading needless and possibly dangerous software, such as fake antivirus protection.

Some other types of malware are such malicious codes that are doing different types of harm and they are many in nature and type but just, for example, some of them are like the key-loggers which are almost invisible application that easily creeps onto computer and perform actions such as recording everything from key store to clicking model, another malicious software is dialer malware which is a piece of software designed to dial a telephone number automatically, also browser hijacker could be considered as a type of malicious software able to, without a user's permission, alter a web browser's settings and in order to insert undesirable advertising into the user's browser it may replace the existing home page, error page, or search page with its own. Generally,

its main aims may be related to force hits to a particular website so as to increase its advertising revenue.

According to (Aycock, 2006) some types of malware have been pointed out with a clear explanation for each of them since some of them have already been mentioned above, hence herewith we are touching upon the rest of those he has mentioned and not given by us above.

Logic Bomb

It is malicious software which consists of two parts; payload which an action to perform and a trigger which is a Boolean condition that is assessed and controls when the payload is executed. Logic bombs can be inserted into the existing code or could be standalone (Aycock, 2006).

Hybrid, Droppers and Blended Threats

It is a kind of malware which has got different hybrid characteristics of different malware and the chance to exist will be given by the software itself. An example for that is the "Thompson's compiler trick" shown in Table 2 below (McGraw and Morrisett, 2000). A dropper is another combination of malware which leaves behind or dropped other malware. When a virus tempt to propagate itself as it exploits a technical vulnerability, this what is known as a blended threat in addition to exhibiting "traditional" characteristics (Aycock, 2006).

Zombies

They are considered as a malware from the nature of their link with spreading it i.e., since they are defined and known as such computers that have been cooperated as the attacker can use them for different tasks and their main common tasks here is to send spam and participating in coordinated, large-scale denial-on-service attacks (Aycock, 2006). We have taken some of the statuses of that malware as stated by the book writer and designed a table which includes some comparison of the same so as to give the researcher a clear picture of the nature of the activity of such malware the table is shown as below.

The above Table 1 shows how the status and functionality of different types of virus is taking place with respect to its ability to self-replicating, population growth and the nature of its existence or behaviour.

Table 1: Status and Functionality of some Malware (Aycock, 2006)

	Status and functionality		
Virus name	Self-replicating	Population growth	Parasitic
Logic bomb	No	Zero	Possibly
Trojan horse	No	Zero	Yes
Back door	No	Zero	Possibly
Virus	Yes	Positive	Yes
Adware	No	Zero	No

Table 2: Some concrete examples of malicious code (McGraw and Morrisett, 2000)

Malicious code	Date	Category	Explanation
Love Bug	2000	Mobile code virus	The fastest spreading virus of all time used VB script and Microsoft Outlook mail to propagate. Caused an estimated $10 billion in damage
Trinoo (and other dDoS scripts)	2000	Remote control attack script	The highly-publicized denial of service attacks of February 2000 was carried out by remotely-planted agent programs
Melissa	1999	Mobile code virus	The second fastest spreading virus of all time used email to propagate. Infected over 1.2 million machines in a few hours
Explore. Zip	1999	Mobile code worm	An e-mail borne worm that exploited problems in Microsoft windows to propagate
Happy99	1999	Virus	A widespread virus infecting Microsoft PCs
CIH	1998	Virus	A particularly dangerous virus that attacks BIOS in PCs. Ran rampant in Asia before being contained
Back orifice	1998	Offensive code	Remote control program installed on Windows machines by crackers. Pervasive
Attack scripts	1998	Offensive code	Crackers called "script kiddies" download malicious code from the Internet and run it against any number of targets. Some expert must create and release the script, to begin with Widespread Most common attack: Buffer overflow
ActiveX (scripting)	1997	Mobile code	Decried by security professionals, Microsoft's ActiveX system introduces grave security risks by relying on user's discretion and judgment
Java attack applets	1996-1999	Mobile code	Attack applets placed on Web sites take advantage of flaws in the Java security model to carry out attacks. 17 known attacks
Morris worm	1988	Worm	Released in 1988 by Robert Morris, Jr, this program affected around 6000 computers (around 10% of the Internet at the time)
Thompson's compiler trick	1984	Trojan Horse	Ken Thompson introduced a Trojan Horse in a C compiler that compiled itself into future programs [Tho84]

Also referring to the nature of virus, it is very clear that only the virus as a malware has the ability to do self-replication whereas other four types of malware have not. Also, it has a positive tendency to achieve a considerable amount of population growth where others show zero ratios these two has resulted into virus been classified as parasitic where other ranges from possibility to be to not to be. *Is it difficult to detect a malware?* It is an interesting question that might arise here and as we are addressing it we can very much say that detecting malware has become a difficult task just because of the transparent and different types of known and unknown malicious software is playing a main part in this (Christodorescu and Jha, 2004) have focused on testing anti-virus software in order to address the issue of testing detectors of malicious software (such as commercial virus scanners), but it is observed that generally their techniques are applicable to other types of malware detectors. In order to understand the difficulties in testing malware detectors they claim that one has to understand the obfuscation-deobfuscation game that malware writers and developers of malware detectors play against each other. They argue that malware detectors deploy better detection algorithms as advance malware writers detection techniques use better hiding techniques to evade detection (Christodorescu and Jha, 2004).

They have taken the polymorphic and metamorphic viruses as examples to justify their argument which that since these two types of viruses are specially designed to avoid detection tools. This could be discussed in more details so as to give the writer a clear picture about the difficulty of detecting the malware. A polymorphic virus follows the technique of morphing itself so as to avoid detection. By encrypting the malicious payload and then decrypt it as the execution time is taken place this is a common technique to "morph" viruses. To obfuscate the decryption routine, several transformations are applied such as register reassignment (permuting the register allocation) and nop-insertion, code transposition (changing the order of instructions and placing jump instructions to maintain the original semantics). In case of metamorphic viruses it is observed that they attempt to evade heuristic detection techniques by using more complex obfuscations. The viruses changes their code when these malicious software replicate, they change their code in a different ways, such as code transposition, substitution of equivalent instruction sequences, change of conditional jumps and register reassignment. It is also been seen that they can "weave" the virus code into a host program, making detection almost impossible by traditional heuristics and this because of the fact that the virus code is mixed with program code and the virus entry point is no longer at the beginning of the program (these are designated as entry point obscuring viruses) (Christodorescu and Jha, 2004). Here two questions could be raised looking into given the obfuscation-deobfuscation game and the code reuse practiced by virus writers:

Question 1: With respect to the obfuscations or variants of known malware, how resistant is a malware detector?

Question 2: Can a hacker or a blackhat1 determine its detection algorithm using limitations of a malware detector in handling obfuscations?

The motivation for the first question exists by the obfuscation-deobfuscation game. Whereas the second question is motivated by the fact that if a blackhat knows the detection algorithm used by a malware detector, they can better target their evasion techniques. In other words, the "stealth" of the detection algorithm is important (Christodorescu and Jha, 2004). In the same line of argument McGraw and Morrisett provide detailed descriptions of various types of malware. They have noted that categorizing malicious code has increasingly become more complex as newer versions appear to be combinations of those that belong to existing categories. They have provided a table which includes some concrete examples of malicious code is provided and they have also given a note that "recent versions of malicious code are really amalgamations of different categories" (McGraw and Morrisett, 2000). Although the statistic has some old historical background but still it provides the researchers with good view and helps in giving a clear idea about the concept of some categories of used malware and this certainly will support the idea of how detecting malware was and still a difficult task which needs a proper techniques to be used in order to design or develop models that can act as a services or tools of detection. With this background, the table provided below which was given McGraw and Morrisett (2000) that shows some explanation for some malicious code has been taken on different dates and from various categories. In order to discuss the content of the above table given McGraw and Morrisett (2000) on their research titled "Attacking malicious code," our first attention as researchers go to the year 1984 which is the eldest year among the other years provided i.e., on the range starting from 1984 to 2000. Looking to this year we found that malicious code been taken was categories as Trojan Horse this indicates that this malware was very strong and active since that time especially if we look into the fact that the introducer of it has used it for a C compiler which is one of the high-level languages. Also, our attention goes to some malicious codes that are related to attacking mobile devices during a pretty long time from now which is 1996 to 1999. The category has also included a malicious code of remote control nature.

Malware Analysis Methods

Malware analysis is a very important process that will determine the purpose and functionality of a given malware sample (such as a virus, worm, or Trojan

horse). It is an important prerequisite for the development of removal tools that can thoroughly delete malware from an infected machine; it is also considered being a necessary step to develop effective detection techniques for malicious code. Traditionally, it has been a tedious manual process which is time-intensive. Unfortunately, the number of samples that need to be analyzed by security vendors on a daily basis is constantly increasing (Bayer *et al*., 2006). One of the most important multi-step processes that provide insight into malware structure and functionality is malware analysis. In this analysis process, there is an important step which is used to observe malware relations with respect to the system and is achieved by employing dynamic coarse-grained binary-instrumentation on the target system; this step is known as behavior monitoring. At the same time, when an initial examination is performed for the collected malware, this is called as profiling. Malware analyses can categories fall into two main methods; one is based on analyzing a given sample during execution which is called dynamic analysis, while another one refers to analyze a sample by extracting useful information without executing it and this is known as static analysis. A brief introduction to both types with the knowledge of their limitations as an approach to malware analyses is discussed. It is a fact that from a traditional point of view in the case of malware detection there are two major approaches which are basically split based on how it will analyze the malware, they are hence static or dynamic analysis (see review (Egele *et al*., 2012). In the case of static analysis, their way of analyzing this malware is done as a direct analyze in binary form, or additionally unpacked and/or decompiled into assembly representation. Whereas in the case of dynamic analysis, it is done through hooking or some access into the internals of virtualization environment the binary files are executed and the actions are recorded. So it is valid to mention that in the case of dynamic analysis it can provide observation of malware action which is less vulnerable to obfuscation (Moser *et al*., 2007) and makes it harder to recycle existing malware. However, in practice, automated execution of software is difficult, since malware can detect if it is running in a sandbox and prevent itself from performing the malicious behavior.

Hence for machine learning approaches, this makes the static analysis more amenable and as data size increased this will result and enable a better performance (Banko and Brill, 2001). Machine learning has been applied to malware detection since that fact is known (Kephart *et al*., 1995), so this has proven the fact that numerous approaches exist since that moment onwards (see reviews (Egele *et al*., 2012; Gandotra *et al*., 2014)). Machine learning consists of two parts, the feature engineering, where the author transforms the input binary into a set of features and a learning algorithm, which builds a classifier using these features.

Static Analysis

Static analysis refers to as such analysis done on the infected file without executing it, here the (CFGs) which is an abbreviation for Control Flow Graphs needs to be extracted as well as Data Flow Graphs (DFGs) along with System call analysis. Such information related to CFGs and DFGs can be gathered by disassembling or decompiling the infected file using some tools like IDA Pro. The file may have auto execution which is against the actual concept of static analysis this could be overcome by analyzing the infected file in a different environment to avoid this auto execution of the malware. The advantages of static analysis may be pointed out into points such as it results in fast, safe and low false positives with the possibility of tracing all paths which will certainly help in getting lot of information to analyze, on the other hand, static analysis may fail analyzing unknown malware that uses code obfuscation techniques (Egele *et al*., 2012).

Dynamic Analysis

In the case of this type of analysis the analysis is done for the infected file during its execution. This type of analysis uses a debugger or virtual machine or even an emulator to execute the infected file on simulated environment in order to analyze its malicious functions, as we have mentioned the infected file needs analysis environment which must be invisible to the malware for the simple reason that the malware writer have tools like anti-virtual machine and anti-emulation used to hide their malware functions if their detection been under analysis. It has been observed that this type of analysis i.e., dynamic one fails to detect activities of interest if the target changes its behavior depending on trigger conditions such as the existence of a specific file or specific day as only a single execution path may be examined for each attempt (Egele *et al*., 2012).

As shown in the above Table 3 both analysis types have their advantages and disadvantages. For the static one since it is based on analyzing the infected file without executing it, this results in fast and safe and low level of false positives with good analyzing multipath malware. On the other hand, it is difficult to analyze unknown malware when following the static analysis. For the dynamic analysis which is based on analyzing the infected file after executing it, this shows good at detecting unknown malware with slow and unsafe with difficulty in analyzing multipath malware as main disadvantages, whereas Hybrid analysis combines aspects of both static and dynamic analysis (Elhadi, 2014).

Table 3: Adv. and disadvantages of static and dynamic analysis (Elhadi, 2014)

Analysis type	Advantage	Disadvantage
Static analysis	Fast and safe but low level of false positives Good at analyzing multipath malware	Difficulty analyzing unknown malware
Dynamic analysis	Good at detecting unknown malware	Slow and unsafe
Hybrid analysis	Combines aspects of both static and dynamic analysis	Difficulty analyzing multipath malware

A Comparison of Static, Dynamic and Hybrid Analysis for Malware Detection

Static analysis of software is performed without actually executing the program. Examples of the information we can obtain from the static analysis include Opcode sequences (extracted by disassembling the binary file), control flow graphs and so on. Dynamic analysis requires that we execute the program, often in a virtual environment. Examples of information that can be obtained by dynamic analysis include API calls, system calls, instruction traces, registry changes, memory writes and so on. Hybrid techniques combine aspects of both static and dynamic analysis (Damodaran *et al*., 2015; Elhadi *et al*., 2015). The below Fig. 4 shows the various methods of malware detection techniques which is basically divided into three categories i.e. static, dynamic and hybrid.

In the below Table 4, the analysis types with their main purpose and the corresponding tools that could be used in association with each analysis type are clearly shown and explained. For instance, if we look into the static one we clearly find that it uses as many antivirus detection engines as possible to assist classification which corresponds to using the virus total as the main tool to implement the same. It also uses strings tool when it wants to search the body of the malware for the string. The dynamic uses various tools for various purposes of malware analysis (Verma *et al*., 2013). Its purposes shown in the Table 4 could be further discussed as with respect to file integration it checks to record baseline configuration. In the case of File monitoring it finds which tools are opening, reading and writing files. Whereas Process monitoring determines resources that are being used such as DLL's and registry keys. Network monitoring uncovers which ports are open, collect network traffic and find vulnerabilities. Last but not least, Registry monitoring monitors registry activities as they occur. In the case of this dynamic analysis certain tools are been used such as: Analysis, Filemon and Process explorer. Analysis type based on coding its main purpose is to perform disassembly and debugging (Verma *et al*., 2013).

Malware Detection Techniques

It's obviously well known fact that malware writers or cyber offenders, otherwise known as hackers, applying some certain sophisticated techniques to evade detection. They follow methods that modify or morphing the malware by means of using several packing and/or program obfuscation techniques. Here comes the concept of malware detector as a system attempts to identify malware using both signatures and other heuristics techniques. The antivirus scanner, for example, is a good example of a malware detector (You and Yim, 2010). The way of detecting a malware may take different approach as in the case of commercial malware detectors such as virus scanners for example which uses a simple pattern matching approach to detect the malware whereby a program is declared as malware if it contains a sequence of instructions that matched by a regular expression (Idika and Mathur, 2007). This has been discovered by a recent study that such malware detectors can be easily defeated using simple program obfuscations that are already known and been used by hackers. One important point that forces the database of commercial virus scanner to be updated frequently is related to the fact that the pattern-matching algorithm is not very resilient to slight variations, these malware detectors have to use different patterns for detecting two malware that is slight variations of each other (Christodorescu and Jha, 2004).

Detection is mainly dealt with the art of knowing how to recognize and locate the malware in its existence location whether been on a system, in a file on that system or in software or hardware or even a media that still to be installed on the system. It is so useful to detect this malicious software at the early stage which will surely help in preventing it from harming the information that may be reachable it. Once we are able to detect this software at the earliest, then this will be a great step towards minimizing the number of infected systems which will result in less effort paid while doing the recovery process and also less level of damage the organization sustains. Malware detection techniques can be divided into three methods; signature-based, behavior-based and specification based each method can be applied using static analysis or dynamic analysis or hybrid analysis (Idika and Mathur, 2007). The Signature-Based is a sequence of instructions unique to a malware used to generate a malware signature, (this signature is captured by researchers in a laboratory environment), a signature should be able to identify any malware exhibiting the malicious behavior specified by the signature and most antivirus scanners are signature based. Whereas the Behavior-Based detection techniques focus on analyzing the behavior of known and suspected malicious code. Such behaviors include factors such as the source and destination a][ddresses of the malware, the attachment types in which they are embedded and statistical anomalies in malware infected systems. One example of a behavior-based detection approach is the histogram-based malicious code detection technology patented by Symantec. Signature refers to the same instructions, same basic blocks and same system calls, whereas behavior

refers to the similarity between system call sequences and their relations. There is a main limitation been observed in the case of specification-based detection which is related to its difficulty to completely specify the entire set of those behaviors which could be marked as valid, whereas in the case of signature-based since it cannot detect zero-day attack, for which there is no corresponding signature stored in the database which has been treated as one of the major drawbacks of it (Moser *et al*., 2007). In the case of specification-based techniques, it leverages some specification to decide the maliciousness of a program been under inspection and that by deciding what is the valid behavior is and how it looks like from the behavior point of view. Signature based detection uses its characterization of what is known to be malicious to decide the maliciousness of a program under inspection (Ellis *et al*., 2004; Han *et al*., 2014; Shabtai *et al*., 2009). The procedure of helping protect the system by detecting malicious behavior consider the fact that the malware detector may or may not reside on the system it is trying to protect, in fact the malware detectors takes two inputs which could are related to the signature or behavior of malware to be identified and known as well as the program that currently under inspection (Riesen *et al*., 2010). Each of them can be applied using any of the methods either static analysis which is implanting signature/behavior/specification-based detection technique without executing the suspected file and it was the first try to detect malware or dynamic analysis, it is applying all of them i.e., signature/behavior/specification-based detection technique during suspected file execution or even hybrid analysis (Idika and Mathur, 2007).

Hence to further explain these techniques and their nature we can say they have and could be categorized as anomaly-based detection, specification-based detection and signature-based detection (Bergroth *et al*., 2000), (Skormin *et al*., 2003; Summerville *et al*., 2005). Anomaly-based detection techniques use the knowledge of what constitutes normal behavior to decide the maliciousness of a program under inspection. Specification-based techniques leverage some specification of what is a valid behavior to decide the maliciousness of a program under inspection. Signature based detection uses its characterization of what is known to be malicious to decide the maliciousness of a program under inspection (Ellis *et al*., 2004; Han *et al*., 2014; Shabtai *et al*., 2009). The fundamental limitation of anomaly-based detection is its high false-positive rate and the time complexity in the training phase. The main limitation of specification-based detection is that it is difficult to specify completely and accurately the entire set of valid behaviors. One of the major drawbacks of signature-based detection is that it cannot detect the zero-day attack, for which there is no corresponding signature stored in the database (Rieck *et al*., 2011).

We can then conclude to the fact that those techniques used for detecting malware can be categorized broadly into three categories: Anomaly-based detection, specification-based detection and signature-based detection. Anomaly-based detection techniques use the knowledge of what constitutes normal behavior to decide the maliciousness of a program under inspection. Specification-based techniques leverage some specification of what is a valid behavior to decide the maliciousness of a program under inspection. Signature-based detection uses its characterization of what is known to be malicious to decide the maliciousness of a program under inspection.

The fundamental limitation of anomaly-based detection is its high false-positive rate and the time complexity in the training phase. The main limitation of specification-based detection is that it is difficult to specify completely and accurately the entire set of valid behaviors. One of the major drawbacks of signature-based detection is that it cannot detect the zero-day attack, for which there is no corresponding signature stored in the database (Zhang and Xia, 2012).

Research Techniques for Malware Detection System

The static analysis which is implementing signature based detection without executing the suspected file was the first attempt to detect malware. Some of the researchers who used this approach applied Objective-Oriented Association (OOA) mining based classification (Ye *et al*., 2008). The proposed model has three major modules, namely, Portable Executable (PE) parser, OOA rule generator and rule-based classifier. This model was taken to a next level by adopting associative classification method based on the analysis of Application Programming Interface (API) execution calls (Ye *et al*., 2010).

Other researchers combined signature-based technique and genetic algorithm technique and their study focused on three types of malware which are; Viruses, Worms and Trojan Horses (Zolkipli and Jantan, 2010). Signature based detection was also applied during suspected file execution (i.e., dynamic analysis) in which the researchers traced API calls and then built the suspected file signature (Nair *et al*., 2010).

These researchers generated a signature for an entire malware class instead of individual malware samples. This approach has the advantage that all the metamorphic viruses that are created from a metamorphic generator can be easily detected once a base signature for that metamorphic generator is obtained. A considerable portion of the existing studies relies on using behavior-based detection, whereas some researchers apply static analysis others apply dynamic analysis. Some of the studies adopt the mapping of kernel memory as a way to develop a monitor for malware behavior. The monitor utilizes time-based view of kernel objects to analyze traces of kernel execution (Rhee *et al*., 2011).

Other studies trace malware behavior exhibited by installer and uninstaller software as a way to avoid false positives (Rhee *et al*., 2010) and suggest a new categorization method for malware based on maximal

common subgraph detection (Park *et al*., 2010). We have observed that some of the current researchers have gone for the idea of combining static and dynamic analysis so as to overcome the drawbacks or limitations on each of them. It is a fact that some limitations can be found and observed in approaches been developed following the concepts of signature or behavior-based. It's then found that the signature-based approaches can be overcome by obfuscation and require prior knowledge of malware samples, whereas in behavior-based not able to detect a lot of polymorphic viruses present (Packers), (Elhadi, 2014). Also it is found that a higher rates of false positive are been generated and incur expensive runtime overheads (Hu *et al*., 2009). This is been clearly explained as in the Table 5 shown below.

In their research paper (Damodaran *et al*., 2015) they compare malware detection techniques based on the methods of detection i.e., static, dynamic and hybrid analysis. They have specifically done a training for the Hidden Markov Models (HMMs) on both static and dynamic feature sets and then they have done some comparison for the resulting detection rates over a substantial number of malware families and also they have considered hybrid cases in the situation where dynamic analysis is used in the training phase with the static techniques used in the detection phase and vice versa which enables them to include the hybrid analysis as well (Damodaran *et al*., 2015). Referring to their work we can conclude a discussion or findings in such a way that as the three methods of detection which are based on static, dynamic or hybrid one, each has its own advantages or contribution over the models that has been designed based on them and also their own disadvantages or limitations on the same. In the above Table 6 we discussed the various types of detection techniques and our focus will be on projecting these advantages/contributions as well as the common disadvantages/limitations of these three types of malware detection techniques in a form of analytical comparison between three of them which will result in an overall scenario that will shape their real existence in malware detection process and this will certainly help the researcher in looking into the concept of malware detection from a deep and different angles. It is clearly observed that the model, however, has some advantages but it requires a lot of effort as an up to date signature dataset is required since the malware will be expected to be present in the database so as to get detected otherwise it will not. Hence repository of signature of known malware should be maintained and this repository should be up to date as well. It is also observed that with this technique been used simple obfuscation technique can be used to evade it. The behavior based passed through phases of analyzing and then classification since first both benign and malware are analyzed during the training phase. The statistical one is mainly considered as a benchmarking process.

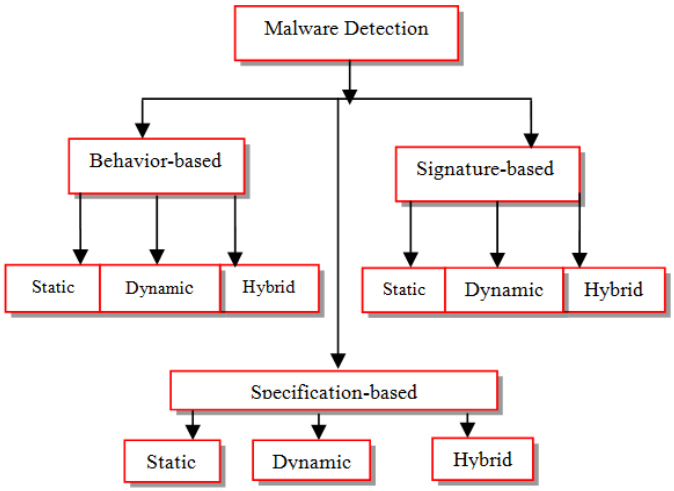

Fig. 4: Organization of malware detection techniques (Elhadi *et al*., 2015)

Table 4: Summary of some malware analysis tools (Verma *et al*., 2013)

Analysis type	Purpose	Tools
Static	Use as many antivirus detection engines as possible to assist classification	Virus Total (Virus Total, 2008)
	Search the body of the malware for the string	Strings (Microsoft, 2008c)
Dynamic	File integrity checks to record baseline configuration	Analysis (Winalysis.com, 2008)
	File monitoring finds which tools are opening, reading and writing files	Filemon (Microsoft, 2008c)
	Process monitoring. Determine resources that are being used such as DLL's and registry keys	Process explorer (Microsoft, 2008c)
	Network monitoring. Uncover which ports are open,	Fport (Foundstone, 2008), tcpview
	Collect network traffic and find vulnerabilities	(Microsoft, 2008c), nessus
	Registry monitoring. Monitor registry activities as they occur	Regmon (Microsoft, 2008c)
Code	Disassembly, debugging	IDA Pro Ollydbg

Table 5: Advantages and disadvantages of Signature and behavior techniques (Elhadi, 2014)

Malware detection types	Advantage	Disadvantage
Signature-based	less scanning time Few false-positives	Unknown malware can easily evade detection cannot deal with simple obfuscation
Behaviour-Based	Best results in detecting of Polymorphic malware	not able to detect a lot of polymorphic viruses present (Packers)

Table 6: Analysis of Malware detection techniques (HMMs) (Damodaran *et al*., 2015)

Detection techniques mentioned	Definition and nature	Advantages/contributions	Disadvantages/limitations
Signature based detection	It is the most widely used anti-virus technique A signature is a sequence of bytes that can be used to identify specific malware. A variety of pattern matching schemes are used to scan for signatures	Simple and relatively fast Effective against most common types malware	Requires an up-to-date signature database as malware not present in the database will not be detected. Relatively simple obfuscation techniques can be used to evade signature detection Must maintain a repository of signatures of known malware The repository must be updated frequently as new threats are discovered
Behavior based detection	It focuses on the actions performed by the malware during execution	Systematical behavior study of the suspected malware	Both benign and malware are analyzed during the training phase Classification of them will be only during the execution phase
Statistical based detection	Properties derived from program features as in the Hidden Markov Models (HMMs), used in their research paper, are used to classify metamorphic malware	Has served a benchmark in a variety of other studies	Visible only when using HMMs as the basis for the malware detection schemes considered in their research

The main techniques that are used In Malware Detection Systems (MDS) are related to a technique which is based on machine learning or data mining or call graph or string representation. Each of them is explained and discussed below.

Malware Detection Based on Machine Learning

It is observed that there are machine learning techniques of different nature have been proposed for the sake of malware detection, it is found that the boosted decision trees based on n-grams are been observed to produce better results than both the Naïve Bayes classifier and Support Vector Machines. In order to distinguish between malware and clean program files, some researchers have used automatic extraction of association rules on Windows API execution sequences. To figure out whether a given program file is/not a variant of a previous program file a model like Hidden Markov is used so as to detect that file.

In this regards and in order to reach the similar goal, different researchers in this field have gone for applying Hidden Markov Models, which have been previously used with great success for sequence analysis in bioinformatics. In the case of polymorphic malware, we can find that neural networks have been used for the sake of detecting the malware. Also to identify patterns of behavior for viruses in Windows executable files Self-organizing maps have been used. (Kolter and Maloof, 2006) mentioned in their research that, it has been observed that classification of malware binaries was first been studied by (Schultz *et al*., 2001) and in a research which describes the use of machine learning and data mining to detect and classify malicious executable as they appear in the wild. Researchers have gathered 1,971 benign and 1,651 malicious executables and encoded each as a training example using n-grams of bytecodes as features. Such processing resulted in more than 255 million distinct n-grams (Kolter and Maloof, 2006). The most important point that we can record and observe here is that these both approaches have used and applied string features of binary executable for training learning algorithms and distinguish between malicious and benign files. Some other researchers recently have devised an extension of the above-mentioned work to unpacked malware binaries as they have proposed Malware Collection Booster (McBoost), a fast statistical malware detection tool that is intended to improve the scalability of existing malware collection and analysis approaches. Given a large collection of binaries that may contain both hitherto unknown malware and benign executable, McBoost reduces the overall time of analysis by classifying and filtering out the least suspicious binaries and passing only the most suspicious ones to a detailed binary analysis process for signature extraction. The McBoost framework consists of a classifier specialized in detecting whether an executable is packed or not, a universal unpacker based on dynamic binary

analysis and a classifier specialized in distinguishing between malicious or benign code (Perdisci *et al*., 2008). On the same line, some others have proposed analyzing the content of the file for detection of malware samples been embedded within files and that is done basically by making use of similar techniques (Li *et al*., 2007). Hence in this regards it is very much relevant to mention that for malware behavior the first application of classification has been introduced by researchers who have to generate evidence and classify the samples with several machine-learning algorithms (Devesa *et al*., 2010). Then a research of the same field has been extended and gone in another line as the focus was on the clustering of malware behavior for the sake of discovering the novel malware and reduction of manual analysis effort, in this regard the first clustering system for observed behavior was introduced, this was then later got extended by another research for the main reason to make it more scalable, where the system devised provides an excellent performance in when it comes to run-time while it is in the practice stage (Bayer *et al*., 2010). However, both approaches require a single batch of malware samples and thus are limited in the overall capacity.

Malware Detection Based on Data Mining

Detecting unknown viruses researchers have also pay a great attention to the representation of malware detection based on data mining. It is observed that a high accuracy rate is found when a number of classifiers have been built; furthermore it's been extended to the research level where it become so common to apply data mining techniques for malware detection as it depends on generating a feature set which includes hexadecimal byte sequence which is otherwise termed as N-grams. It also includes instruction sequences, API/system call sequences etc. (Siddiqui *et al*., 2008). Practically speaking, in the case of features methodology the number of features been extracted from the targeted files is usually very high this has directly/indirectly made several techniques from text classification have been employed to select the best features whereas in other hand features that may include printable strings extracted from the files and some operating system dependent features such as DLL information. When collecting the data the activity monitoring methods may be used, but still, data mining remains the principal detection method. Here we discuss the concept of N-grams which is basically a sequence of bytes of fixed or variable length, extracted from the hexadecimal dump of an executable program. They are used as a general term for both overlapping and non-overlapping byte sequences. They have been defined to remain at a syntactic level of analysis. The first major work that used data mining techniques for malware research was an automation of signature extraction for viruses so it is a clear fact that the signature extraction is a major step in data mining technique and with respect to the N-grams technology here comes the concept of Dynamic Misuse Detection, Dynamic Hybrid Detection, Static Hybrid Detection and Static Misuse Detection (Siddiqui *et al*., 2008). Table 7 summarizes malware detection using data mining which differs in techniques, models, features and data set used. From the below table representation for malware detections work which is mainly related to data mining as a representation technique been used, we can observe that researchers have used different methods which have got a direct relation with the nature of the platform or the system nature that it has been used in for example clustering has been used in Network whereas classification has been used when it comes to detecting various types of malware. More (Norouzi *et al*., 2016) gave a detailed review from different researches related to some works for malware detection in data mining methods. Some researchers leverage standard data mining algorithms to classify the file content of every block as normal or potentially malicious (Tabish *et al*., 2009).

Malware Detection Based on API Call Graph

API which is an abbreviation for an Application Programming Interface which is basically a collection or set of rules or codes since it related to programming and the program can be defined as a set of codes. These particular rules and specifications are been used by the various software programs in order to communicate with each other. As the user interacts with the computer through various interfaces, hence API is an interface between those various and different software programs, that interface will facilitates the interaction of those different software programs with each other. API could be created for many applications, libraries, operating system, etc. this is true just because the way of defining their "vocabularies" and resources needs and request some conventions, for example, the function-calling conventions. Since API is a programming specifications it may include such those specification which ranges between those been used and needed for data structure, object classes, routines and up to the level the protocols used to communicate between the consumer program and the implementer program of the API itself, the API calls list is extracted from a binary executable through static analysis of the binary with disassembly tools such as IDA Pro (Elhadi *et al*., 2015). Also, it may be through dynamic analysis after executing the binary in a simulated environment, which is the technique adopted by tools such as API monitor (Monitor-Spy, 2012). It's observed that there is low malware detection accuracy exists in the case of malware detectors that based on API call graph architecture. This is basically due to some problems in both the API call graph construction and matching algorithms.

Table 7: Selected previous work in the malware detection data mining technique (Piyanuntcharatsr *et al*., 2015)

Work	Data mining/ clustering	Dataset	Network	Malware type	Method
Pratheema, Prabha - and Kavitha	Data mining (classifier: NB, KNN, J48)	100 binary (training benign = 90, malware = 10)	-	15 subfamily	Hex code/ngrams of different size
Rafique and Calballero (0000)	clustering	16,000 malware binaries	Yes	Network signature	11 network feature
Bailey et.al. (0000)	Classification	3,698	Yes	Groups of Malware	System state Changes vector
Komashinskiy and Kotenko	classification Decision Table C4.5 Random Forest Naive Bayes	Malware = 5854 benign = 1656	-	Malware and benign	<position, byte>
Choudhary and Saharan (0000)	Classification: SVM, NN	200, 500 files	-	Type 1, Type 2	Instruction sequence
Kumar and Mishra (2010)	Classification IBK	323 (virus+worm)	-	Virus, worm	Sequence alignment
Tabish *et al.* (2009)	Classification-decision tree (J48)	Benign = 1800 and malware = 10, 311	-	backdoor, Trojan, virus, worm, constructor	Statistical features and miscellaneous

The fact is that in the case of API call graph construction algorithms there is a major issue of building a precise call graph from the information collected about malware samples. Also in the case of API call graph matching algorithms it is found that they have NP-complete problems and suffer from being slow because of their computational complexity (Anderson *et al*., 2011; Han *et al*., 2012; Lee *et al*., 2010b). In malware detection, some researchers have paid attention to building their detection systems or architectures based on API call graphs because they have shown some sufficient expressiveness to model complicated structures and their use is gaining momentum in representing structural information. We will be showing in the following paragraphs some of the related work in this area. There are several approaches that the researchers have gone for so as to build the graph, as most of them present graph nodes as system calls. This is very clear as in the case of; Lee *et al*. (2010a) who have created their graph by transforming a Portable Executable (PE) file into a call graph with nodes and edges which represent system calls and system call sequence, respectively. After that, minimization is applied to the call graph turning it into a code graph to speed up the analysis and comparison process (Lee *et al*., 2010b). Whereas it is observed that some researchers use the same approach by using 4-tuple nodes to denote a system call, edges, the dependencies between two system calls and a label for nodes and edges, (Park *et al*., 2010) it is also observed as in the case of Park and Reeves 2011 some other studies use graph nodes to denote kernel objects instead of system calls. It is also seen that the graph was built from subroutines where the nodes and their corresponding call references as edges (Kostakis *et al*., 2011). On the other hand, some others researchers have used a dependency graph whose vertex represents a line in the semantic code and the dependency between two lines is represented by a directed edge (Kim and Moon, 2010). In some research, we have found that researchers have proposed an algorithm to construct a dependence call graph, where graph nodes represent system calls and two types of dependencies exist between system calls to present the edges (Christodorescu *et al*., 2008). Furthermore a researcher has proposed a graphs which are related and based on behavior with the proficiency that they share similarities with graphs of other researchers i.e., Christodorescu in their research paper which is focusing on mining specification of malware behavior, with the clear difference that the edges of their graph have been produced in a way which is totally different with the point that there is no constraint with respect to API call parameters been used. The idea of that usage of both types of research is depends on the fact that an edge is connecting node a to node when there is data dependency between API call nodes a and b, the data dependency represents when the return value of first API call has taint label that is used in one of the input parameter lists for the second API call (Christodorescu *et al*., 2008; Kolbitsch *et al*., 2009). This idea is been used later as a researcher has enhanced the dependence graphs proposed by the above two researchers following the methodology of assigning labels to particular files, directories, registry keys and devices based on their significance to the system (e.g., system startup list, firewall settings, system executable) (Christodorescu *et al*., 2008; Kolbitsch *et al*., 2009; Fredrikson *et al*., 2010). It is true and valid to come out from all the above with the fact that all the above studies have used different graph matching techniques while doing the graphs comparison process, these techniques ranges from formula building using intersection and union of graphs, weighted common behavioral graph generation based on an approximate algorithm and maximal common subgraph (Park *et al*., 2010; Lee *et al*., 2010a; Kim and Moon, 2010). One important point that forces the database of commercial virus scanner to be updated frequently is related to the fact that the pattern-matching algorithm is not very resilient to slight variations, these malware detectors have to use different patterns for detecting two malware that is slight variations of each other (Christodorescu and Jha, 2004).

In below Table 8 a thorough review of Malware Detection System using data mining is been provided with the findings been related to date mining methods. This review table will provide researchers with a clear view of data mining usage and its contribution in building logic for malware detection.

Table 8: A review of MDS using data maiming (Norouzi *et al*., 2016)

Paper title, author, Year of publication	Findings with relation to data mining methods
Schultz *et al*. (2001). Data mining methods for detection of new malicious executable	Proposed a data mining method to Recognise the new malicious files in run-time execution. Their method was based on three types of DLL calls such as the list of DLL issued by the binary; the list of DLL function calls and a number of different systems calls used within each DLL. Also, they examine byte orders extracted from the hex-dump (a hexadecimal scheme of computer data) of an executable file using signature methods. The main structure of this method is based on Naive-Bayes (NB) algorithm. They compared the experimental results by traditional signature-based methods
Kolter and Maloof (2004). Learning to detect malicious executable sin the wild	Presented a data mining approach and n-gram analysis to identify malicious executable files based on signature approach. They presented a hex-dump utility for translating each executable file to hexadecimal code in an ASCII format. Their main data set consisted of the clean programs and the malicious programs. They analyzed the proposed approach by some popular classification method such as instance-based learner, TFIDF, Naive-Bayes, support vector machines, decision tree, boosted Naive-Bayes and boosted decision tree. In the other research
Siddiqui *et al*. (2008). Detecting internet worms using data mining techniques	Proposed data mining techniques for recognition some malware programs such as Worms. They considered variable length instruction sequence for their approach. Their main data set includes some Windows files and Worms. As experimental results sequence reduction was executed, 97% of the sequences were removed and random forest decision tree model was performed slightly better than the others. Also, some research work presented the data mining methodologies for a different approach
Yang and Yi-Ping (2015). Data mining in lung cancer pathologic staging diagnosis: Correlation between clinical and pathology information	The researchers presented various data mining methods that have been developed for cancer diagnosis. Consequently, this research focused on captivating the clinical information which can be found without surgery to exchange the pathology report. They used to discover the association between the clinical information and the pathology report in order to maintain lung cancer pathologic staging diagnosis using data mining techniques.
Gandotra *et al*. (2014). Malware analysis and classification: A survey.	The authors proposed a data mining approach to analyzing the students' careers.
Campagni *et al*. (2015). Data mining models for student careers	Their approach is based on clustering and sequential methods with the aim of categorizing strategies for refining the performance of the exams scheduling and students. They analyzed a real case study using K-mean cluster techniques in WEKA tool
Bayer *et al*. (2006). Dynamic analysis of malicious code	Presented a new data mining method for the problem of detecting the phishing websites using a developed associative classification method called multilabel classifier that generates multiple labels rules. They analyzed the experimental results by various patterns in WEKA software
Rahman and Hasan (2011). Using and comparing different decision tree classification techniques for mining ICDDR,B Hospital Surveillance data.	Analyzed the several decision tree models to classify patients of the hospital surveillance data as a real case study. The experimental results of their analysis showed that their approach improved identical dissemination of instances in each class
Ghosh *et al*. (2014). A novel Neuro-fuzzy classification technique for data mining	Used a neuro-fuzzy data mining approach for classification of generalized bell-shaped membership functions. They applied the proposed technique to ten real standard datasets from the UCI machine learning repository for classification using Kappa statistic. They simulated proposed technique in MATLAB.
Moskovitch and Shahar (2015). Classification-driven temporal discretization of multivariate time series	Presented a novel managed discretization technique for analyzing multivariate time series which uses frequent temporal patterns as features for classification of time chain for geared near improvement of classification correctness. This paper used temporal abstraction classification approach and time intervals mining for the presented multivariate time series
Stopel *et al*. (2006). Application of artificial neural networks techniques to computer worm detection	Presented novel Artificial Neural Networks (ANN) based mechanism for discovering the computer Worms based on the behavioral computer events. According to the estimation of different parameters of the infected computers, the ANN, decision tree and K-nearest neighbors' classification techniques are compared
Nissim *et al*. (2012). Detecting unknown computer worm activity via support vector machines and active learning	Where the author's presented computer measurement extracted mechanism for identifying unknown computer Worm activity in the operating system using support vector approaches. This paper separates a series of trials to check the new technique by retaining several computer configuration activities

Malware Detection Based on String Representation

This technique as we proposed it, we will address it showing how our proposed model works. Our model has three phases, the first one will basically represent the various known malware in text into string format and hence identify the various functions and their related parameters, phase two will make use of the redundancy there to identify which parameters are repeatedly been called by same functions and hence once this occurrence exists, it will be checked and if it is greater than the threshold then that area will be identified as risky zone which represents the actual malware part of the whole malware file. Phase three will work with unknown malware samples and detecting them only against the risky zone or red area identified in phase two so as to perform a fast detection system which will give a high detection accuracy and be fast so no computational complexity will be experienced with it and as a result of that less memory space will be used. All of these are in fact drawbacks been noted in many of other malware representation techniques. The model we propose is a hybrid one as it makes use of signature and behavior based techniques. Another model is related to Hancock is the first string signature generation system that takes on this challenge on a large scale, Hancock is able to automatically generate string signatures with a false positive rate below 0.1%., it also proven to be given a set of malware samples, Hancock is designed to create a minimal set of N-byte sequences, each of which has a sufficiently low false positive rate, that collectively cover as large a portion of the malware set as possible this has been mentioned in their research. Griffin *et al*. (2009) published by Symantec Research Laboratories about automatic generation of string signature for malware detection, they have claimed that Hancock differs from previous work by focusing on automatically generating high-coverage string signatures with extremely low false positives (Griffin *et al*., 2009). Kreibich and Crowcroft (2004) developed Honeycomb a system that uses honeypots to gather inherently suspicious traffic and generates signature by applying the Longest Common Substring (LCS) algorithm to search for similarities in the packet payloads. As they mentioned in their research paper (Lin and Stamp, 2011) that their system is unique in that it generates signatures. In contrast to NIDSs, it cannot read a database of signatures upon startup to match them against live traffic to spot matches thus, the commonly employed pattern-matching algorithms in NIDSs which are of no use to the researchers. Instead, the system tries to spot patterns in traffic previously seen on the honey pot as the researchers have overlay parts of flows in the traffic and use a Longest Common Substring (LCS) algorithm to spot similarities in packet payloads. Like pattern matching, LCS algorithms have been thoroughly studied in the past. Their LCS implementation is based on suffix trees, which are used as building blocks for a variety of string algorithms. Using suffix trees, the longest common substring of two strings is straightforward to find in linear time (Kreibich and Crowcroft, 2004).

The Importance of Malware Detection

The topic of malware detection has taken a high level of importance in many types of research in recent years and in different types of study fields. According to the analytical data from Scopus, the study of the malware detection has a significantly increased in the last six years (St'astna and Tomasek, 2015) this is shown in the below Fig. 5.

Dynamic Programming Representation Tools

Here some dynamic programming-based tools that are currently been used in most of the representation techniques will be briefly discussed such as Basic Local Alignment Search Tool (BLAST), Longest Common Sequence (LCS) and Dynamic Time Warping (DTW) as in certain representation techniques specially those based on comparison between any two or more inputs/functions/parameters they could be used to generate the desired results based on the final algorithm that will ensure better improvement of malware detection accuracy.

Basic Local Alignment Search Tool (BLAST)

BLAST is a system which has become very popular in recent times (Altschul *et al*., 1990). While the 394 ISMB-95 Needleman-Wunsch/Smith-Waterman systems are able to deal with biosequences containing gaps and FASTA introduces something similar to gaps in the process of joining adjacent regions, BLAST deals exclusively with uncapped biosequences. However, what is lost by disregarding gapped sequences is compensated for by the very fast execution times due to the construction of a finite state machine to recognize all substrings of some fixed size (called w-meters) of the query biosequence that scores above a given threshold value. (Given the presence of mutations, the authors estimate that each residue in the query biosequence will typically contribute 50 w-mers to the finite state machine). Hits generated by the scan are then extended until the score for the extended match falls below better scoring shorter matches. So in bioinformatics BLAST for Basic Local Alignment Search Tool is an algorithm for comparing primary biological sequence information, such as the amino-acid sequences of different proteins or the nucleotides of DNA sequences. A BLAST search enables a researcher to compare a query sequence with a library or database of sequences and identify library sequences that resemble the query sequence above a certain threshold. BLAST is one of the most widely used bioinformatics programs for sequence searching. It addresses a fundamental problem in bioinformatics research. The heuristic algorithm it uses is much faster

than other approaches, such as calculating an optimal alignment. This emphasis on speed is vital to making the algorithm practical on the huge genome databases currently available, although subsequent algorithms can be even faster, BLAST is also often used as part of other algorithms that require approximate sequence matching. BLAST can be used for several purposes. These include identifying species, locating domains, establishing phylogeny and comparison. With the use of BLAST, you can possibly correctly identify a species or find homologous species. This can be useful, for example, when you are working with a DNA sequence from an unknown species. When working with a protein sequence you can input it into BLAST, to locate known domains within the sequence of interest. Using the results received through BLAST you can create a phylogenetic tree using the BLAST web page. Phylogenies based on BLAST alone are less reliable than other purpose-built computational phylogenetic methods, so should only be relied upon for "first pass" phylogenetic analyses. In the case of comparison when working with genes, BLAST can locate common genes in two related species and can be used to map annotations from one organism to another. Before BLAST, FASTA was developed by (Lipman and Pearson, 1985). But BLAST is more time-efficient than FASTA by searching only for the more significant patterns in the sequences, yet with comparative sensitivity (Wise, 1995).

Longest Common Sequence (LCS)

The Longest Common Subsequence (LCS) problem is the problem of finding the longest subsequence common to all sequences in a set of sequences (often just two sequences). For example in a typical *LCS* Problem Statement: Given two sequences, find the length of longest subsequence present in both of them. A subsequence is a sequence that appears in the same relative order, but not necessarily contiguous. For example, “abc”, “abg”, “bdf”, “aeg”, ‘”acefg”, etc are subsequences of “abcdefg”. More explanation is given in the below-shown examples: LCS for input Sequences “ABCDGH” and “AEDFHR” is “ADH” of length3. LCS for input Sequences “AGGTAB” and “GXTXAYB” is “GTAB” of length 4, the naive solution for this problem is to generate all subsequences of both given sequences and find the longest matching subsequence. This solution is exponential in term of time complexity. Let us see how this problem possesses both important properties of a Dynamic Programming (DP) Problem, for the general case of an arbitrary number of input sequences, the problem is NP-hard (Maier, 1978) when the number of sequences is constant, the problem is solvable in polynomial time by dynamic programming for the case of two sequences of n and m elements, the running time of the dynamic programming approach is O (n×m). There exist methods with lower complexity (Bergroth *et al*., 2000) which often depend on the length of the LCS, the size of the alphabet, or both. Notice that the LCS is not necessarily unique; for example, the LCS of "ABC" and "ACB" is both "AB" and "AC". Indeed, the LCS problem is often defined to be finding all common subsequences of a maximum length. This problem inherently has higher complexity, as the number of such subsequences is exponential in the worst case, (Greenberg, 2003) even for only two input strings.

Dynamic Time Warping (DTW)

DTW algorithm has earned its popularity by being extremely efficient as the time-series similarity measure which minimizes the effects of shifting and distortion in time by allowing “elastic” transformation of time series in order to detect similar shapes with different phases. Given two time series $X = (x_1, x_2, ...x_N)$, $N \in N$ and $Y = (y_1, y_2, ...y_M)$, $M \in N$ represented by the sequences of values (or curves represented by the sequences of vertices) DTW yields optimal solution in the O (MN) time which could be improved further through different techniques such as multi-scaling (Müller *et al*., 2006; Satrya *et al*., 2015). The only restriction placed on the data sequences is that they should be sampled at equidistant points in time (this problem can be resolved by re-sampling) (Senin, 2008). The below figure is taken from (Tobiyama *et al*., 2016) explains more in the concept of DTW.

Figure 6 on the left shows the two-time series which are similar but out of phase and has produced a large Euclidean distance. To align the sequences a warping matrix has been constructed and search for the optimal warping path (red/solid squares) as shown on the right figure with the Sakoe-Chiba Band with width R is used to constrain the warping path. This supports the idea that Dynamic time warping is an algorithm for measuring the similarity of two-time series (Tobiyama *et al*., 2016).

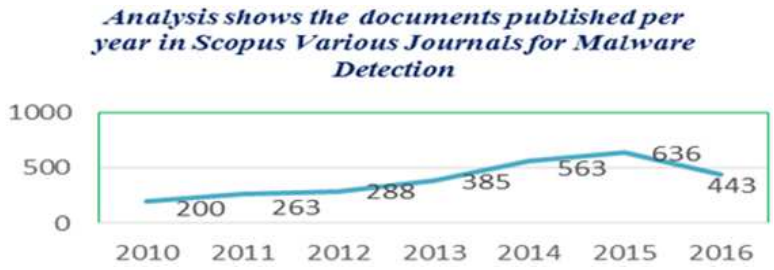

Fig. 5: Statistic, published research papers “scopus” (St'astna and Tomasek, 2015)

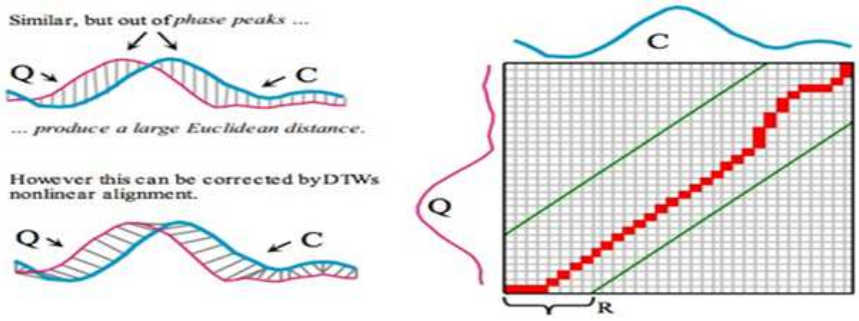

Fig. 6: Aligning using DTW concept (Tobiyama *et al*., 2016)

Discussion

Research Issues for MDS-Based On Call Graph and Data Mining

The graph is such a powerful tool for modelling structured objects and applications. A graph is a way of representing the systems because they offer a method to express models by the image. There are many definitions for graphs in literature, any definition considered applications that depend on it. It proves that the definition given below is flexible enough for a wide range of tasks (Riesen and Bunke, 2010). A graph G consists of a collection of two types of elements, namely vertices and edges, vertices are connected together by the edges. A vertex is simply drawn as a node. The vertex set of G is usually denoted by V (G), or V. |V (G)| is the order of a graph and it represents the number of the vertices. E (G), or E is the edge set of G, An edge (a set of two elements) with two endpoints "a" and "b" is represented by ab (without any symbol in between). An edge is plotted as a line that connects two vertices, called an endpoint. The main issues of using API call graph for detecting malware may be listed as:

- Input pre-processing
- API call graph construction
- API call graph optimization
- API call graph matching and similarity

API call graph techniques follow two main steps, namely, the transformation of malware samples into an API call graph using API call graph construction algorithm and matching the constructed graph against existing malware call graph samples using graph matching algorithm. A major issue facing malware API call graph construction algorithms is building a precise call graph from information collected about malware samples. On the other hand call, graph matching is an NP-complete problem and is slow because of computational complexity. Although the research was done has shown good Experimental results on 514 malware samples demonstrate that the proposed system has 98% accuracy and 0 false positive rates (Barry *et al*., 2015) but still it has computational time complexity which also results in wastage of large memory space. When it comes to applying data mining techniques for malware detection, a feature set has to be first generated. These features include hexadecimal byte sequences, instruction sequences, API/system call sequences etc. Usually, the number of features extracted from the files is very high. In order to select the best features here several techniques from the text, classification has been employed. Some other features include printable strings extracted from the files and some operating system dependent features such as DLL information. Some of these methods might use activity monitoring as a data collection method but data mining remains the principal detection method overall data mining needs all of the above scenarios of feature set generation and extraction which in almost the cases a difficult task to achieve unless very high level of detection methods/models been used (Elhadi, 2014). With all the above drawbacks been observed when using representation technique for malware based on various techniques We recommend and propose the Signature (String-based) and behavior representation technique which will solve all of the above issues and with high detection rate using very less memory space with the specialty in pinpointing the actual crucial zone of the malware sample been detected as malware and hence identifying it from the rest of the code which maybe a Benin and not actually malware.

API with Machine Learning Algorithms

Some researchers have used certain machine learning algorithms with API calls for malware detection such as: Instance Based Learner (IBk), decision tree (J48), Naıve Bayes (NB), inductive rule learner (RIPPER) and Support Vector Machine (SMO) machine learning classification algorithms (Ahmed *et al*., 2009).

Conclusion

In this study, we briefly surveyed the different types of malware and malware detection system as we have reviewed certain malware detection techniques such as

techniques based on machine learning, data mining, call graph and string representation. We have also pointed out the various methods that are used in malware analysis whether been static, dynamic or hybrid. We have discussed the use of some dynamic programming-based tools that could be used in the representation of the malware sampled gathered. We also gave a detailed discussion which reflects our findings with respect to the various malware detection representation techniques and corresponding methods that they have been used in producing effective malware detection systems as we have provided a proposal for string representation technique which will give high detection rate with the specialty of pinpointing the crucial or risky zone where the actual malware exists within the whole malware file. The objective of the survey is to provide a procedure, which could be suitable for further studies to develop malware detection techniques.

Author's Contributions

Gamal Abdel Nassir Mohamed: Has given a significant contribution in the preparation of this article. Produced the initial draft of the article, develop and carry out this manuscript.

Norafida Bte Ithnin: Is the supervisor, who oversaw the overall research article, Reviewed and finalised the draft of the article before it is being submitted.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Ahmed, F., H. Hameed, M.Z. Shafiq and M. Farooq, 2009. Using spatio-temporal information in API calls with machine learning algorithms for malware detection. Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence, Nov. 09-13, IEEE Xplore Press, Chicago, pp: 55-62. DOI: 10.1145/1654988.1655003

Al Daoud, E., I.H. Jebril and B. Zaqaibeh, 2008. Computer virus strategies and detection methods. Int. J. Open Problems Compt. Math, 1: 12-20.

Altschul, S.F., W. Gish, W. Miller, E.W. Myers and D.J. Lipman, 1990. Basic local alignment search tool. J. Molecular Biol., 215: 403-410.

Anderson, B., D. Quist, J. Neil, C. Storlie and T. Lane, 2011. Graph-based malware detection using dynamic analysis. J. Comput. Virol., 7: 247-258.

Aycock, J., 2006. Computer Viruses and Malware (Advances in Information Security). 1st Edn., Springer-Verlag, New York.

Banko, M. and E. Brill, 2001. Scaling to very very large corpora for natural language disambiguation. Proceedings of the 39th Annual Meeting on Association for Computational Linguistics, Jul. 06-11, IEEE Xplore Press, Toulouse, pp: 26-33. DOI: 10.3115/1073012.1073017

Barry, B., E. Elhadi, M.A. Maarof and A.A.E. Elhadi, 2015. Enhancing the detection of metamorphic malware using call graphs.

Bayer, U., E. Kirda and C. Kruegel, 2010. Improving the efficiency of dynamic malware analysis. Proceedings of the 2010 ACM Symposium on Applied Computing, Mar. 22-26, IEEE Xplore Press, Sierre, Switzerland, pp: 1871-1878. DOI: 10.1145/1774088.1774484

Bayer, U., A. Moser, C. Kruegel and E. Kirda, 2006. Dynamic analysis of malicious code. J. Comput. Virol., 2: 67-77.

Bergroth, L., H. Hakonen and T. Raita, 2000. A survey of longest common subsequence algorithms. Proceedings of the International Symposium on String Processing and Information Retrieval, Sept. 29-29, IEEE Xplore Press, A Curuna, Spain, pp: 39-39. DOI: 10.1109/SPIRE.2000.878178

Christodorescu, M. and S. Jha, 2004. Testing malware detectors. Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis, Jul. 11-14, IEEE Xplore Press, Boston, Massachusetts, pp: 34-44. DOI: 0.1145/1007512.1007518

Christodorescu, M., S. Jha and C. Kruegel, 2008. Mining specifications of malicious behavior. Proceedings of the Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Sept. 03-07, IEEE Xplore Press, pp: 5-14. DOI: 10.1145/1287624.1287628

Campagni, R., M. Donatella, R. Sprugnoli and M.C. Verri, 2015. Data mining models for student careers. Expert Syst. Applic., 42: 5508-5521.

Damodaran, A., F. Di Troia, C.A. Visaggio, T.H. Austin and M. Stamp, 2015. A comparison of static, dynamic and hybrid analysis for malware detection. J. Comput. Virol. Hack. Techniques, 13: 1-12. DOI: 10.1007/s11416-015-0261-z

Devesa, J., I. Santos, X. Cantero, Y.K. Penya and P.G. Bringas, 2010. automatic behaviour-based analysis and classification system for malware detection. Proceedings of the 12th International Conference on Enterprise Information Systems, Jun. 8-12, Madeira, Portugal.

Egele, M., T. Scholte, E. Kirda and C. Kruegel, 2012. A survey on automated dynamic malware-analysis techniques and tools. ACM Comput. Surveys, 44: 6-6. DOI: 10.1145/2089125.2089126

Elhadi, A.A.E., 2014. Enhanced application programming interface call graph architecture for malware detection. PhD Thesis, University of Technology, Malaysia.

Elhadi, A. A.E., M.A. Maarof and A.H. Osman, 2012. Malware detection based on hybrid signature behaviour application programming interface call graph. Am. J. Applied Sci., 9: 283-283.

Elhadi, E., M.A. Maarof and B. Barry, 2015. Improving the detection of malware behaviour using simplified data dependent api call graph.

Ellis, D.R., J.G. Aiken, K.S. Attwood and S.D. Tenaglia, 2004. A behavioral approach to worm detection. Proceedings of the 2004 ACM Workshop on Rapid Malcode, Oct. 29-29, IEEE Xplore Press, Washington DC, pp: 43-53. DOI: 10.1145/1029618.1029625

Fredrikson, M., S. Jha, M. Christodorescu, R. Sailer and X. Yan, 2010. Synthesizing near-optimal malware specifications from suspicious behaviors. Proceedings of the IEEE Symposium on Security and Privacy, May 16-19, IEEE Xplore Press, Berkeley/Oakland. DOI: 10.1109/SP.2010.11

Gandotra, E., D. Bansal and S. Sofat, 2014. Malware analysis and classification: A survey. J. Inform. Security.

Ghosh, S., S. Biswas, D. Sarkar and P.P Sarkar, 2014. A novel Neuro-fuzzy classification technique for data mining. Egypt. Informat. J., 15: 129-147.

Greenberg, R.I., 2003. Bounds on the number of longest common subsequences.

Griffin, K., S. Schneider, X. Hu and T.C. Chiueh, 2009. Automatic generation of string signatures for malware detection. Proceedings of the International Workshop on Recent Advances in Intrusion Detection, (AID' 09), pp: 101-120. DOI: 10.1007/978-3-642-04342-0_6

Han, K.S., I.K. Kim and E.G. Im, 2012. Detection methods for malware variant using api call related graphs. Proceedings of the International Conference on IT Convergence and Security, (ICS' 12), pp: 607-611. DOI: 10.1007/978-94-007-2911-7_59

Han, L., M. Qian, X. Xu, C. Fu and H. Kwisaba, 2014. Malicious code detection model based on behavior association. Tsinghua Sci. Technol., 19: 508-515.

Hu, X., T.C. Chiueh and K.G. Shin, 2009. Large-scale malware indexing using function-call graphs. Proceedings of the 16th ACM Conference on Computer and Communications Security, Nov. 9-13, Chicago, Illinois, pp: 611-620. DOI: 10.1145/1653662.1653736

Idika, N. and A.P. Mathur, 2007. A survey of malware detection techniques. Purdue University.

Kephart, J.O., G.B. Sorkin, W.C. Arnold, D.M. Chess and G.J. Tesauro *et al*., 1995. Biologically inspired defenses against computer viruses. IJCAI.

Kim, K. and B.R. Moon, 2010. Malware detection based on dependency graph using hybrid genetic algorithm. Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, Jul. 07-11, IEEE Xplore Press, Portland, pp: 1211-1218. DOI: 10.1145/1830483.1830703

Kolbitsch, C., P.M. Comparetti, C. Kruegel, E. Kirda and X.Y. Zhou *et al*., 2009. Effective and efficient malware detection at the end host. Proceedings of the 18th Conference on USENIX Security Symposium, Aug. 10-14, IEEE Xplore Press, Montreal, pp: 351-366.

Kolter, J.Z. and M.A. Maloof, 2006. Learning to detect and classify malicious executables in the wild. J. Machine Learn. Res., 7: 2721-2744.

Kostakis, O., J. Kinable, H. Mahmoudi and K. Mustonen, 2011. Improved call graph comparison using simulated annealing. Proceedings of the ACM Symposium on Applied Computing, Mar. 21-24, IEEE Xplore Press, TaiChung, Taiwan, pp: 1516-1523. DOI: 10.1145/1982185.1982509

Kreibich, C. and J. Crowcroft, 2004. Honeycomb: Creating intrusion detection signatures using honeypots. ACM SIGCOMM Comput. Commun. Rev., 34: 51-56.

Kumar, N., H. Shah and R. Shyamasundar, 2010. Can we certify systems for freedom from malware. Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, May 2-8, IEEE Xplore Press, Cape Town. DOI: 10.1145/1810295.1810323

Landwehr, C.E., A.R. Bull, J.P. McDermott and W.S. Choi, 1994. A taxonomy of computer program security flaws. ACM Comput. Surveys, 26: 211-254.

Lee, J., K. Jeong and H. Lee, 2010a. Detecting metamorphic malwares using code graphs. Proceedings of the ACM Symposium on Applied Computing, Mar. 22-26, IEEE Xplore Press, Sierre, Switzerland, pp: 1970-1977. DOI: 10.1145/1774088.1774505

Lee, V.E., N. Ruan, R. Jin and C. Aggarwal, 2010b. A survey of algorithms for dense subgraph discovery. Manag. Min. Graph Data, 40: 303-336.

Li, W.J., S. Stolfo, A. Stavrou, E. Androulaki and A.D. Keromytis, 2007. A study of malcode-bearing documents. Proceedings of the International Conference on Detection of Intrusions and Malware and Vulnerability Assessment, (MVA' 07).

Lin, D. and M. Stamp, 2011. Hunting for undetectable metamorphic viruses. J. Comput. Virol., 7: 201-214.

Lipman, D.J. and W.R. Pearson, 1985. Rapid and sensitive protein similarity searches. Science, 227: 1435-1441.

Maier, D., 1978. The complexity of some problems on subsequences and supersequences. J. ACM, 25: 322-336.

Mavrommatis, N.P.P. and M.A.R.F. Monrose, 2008. All your iframes point to us. Proceedings of the 17th Conference on Security Symposium, Jul 28-Aug. 01, San Jose, pp: 1-15

McGraw, G. and G. Morrisett, 2000. Attacking malicious code: A report to the Infosec Research Council. IEEE Software, 17: 33-33.

Mehdi, B., F. Ahmed, S.A. Khayyam and M. Farooq 2010. Towards a theory of generalizing system call representation for in-execution malware detection. Proceedings of the IEEE International Conference on Communications, May 23-27, IEEE Xplore Press. DOI: 10.1109/ICC.2010.5501969

Monitor-Spy, A.P.I., 2012. Display API calls made by Win32 applications.

Schultz, M.G., E. Eskin, E. Zadok and S. J. Stolfo, 2001. Data mining methods for detection of new malicious executables. Proceedings of the IEEE Symposium on Security and Privacy, (SSP' 01), Los Alamitos, CA, pp: 38-49.

Moser, A., C. Kruegel and E. Kirda, 2007. Limits of static analysis for malware detection. Proceedings of the Computer Security Applications Conference, (SAC' 07).

Moskovitch, R. and Y. Shahar, 2015. Classification-driven temporal discretization of multivariate time series. Data Min. Knowl. Discovery, 29: 871-913.

Müller, M., H. Mattes and F. Kurth, 2006. An efficient multiscale approach to audio synchronization.

Nair, V.P., H. Jain, Y.K. Golecha, M.S. Gaur and V. Laxmi, 2010. MEDUSA: MEtamorphic malware dynamic analysis using signature from API. Proceedings of the 3rd International Conference on Security of Information and Networks, (SIN' 10).

Norouzi, M., A. Souri and M.S. Zamini, 2016. A data mining classification approach for behavioral malware detection. J. Comput. Netwo. Commun.

Nissim, N., M Robert., L. Rokach and E. Yuval, 2012. Detecting unknown computer worm activity via support vector machines and active learning. Patt. Anal. Applic. 15: 459-475.

Park, Y., D. Reeves, V. Mulukutla and B. Sundaravel, 2010. Fast malware classification by automated behavioral graph matching. Proceedings of the 6th Annual Workshop on Cyber Security and Information Intelligence Research, (IIR' 10).

Pathak, P. and Y.M. Nanded, 2016. A dangerous trend of cybercrime: Ransomware growing challenge. Int. J. Adv. Res. Comput. Eng. Technol., 5: 371-373.

Patil, D.R. and J. Patil, 2015. Survey on malicious web pages detection techniques. Int. J. e-Service, Sci. Technol., 8: 195-206.

Perdisci, R., A. Lanzi and W. Lee, 2008. McBoost: Boosting scalability in malware collection and analysis using statistical classification of executables. Proceedings of the Computer Security Applications Conference, (SAC' 08).

Piyanuntcharatsr, S.S.W., S. Adulkasem and C. Chantrapornchai, 2015. On the comparison of malware detection methods using data mining with two feature sets. Int. J. Security Applic., 9: 293-318.

Rad, B.B., M. Masrom and S. Ibrahim, 2012. Camouflage in malware: From encryption to metamorphism. Int. J. Comput. Sci. Netw. Security, 12: 74-83.

Rahman, R.M. and F.R.M. Hasan, 2001. Using and comparing different decision tree classification techniques for mining ICDDR, B Hospital Surveillance data. Expert Syst. Applic., 38: 11421-11436.

Rey, A.M., 2015. Mathematical modeling of the propagation of malware: A review. Security Commun. Netw., 8: 2561-2579.

Rhee, J., R. Riley, D. Xu and X. Jiang, 2010. Kernel malware analysis with un-tampered and temporal views of dynamic kernel memory. Proceedings of the International Workshop on Recent Advances in Intrusion Detection, (AID' 10).

Rhee, J., R. Riley, D. Xu and X. Jiang, 2011. LiveDM: Kernel malware analysis with un-tampered and temporal views of dynamic kernel memory. Proceedings of the 12th Annual Information Security Symposium, (ISS' 11).

Rieck, K., P. Trinius, C. Willems and T. Holz, 2011. Automatic analysis of malware behavior using machine learning. J. Comput. Security, 19: 639-668.

Riesen, K. and H. Bunke, 2010. Graph classification and clustering based on vector space embedding. World Scientific Publishing Co., Inc.

Riesen, K., X. Jiang and H. Bunke, 2010. Exact and inexact graph matching: Methodology and applications managing and mining graph data.

Santos, I., F. Brezo, X. Ugarte-Pedrero and P.G. Bringas, 2013. Opcode sequences as representation of executables for data-mining-based unknown malware detection. Inform. Sci., 231: 64-82. DOI: 10.1016/j.ins.2011.08.020

Satrya, G.B., N.D. Cahyani and R.F. Andreta, 2015. The detection of 8 type malware botnet using Hybrid Malware analysis in executable file windows operating systems. Proceedings of the 17th International Conference on Electronic Commerce, Aug. 03-05, IEEE Xplore Press, Seoul. DOI: 10.1145/2781562.2781567

Senin, P., 2008. Dynamic time warping algorithm review. Proceedings of the Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA, pp: 1-23.

Shabtai, A., R. Moskovitch, Y. Elovici and C. Glezer, 2009. Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. Informat. Security Tech. Rep., 14: 16-29.

Siddiqui, M., M.C. Wang and J. Lee, 2008. A survey of data mining techniques for malware detection using file features. Proceedings of the 46th Conference Annual Southeast Regional, Mar. 28-29, IEEE Xplore Press, Auburn, pp: 509-510. DOI: 10.1145/1593105.1593239

Skormin, V.A., D.H. Summerville and J.S. Moronski, 2003. Detecting malicious codes by the presence of their "gene of self-replication. Proceedings of the International Workshop on Mathematical Methods, Models and Architectures for Computer Network Security, (CNS' 03), pp: 195-205. DOI: 10.1007/978-3-540-45215-7_16

St'astna, J. and M. Tomasek, 2015. Exploring malware behaviour for improvement of malware signatures. Proceedings of the IEEE 13th International Scientific Conference on Informatics, Nov. 18-20, IEEE Xplore Press, Poprad, Slovakia, pp: 275-280. DOI: 10.1109/Informatics.2015.7377846

Stopel, D., Z. Boger, R. Moskovitch, Y. Shahar and Y. Elovici, 2006. Application of artificial neural networks techniques to computer worm detection. Proceedings of the International Joint Conference on Neural Networks, Jul. 16-21, IEEE Xplore Press, Vancouver. DOI: 10.1109/IJCNN.2006.247059

Summerville, D., V. Skormin, A. Volynkin and J. Moronski, 2005. Prevention of information attacks by run-time detection of self-replication in computer codes. Proceedings of the International Workshop on Mathematical Methods, Models and Architectures for Computer Network Security, (CNS' 05), pp: 54-75. DOI: 10.1007/11560326_5.

Tabish, S.M., M.Z. Shafiq and M. Farooq, 2009. Malware detection using statistical analysis of byte-level file content. Proceedings of the ACM SIGKDD Workshop on Cyber Security and Intelligence Informatics, Jun. 28-28, IEEE Xplore Press, Paris, France, pp: 23-31. DOI: 10.1145/1599272.1599278

Tobiyama, S., Y. Yamaguchi, H. Shimada, T. Ikuse and T. Yagi, 2016. Malware detection with deep neural network using process behavior. Proceedings of the IEEE 40th Annual Computer Software and Applications Conference, Jun. 10-14, IEEE Xplore Press, Atlanta. DOI: 10.1109/COMPSAC.2016.151

Verma, A., M. Rao, A. Gupta, W. Jeberson and V. Singh, 2013. A literature review on malware and its analysis. Int. J. Current Res. Rev., 5: 71-71.

Vinod, P., R. Jaipur, V. Laxmi and M. Gaur, 2009. Survey on malware detection methods. Proceedings of the 3rd Hackers' Workshop on Computer and Internet Security, (CIS' 09).

Walenstein, A., R. Mathur, M.R. Chouchane and A. Lakhotia, 2007. The design space of metamorphic malware. Proceedings of the 2nd International Conference on i-Warfare and Security, (CWS' 07).

Wise, M.J., 1995. Neweyes: A system for comparing biological sequences using the running Karp-Rabin Greedy String-Tiling algorithm. Proc. Int. Conf. Intell. Syst. Mol. Biol., 3: 393-401.

Yang, H. and P.C. Yi-Ping, 2015. Data mining in lung cancer pathologic staging diagnosis: Correlation between clinical and pathology information. Expert Syst. Applic., 42: 6168-6176.

Ye, Y., T. Li, Q. Jiang and Y. Wang, 2010. CIMDS: Adapting post processing techniques of associative classification for malware detection. IEEE Trans. Syst. Man Cybernet., 40: 298-307.

Ye, Y., D. Wang, T. Li, D. Ye and Q. Jiang, 2008. An intelligent PE-malware detection system based on association mining. J. Comput. Virol., 4: 323-334.

You, I. and K. Yim, 2010. Malware obfuscation techniques: A brief survey.

Zhang, Y. and F. Xia, 2012. A self-relocation based method for malware detection. Applied Mechanics Mater.

Zolkipli, M.F. and A. Jantan, 2010. A framework for malware detection using combination technique and signature generation. Proceedings of the 2nd International Conference on Computer Research and Development, May 7-10, IEEE Xplore Press, Kuala Lumpur. DOI: 10.1109/ICCRD.2010.25