STUDY AND ANALYSIS OF VARIOUS APPROACHES FOR MALWARE DETECTION AND IDENTIFICATION

1G. HAMSA, 2DEEPTI VIDYARTHI
Defense Institute of Advance Technology, Pune, India
hamsa_gopalaswamy@yahoo.com, deepti.vidyarthi@gmail.com

Abstract— There is a continuous ever-growing arm race between the security defenders and the malware writers in introducing new techniques of malware detection and its evasion. Any new technique introduced and adopted by the anti-malware community is immediately responded by an effective evasion technique by the malware community. Understanding the pros and cons of various approaches of malware detection and identification is very essential to win the arm race. Each approach suffers specific evasion technique but at the same time has its own benefits too. A technique which uses a combination of approaches that complement each other by bringing out the detection capability of an approach and nullifying the evasion techniques of another will best suit today’s need of security. The paper studies and analyses various techniques used for malware detection and identification. It categorizes the malware detection techniques into four quadrants based on the basic approach they adopt. It reflects upon each of these categories and their effectiveness so that a combined, more effective approach of malware detection might be found out.

Index Terms— Dynamic analysis, Malware analysis, Malware detection, Static analysis.

I. INTRODUCTION

Malware or malicious software can be defined as any program intended to cause disruption or gain access to unauthorized information and resources and thus affect the secrecy, the integrity and the functionality of the system. The term malware is a generic term used to collectively refer to all threats to the computer system. Malware analysis has been and is likely to continue to be the most challenging and unsolved problem of the present era. There is an arm race continuously taking place between the malware writers and the security defenders. Any new technique implemented by the security defenders is in no time evaded by the malware writers and the race goes on and on. A wide range of malware exists in the present era with each targeting at a specific result and exploiting a specific vulnerability of the system. Designing a single automated antimalware system to collectively identify or detect this wide range of malware is out of practicality. The malware analysis tools available today are specific either to a class of malware or aim to decipher the presence of malware with a group of similar characteristics. Though the present antivirus industries succeed to a large extent in detecting malicious files and reverting their effects, malware evolve with an enormous speed and hard-to-detect characteristics. “The Vulnerability and Threat Categories chart shows a significant increase in threat totals— in 2012, threats increased 19.8 percent over 2011. This sharp increase in threats is placing a serious strain on the ability of organizations to keep vulnerability management systems updated and patched —” says 2013 Cisco Annual Security Report [19]. This paper categorizes the techniques used by the security defenders into four categories of approaches. Each approach is effective in detecting a particular kind of malware and at the same time falls prey to certain evasion techniques. Each approach is capable of bringing out the malicious nature of the programs in a certain way. On the other side of the coin, each approach also has a few shortcomings which compromise the quality of the approach. A technique that adopts a combination of these approaches can be very effective and successful in detecting threats of the present era.

Organization of the paper:
The remainder of this paper is organized into five main sections. (Section II, III, IV, V and VI). Section II states the clear categorization of the various malware analysis techniques into four distinct approaches. Section III describes the approaches based on the static analysis. It also involves the common techniques used to evade static analysis. Section IV discusses the approaches based on dynamic analysis and includes the evasion techniques used against dynamic analysis. Section V bullets the different important features under each approach. Section VI gives a brief conclusion.

II. APPROACHES OF MALWARE ANALYSIS

The available collection of the techniques for malware analysis inclusive of those adopted by the industries and those that are not can be categorized into four approaches. They fall in one of the four quadrants of the matrix given in Table 1.
III. APPROACHES BASED ON STATIC ANALYSIS

Static analysis refers to the analysis of the malware from its binaries without an actual run. The analysis may be based directly on the binary byte sequence or on the disassembled binary.

A. Approach 1: Static Signature Generation.

The static analysis generally aims to generate signature specific to a class of malware.

Table 1: Four different approaches of malware analysis

<table>
<thead>
<tr>
<th>Signature based</th>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quad I Approach 1</td>
<td>Quad III Approach 3</td>
<td></td>
</tr>
<tr>
<td>Quad II Approach 2</td>
<td>Quad IV Approach 4</td>
<td></td>
</tr>
</tbody>
</table>

The signature generated is based on the hash values of the byte sequence found in the malicious binaries. Though the signature database is to be updated very frequently due to polymorphic and metamorphic malwares, signature comparison is largely used in today’s anti malware industries. This intensive use of signature comparison is mainly due to the very low rate of false positives it offers. The drawback with the static signature generation approach is that the number of unique signatures is very large. The malware writers increase the number of unique signature by releasing various polymorphic and metamorphic stains of the same malware. The speed, quality and the efficiency of the updates of this signature database determines the success stories of the AntiVirus industries. The average update rate per day in a week for some of the successful antivirus industries is given in fig.1.[18]. Moreover signature based techniques helps in no means to detect unknown threats. The time consumed for signature generation for zero day malware should be as low as possible to minimize the damage. Yangseo Choi et al [16] suggest a method to minimize this time. The technique is based on identifying the optimal position in the binary in which the unique signature can be identified. Once this position is identified a n-byte signature starting from this position can be generated with low probability of false positives. The method relies on entropy and variance based approach to locate the most optimal position. The false positive rate with 32 byte length signature is as low as 0.032% for variance based and 1.17% for entropy based approach. The storage of hash signatures makes the volume of the database huge. The Hancock Signature generation technique by K.Griffing et al [9] of Symantec Research Laboratories claims a way to reduce the volume by replacing hash signatures by “string signatures” which corresponds to “short contiguous byte sequence from a malware binary”. A way to ensure that this approach does not increase the rate of false positive is also taken care by the technique. It verifies that the string generated is not a part of library functions. A goodware is also expected to use the library function. So such verification increases the probability of low false positive rates. An extended Hancock is also suggested in the paper which generates string signatures that consist of multiple disjoint byte sequence rather than a single continuous byte sequence. Hence this technique can be very helpful to avoid storage of huge volume of malware signature without compromising the low false positive rates.

![Fig 1: The average weekly update rates for five antivirus companies](image)

B. Approach 2: Static Behavior Based

The behavior of the program code can be analyzed statically without a run using control flow analysis, data flow analysis and various other techniques. Control flow is a graphical representation of software by use of nodes that represent basic blocks and directed edges to represent paths from one block to another. Data flow analysis is used to collect run-time (dynamic) information about data in software while it is in a static state (Wögerer, 2005). If the control and data flow of a program in comprehended clearly, the behavior of the program can be deduced from its code in the static state. The analysis based on static behavior is not as widely used as the signature generation based static analysis. Though many techniques based on this approach is available in the literature, this approach suffers from the drawback of high false positive rates. If an assured way to minimize false positive rate is incorporated, this approach can also flourish in the industries. A novel approach to detect new strands of malware under the same family by observing the static behavior of the malware (code behavior) is proposed by V. Sathyanarayan et al [17]. The technique in the paper suggest disassembling of the binary using IDAPro, identifying the sequence of critical API calls specific to a family and using it as a signature for the entire family. This creates a unique signature for the entire family independent of release
of various strands. As the technique considers only critical API calls, the effect of obfuscation on the signature generation process is also very low. The SIGNATURE GENERATE algorithm considers the profile of a particular class of malware as a whole and generates signatures based on the critical API calls. So any new variant under the class is very likely to be detected by this algorithm. As the signature is generated based on the behavioral structure of the disassembled code rather than the byte sequence located in the binary, this technique can be categorized as belonging to Approach 2 rather than Approach 1. No way to ensure low false positive rates is discussed in the paper.

JIN-CHENG DENG et al [10] describes an automatic engine to detect the malicious behavior of the binary. The code slicing is carried out and the behavior or the flow of the code is recorded. If this code pattern matches with a prerecorded pattern for malicious code, an alarm or a notification is generated. No specific way to decrease the number of false positives is discussed in this paper too. Techniques based on static taint analysis are also useful to observe the behavior of the code in a static mode. Static taint analysis is typically performed on Control Flow Graphs. Any variable whose flow is monitored is called a taint. Any variable depended or derived from the taint is also tainted. The tainted tool proceeds variable by variable until it has a complete list of all variables which are potentially tainted. If any of these variables is used to execute dangerous commands, the taint checker warns the program that it is using a potentially dangerous tainted variable and a malicious behavior is suspected. For e.g. If an Input variable (all inputs from the users are tainted) determines a jump address, a malicious behavior is suspected. Static taint analysis should look at multiple paths and must either over or under approximate taint at confluence of paths. Automatic static analyzer designed by Jia Zhang et al [12] uses call graphs to analyze the behavior of the program code. Any packed code is unpacked before being fed into the pattern analyzer. The call graph gives a pattern in which the program calls the API Functions and the system calls. This pattern is later compared to the observed pattern in a variety of malware. A match suspects malicious behavior of the code. This technique too does not specify any method to reduce the rate of false positives. If the pattern represents the use of any library function, a goodware too can use it. An effective way to reduce false positives should be identified before this technique is adopted.

C. Evasion of Static Analysis Approaches

The malware writers evade the method of static analysis (both signature based and behavior based) by various obfuscation techniques like insertion of dead code, register reassignment, opaque predicates, instruction substitution, code transposition and what not. The surface obfuscation techniques is concerned only in making the code difficult to read and does not change the basic semantic of the program. Changing the variable names to an arbitrary one from a meaningful one can be thought as surface obfuscation. Eg: The DotFusctor tool does surface obfuscation on .Net source codes. [4].The deep obfuscation technique attempts to change the entire structure of the code by changing its control flow or the data reference behavior. Thus the obfuscation techniques range from creating difficulties to the static analysis tools to entirely evade them from detecting the malicious behavior. The various deobfuscation techniques developed and tested during the past decade has only increased the arm war race and the malware writers come up with newer obfuscation techniques every time.

IV. APPROACHES BASED ON DYNAMIC ANALYSIS

The dynamic analysis is the effective way of detecting the effects of the malware on the system. Dynamic analysis is done by running the malware from one to many times mostly in a virtual environment. The exact change in the behavior of the computer system due to the malware at runtime can be analyzed by dynamic analysis.

D. Approach 3: Dynamic Signature Based

The specific behavior of the malware during runtime is identified. This behavior is then mapped to the presence of certain section of byte sequence present in the binary or code patterns seen in the disassemblies. These byte sequence or code patterns that are responsible for the malicious behavior are treated as a signature for the class of malware. This approach for signature generation can be adopted to thwart the ill effects of an exclusive static signature generation. Signatures are generated based on the dynamic behavior of the malware. Signatures are also generated based on the traced API Calls or system calls during execution of the binaries. This deals with the “behavior of the system” during runtime. The runtime behavior of the program is to be recorded and this behavior is mapped to the patterns found in the binary/disassembled binaries. These patterns are treated as signatures for further detection. Exclusive dynamic signature based approach is undefined.

E. Approach 4: Dynamic Behavior Based

Two different technique of dynamic behavioral analysis are largely followed which vary in their level of granularity and effectiveness to determine the exact consequence of the program run on the system. First technique is a program centric approach. The actions performed by the malware during its execution are monitored. Second technique is a system centric approach which records the system state before and
after the malware run. The two states are compared and the effect of the malware run on the system is deciphered.

i. Program Centric Approach

The system centric approach fails to observe any intermediate changes. For eg. If a file is created and deleted by the malware during its runtime, it will not reflect in the after-run state of the system. The first technique is more granular and therefore adopted by most of the dynamic analysis tools. A large number of automated tools namely Anubis, CWSandbox, Norman Sandbox are available which create a virtual environment for the malware to run and then analyze them. These tools largely use techniques like Windows API Calls sequence, OS System call sequence and Windows Native API Calls. Anubis [1] is aware of the fact that Windows assigns each running process its own page directory and uses this information to monitor only the required process. Anubis uses the physical address of the page directory of the currently running process which is always present in the CR3 CPU register. Anubis also monitors all processes that are created by the original process. This is possible by monitoring the APIs and the system calls that are responsible for creating new processes. Recently (2012) Anubis has come up with the Andrubis tool to analyze Android malware. CWSandbox [2] uses API function hooking for detecting the malicious behavior of the program. When Microsoft releases newer versions of the Windows, it does so by changing the implementation of the native API functions. The interface of the regular windows API function is kept without any change. The control flow of a malware sample can be observed if the access to the difference API function is gained. API function hooking is a technique of gaining this access by intercepting any calls to the functions. CWSandbox exploits this technique to analyze the behavior of the malware during its runtime. The hooking should be done in a smart way such that malware is totally unaware of it. Else the analysis environment will be detected by the malware.

Norman Sandbox [3] creates a tightly controlled virtual environment that includes the host computer, the associated LAN and even the internet connectivity. The main objective of the Norman Sandbox was to detect malware that spread through the net. This tool also employs the API hooking and parameter monitoring technique to detect malicious behavior. Norman Sandbox goes a step ahead and enables automatic invocation after shutdown and reboot.

V. SYSTEM CENTRIC APPROACH

All the above tools are based on program centric approach. Program centric techniques assess the behavior of the program by the way the program behaves during runtime. The behavior is largely based on sequence of system calls, sequence of API calls etc. To differentiate between the malware and the cleanware, the techniques records the call sequence for a number of benign software and for known malware. During detection, if the behavior pattern of the required program to be analyzed matches largely with benign software, the program is considered as benign. Else if it matches to a large extent to a known malware, it is coined as malicious. The reduction in the rate of false positives in program centric techniques depends largely on the number of software (cleanware/malware) analyzed earlier. When a newer behavior that is not found in the set of benign software analyzed earlier is encountered, it is generally coined as malicious. This is to avoid the most unwanted condition of false negative. Any new behavior pattern is suspected for malicious activities for the only sake that it is new. System centric approach comes into picture to overcome this shortcoming of program centric approach. The system centric approach observes and compares the effect of the program execution on the system and its resources. Andrea Lanzi et al [6] suggest a technique based on system centric approach. This technique observes the way in which the program interacts with the operating system and uses its resources (Memory, files etc). The approach derives an access activity model which models the way in which the operating system resources are accessed in an authenticated manner. Any software that violates this activity model is suspected for a malicious intention. Though system centric approach is not aware of intermediate action, effect of such actions on the outcome of the program is very little and negligible to a lot extent. For eg.: if a malware creates and deletes a file during its execution, the system centric approach will surely be unaware of such an activity. But the effect of file creation is nullified by file deletion and is of very little concern provided there is no ultimate effect on the entire system.

F. Evasion of Techniques based on Dynamic Analysis

i. Detection of Virtual/ Monitoring environment

All the above mentioned dynamic analysis tools work fine until the virtual environment is detected by the malware. Once the virtual environment is detected, the malware stops exhibiting its true functionality and remains dormant with respect to its malicious behaviour. For instance the malware may write into a file and read from it subsequently. If it is unable to read the original contents, it detects a virtual environment and remains dormant. The dynamic analysis tools are designed best to hide the presence of an analyser from the malware. Here too the arm race begins and malware writers react quickly to evade such techniques. Xu chen et al [8] has given a
consolidated view on the various features a virtual or monitoring environment should take care in order to prevent being detected. A virtual environment is safe from detection to the extent to which it minimizes the difference between its own characteristic and those of the real environment. Xu chen et al [8] measure the effectiveness of an environment based on the level of access required to detect it, the effective functioning of the environment when no evasion techniques is used, extent to which it mimics the real environment and the complexity of techniques used to detect it. The hardware detectable changes like the presence of device, device drivers should be greatly reduced. Malware look for the presence of tell-tale virtual drivers to understand that it is run in a virtual environment. For instance, The SoftICE kernel debugger uses virtual drivers with well known file names to communicate with the kernel. The monitoring environment should hide its presence from the malware by minimizing the changes due to execution environment. For e.g., the Windows API sets the flags IsDebuggerPresent() and CheckRemoteDebugger() when a program is run in a debugger. These flags can be intentionally and carefully masked by the system with the right tools. This makes it difficult for the malware being analyzed to detect the presence of a debugger. Poorly designed debuggers invoke CPU instruction bugs which can be used by the malware to detect the environment under which it is run. The environment detection results in malware not exhibiting its malicious behavior. A few changes in the real environment which make it appear as a monitoring environment will force the malware to hide its malicious functionality and minimize the threat to the real system. A few such techniques is also discussed by Xu chen et al [8]. The techniques are simple, direct fool proof techniques which when implemented fool the malware. These techniques are meant not for malware detection but for malware defense. Care should be taken that the real environment not loses its major functionalities due the changes incorporated.

ii. Dormant functionality based on external input, event or trigger.

Dormant functionality is not only exhibited on detection of a virtual environment. Certain malware do not show of their true malicious nature unless triggered by certain temporal or environment constraints. A single run of the malware can never detect this dormant functionality of the malware. Though multiple runs can detect them, the exact no of runs that “multiple” refers to is never known. The temporal or environment constraint can either be the reaching of a particular time/day in the calendar or may depend on the value of an input variable or anything else. These constraints are converted into a trigger condition variable. The condition variable is checked for a particular value which when true, triggers the malicious nature of the malware. The non-deterministic behavior of the malware due to dormant functionality it possess makes dynamic analysis of the malware infeasible. Moser et al [15] designed an automated tool based on the method of multiple execution paths to bring out the dormant malicious nature of a program. The tool suggests tainting of any external input to the program (may be a variable, a file, current system time or any external trigger) and following the flow of the tainted variable. If the tainted variable is used in a condition statement, the snapshot of the program is taken at the present point of execution say point a. The program execution is monitored with a known value of tainted variable. If no malicious behavior is observed with the given value input, the program is reverted to point a and continued with another input value. Thus a multiple path execution is followed for all possible values of the input. This technique may lead to infinite number of path based on the input value and the program executes on and on with no end. The use of this tool may also lead to Denial-Of -Service attack if the attackers intentionally use a huge number of conditional branches in his program. Moreover If the condition statement uses a one-way function (eg hash function) based on the input, the tool may stop functioning. The idea of multiple path exploration has led to further studies in this area and effects to overcome above said limitations are on. Trigger based condition never show up until the trigger is activated. The trigger expected may be reaching a system time or receiving a command from a command centre like a botmaster. The bots in the botnet are inactive till a trigger from the botmaster is received. Brumley et al [11] discusses a method to evade dormant functionality based on external triggers as in a botnet. When an external input is used in a conditional statement, the path taken by all possible value of the inputs is followed and the behavior of the path taken is recorded. This method also leads to path explosion as the range of values an external input can take is very large.

VI. OBSERVATION

Categorizing malware analysis into four different approaches aids in clearly comprehending a particular technique. The effect to nullify the evasion in the particular technique can be made accordingly. The specific characteristics of each approach are bulleted below.

Approach 1: Static Signature based.
1. This is the approach used extensively by antivirus industries
2. This approach is effective due to extremely low false positive rates
3. This is affected largely by code obfuscation. Effective deobfuscation techniques is required to be used for evading obfuscation.
4. Unique number of signatures is very large due to polymorphism and metamorphism. This effect can be overcome if a signature common to an entire family irrespective of frequently released variants is identified effectively.
5. Huge database for storage of hash signature increases the storage space. Volume of the database can be minimised by implementing string signatures
6. Care should be taken that any enhancement to declare the defects of polymorphism, metamorphism, huge volume of the database or obfuscation do not compromise with the low false positive rate

Approach 2: Static Behavior based
1. This approach has very less direct usage in the industry.
2. This approach suffers high false positive rates.
3. Clear understanding of the control flow and data flow can make this approach more effective. But this understanding too will in no ways ensure a way to avoid high false positive rates.

Approach 3: Dynamic signature based
1. Exclusive dynamic signature based techniques are not defined.
2. The behavior during the program execution is observed. The code pattern/byte sequence responsible for the behavior is identified and the signature based on the code pattern/byte sequence is generated.
3. As this approach highly depends on dynamic behavior analysis, it adopts all the pros and cons of Approach 4

Approach 4: Dynamic behavior based
1. This approach is used in industry to understand the runtime behaviour of a malware
2. This approach can further be differentiated into system centric and program centric.
3. The major number of dynamic analysis tools uses the program centric approach due to its higher level of granularity.
4. System centric approach is very effective to reduce false positives
5. Detection of a monitoring environment can result in making the malware to hide its true functionality
6. Hidden dormant functionality is difficult to be analysed
7. In operational environment, when a malware arrives to a system, it should be blocked without running it. So an exclusive dynamic behavior based approach cannot be used for active defense. A signature can be generated based on the byte sequence or code pattern responsible for the dynamic behaviour. This signature can be used to block the malware before its entry into the system.

A comprehensive technique with the combination of one or more of the above four approaches that will best suit the malware is required to be identified and applied accordingly. Static signature based approach is the one that is widely used in the present Anti-virus industries. As many deobfuscation techniques are in use and more techniques are sure to evolve, obfuscation is not as harmful as release of multiple variants. This approach is the best if new stains of malware which are self modifying are recognized easily. If faster generation of signatures for zero day malware is achieved to minimize damage, then maximum benefit from this approach can be gained.

Static signatures can determine whether the malware is packed or unpacked. Once it is determined that the malware is packed, a suitable unpacking algorithm can be used to unpack them. A subsequent dynamic analysis of the unpacked malware can reveal the malicious nature of the software. Static behavior based approach is highly affected by the high rate of false positives. If critical API calls / system calls are considered intensely rather than the API Calls / System calls that are common to both goodware and malware, the false positives can be avoided to a large extent. The part of the disassemblies responsible for the malicious behavior and the byte sequence corresponding to the code pattern can be identified. This can generate a static signature from the static behavior of the code. Thus a combination of approach 1 and approach 2 can be used to wisely avoid high false positive rates.

Another widely used approach is the dynamic behavior based approach. Dynamic behavior based approach is largely used to analyze the exact behavior of the malware during runtime. If tools like joebox which works in real environment is used to avoid dormant functionality, dynamic analysis bring out the activities in which the malware is involved during its execution. If the malware is residing in the memory (or say using system memory resources), actually doing nothing, dormant functionality is suspected. Code analysis of such a malware can be carried out to decipher its intention. Some malware like rootkits entirely hide its activity from the system administrator. They hide any change in the system state due to its presence. A system centric dynamic analysis can help very little to detect malware like rootkits. On the other hand, malware like rootkness create a lot of changes in the system due to its presence. They include themselves in normal executives and make abnormal changes in the system in the background. A system centric approach will be
very helpful in detecting them. For implementing active defense, the signature that corresponds either to the static or dynamic behavior of the malware is generated. But before generating a signature, the behavior of the malware is analyzed to understand the functionality of the malware as a whole.

CONCLUSION

Malware writing was initially used as a means to prove ones’ programming ability or as a hobby of freelancers. But today it has evolved as a means for illegal financial, anti-social and anti-national acts. A wise combination of malware analysis approaches is essential in reducing the number of malware threats that is increasing at an exponential rate.

REFERENCES

[14] Ronghua Tian, Rafiqul Islam and Lynn Batten, Steve Versteeg, Differentiating Malware and Cleanware Using Behavioural Analysis

★ ★ ★