Dynamic Analysis of Malicious Code and Response System

Ajay Katangur1, Vinay Chaitankar1, Dulal Kar1, and Somasheker Akkaladevi2
1School of Engineering and Computing Sciences, Texas A&M University-Corpus Christi, Corpus Christi, Texas, USA
2Department of Computer Information Systems, Virginia State University, Petersburg, Virginia, USA

ABSTRACT - Malicious code detection and removal is critical to the security of a computer system. Virus scanners rely on a database of known signatures for viruses and malware for detection. This research paper presents novel methodologies and tools to detect any malicious code present on windows based machine dynamically, and can be used as a preventive measure to protect the system from being infected. Malicious code analysis can be static and dynamic. Dynamic code analysis has a greater edge over static code analysis as the instructions are analyzed at runtime. Thus polymorphic malware can also be detected. The work presented in this paper uses a newly designed dynamic code technique in conjunction with a developed minifilter driver for malware detection. It runs in a virtual environment to perform the analysis, thus making it impossible for malwares to detect the presence of the developed tool. The minifilter driver is used to monitor the windows API calls, registry changes and used to generate reports which are used to analyze a program as malware or normal. These reports can be analyzed to categorize a program as normal or malware. The developed tool is tested using Symantec malware database and compared with other pre-existing tools to evaluate its effectiveness.

Keywords: Mini filter driver, Malware, Virtual Operating System

I. INTRODUCTION

A. Malicious code

Malicious code is a term used to refer to any code that can cause undesired effects, security breaches and potential damages to the software system without the users consent. A program or software is classified as malware based on the users intent rather than the features of it. Any harmful software cannot be considered as malware. For example, defective software can be legitimate and can still cause potential damage due to the presence of harmful bugs. Malware can be categorized into Viruses, Worms, Trojan horses, Attacker tools etc. [1, 6, 9]. Table 1 lists the various characteristics of malwares. Destructive malware [6] generally spreads through the web, email etc. A special category of malware called data-stealing malware exists. This type of malware intends to steal personal and confidential information of a person or an organization. Once this kind of malware gets successfully installed on the target machine, they can compromise the Intrusion Detection System or anti-virus programs protecting the system.

| Characteristic | Self-contained? | Self-replicating? | What is its method of propagation?
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Virus</td>
<td>No</td>
<td>Yes</td>
<td>User-Interaction</td>
</tr>
<tr>
<td>Worm</td>
<td>Yes</td>
<td>Yes</td>
<td>Self-Propagation</td>
</tr>
<tr>
<td>Trojan Horse</td>
<td>Yes</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>Tracking Cookie</td>
<td>Yes</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>Attacker Tools</td>
<td>Yes</td>
<td>No</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Many factors may leave a system vulnerable to attacks. The most common being the exploits because of bugs in the operating system (OS) design, and the existence of over-privileged users (who can leave the system vulnerable to the malware by making wrong decisions).

B. Malicious Code Analysis

Malicious code analysis is used to refer to the process of determining the intent and nature of the malware sample. For a long time the malicious code analysis was a manual, time consuming and tedious task. Thus there was need for automated systems which could detect the presence of the malware and automatically act to prevent the malware from achieving its intended task.

The most important preventive measures for malware are the virus scanner, but these scanners rely on a database of known signatures for virus. Thus they are restricted to only known viruses and malware, but many new types of malware and viruses attack the computer systems every day. So there is a need for a better malware tracking solution. Whenever a new malware is found, its signature is written to the database of signatures, so that all systems infected with this malware can be easily fixed. Generally malware analysis is conducted by allowing the malware program to be executed in a restricted environment and by observing its actions. The actions of the program are analyzed using a debugger. This manual analysis is a time consuming and
tedious task. Thus there is a need for automated analysis. This automation is generally achieved by executing the affected program in a virtual environment (VE) and recording the actions of the programs, and finally sending the recorded actions to the human analyst [1].

The existing automated malicious code analyzers have shortcomings. The most important among them is the failure of the analyzers because of the presence of detection routines within the malware [1, 3, 4]. The detection routines allow the malware to detect if the program is running in a VE. If so, the malware program acts in a different way, thus hiding its existence. Some malwares have the capability to check the existence of both hardware and software breakpoints, which can be used to detect the existence of a malware. Other problems with the automated malware analysis include the incapability of the tool to detect the complete interaction of the program with the system. Malware analysis can be static and dynamic.

1) Static Analysis: Static analysis is the process of analyzing the malware without actually executing it [3]. In this technique the binary code is converted into corresponding assembler level instructions as shown in Figure 1. After the transformation, control flow and data flow analysis techniques are implemented to draw a conclusion about the programs functionality [5].

![Fig. 1. Framework for Static Analyzer](image)

Generally the analysis is conducted in a VE. Thus the risk of the system being damaged is reduced, because the VE image could be replaced with a new one [5]. One of the significant drawbacks of conducting the analysis in a VE is that the malware could determine that it is running in a VE and may change its behavior accordingly. VE detection tools are easily available [8]. These tools make use of CPU instructions to determine the existence of a VE. Some malwares can detect the VE and change behavior accordingly to hide itself from the defensive system [8]. The work presented in this paper along with the developed tool examines the code in a VE along with a developed minifilter driver and dynamically detects any malicious code present. The minifilter driver is used to monitor the API calls, registry changes, and generate reports. These reports can be analyzed to categorize program as normal or malware.

II. MINIFILTER DRIVER

A. Analyzing a Executable

The main aim of this research is to analyze a given executable and generate a report of the changes made to the system by it. The analysis of the report is performed to decide if the executable is a malware or a normal program. This analysis looks into the list of open, modified, added and deleted files. It also considers changes made to the registry.

The executable is tested in a VE. The newly designed analysis tool has two main components. They are:

1. Minifilter Driver: The minifilter driver is used to dynamically monitor the activities of the program that is being tested. This driver can track the Windows API calls made by the program.

2. Analysis Tool: This tool works along with the driver. It makes use of the track report of the Windows API calls made by the program and generates a report for analysis. The report includes all the file and registry operations made by the program.

The minifilter driver is created using Windows programming with the Windows Driver Kit (WDK) [16]. The analysis component is developed using C and C++ code. Based on the reports generated by the analysis tool, the changes made by the program to the system are taken into
consideration. An analysis is then performed to obtain a decision of whether the program is malicious or normal.

B. Malicious Code Detection

Malicious code detection is the process of detecting various malwares that can cause potential damage to the system. Any defense technology can be separated into two components – a technical component and an analytical component. The technical component is a collection of program functions and algorithms that selects the data that will be analyzed by the analytical component. The analytical component serves as the decision-making entity. It assesses the data provided by the technical component using one or more algorithms and then issues a verdict about the data. The security program will then use the verdict to take action on the malicious program according to the security policy set in the security program. For example, a few of the possible actions that could occur based upon the verdict might be:

1. Notifying the user
2. Requesting further instructions from the user
3. Placing the file in quarantine
4. Blocking unauthorized program actions

Consider the following example code, which is extracted from the assembly code of Bagle virus [5], which is a widespread email-based virus. For ease of presentation and understanding, some simplifications have been made.

```
1  lea edi, ptr [ebp+0x4025]
2  mov edx, 0xef4013a0
3  mov ecx, 0x3ec5
4  loop:    mov al, byte ptr ds[edi]
5        sub al, dl
6        xor al, dh
7        rol edx, cl
8 9        mov byte ptr ds[edi], al
10       inc edi
11      dec ecx
12      jnz loop
13     push edi
14    call 0x7c92a950
```

The key part of the sample code is a loop formed by instructions 4 through 12. The instructions preceding the loop (i.e., instructions 1 through 3) initialize the loop counter (ecx), starting address (edi), and another variable (edx). In the body of the loop a value is fetched from the data segment, performs a calculation based upon that value, and then finally stores the computed value back into the data segment. Following the loop, the program calls a library function that uses the newly computed values. The use of these values triggers the actions of the virus. Many different kinds of obfuscation transformations can be applied to this piece of code to affect mutations of the Bagle virus [5]. Existing techniques proposed by authors in [1, 3, 5, 9] would not be able to correctly identify mutated versions of the Bagle virus. In order to overcome these shortcomings the new novel techniques of detection in a VE using newly designed and developed minifilter drivers is considered.

C. Minifilter Driver

Minifilter: The filter manager is a kernel-mode driver that performs in accordance to a standard file system filter model. The ultimate goal of a filter manager is to provide the generic functionality that is required by file system filter drivers. This functionality is very useful for the third party driver developer to develop and write minifilters for the user applications [13]. The applications developed by minifilters are more robust and versatile [11].

Filter Manager Concepts: All Microsoft Windows OS based systems have a filter manager installed on them [12]. But the filter manager is turned into active mode only when a minifilter driver is loaded. The minifilter driver can register itself with the filter manager to perform filtering of a chosen set of I/O operations.

Load Order Groups: The “load order group” determines the position of a legacy filter driver position in the file system I/O stack relative to other filter drivers [14].

Altitude of a Minifilter: The altitude of a minifilter is the characteristic that identifies the position of a minifilter relative to other minifilters in the I/O stack when the minifilters are loaded [14].

Instance of a Minifilter: The attachment of a minifilter driver at a particular altitude on the file system stack is called an instance of the driver. Figure 3 shows the I/O stack with filter manager and three minifilter drivers [12].

Callback Routines of a Minifilter: A minifilter has the capability to filter all the three major I/O based operations. The three I/O operations are IRP-based I/O operations; fast I/O and file system filter (FSFilter) callback operations.

![Image](Fig_3_I/O_stack_with_filter_manager_and_three_minifilter_drivers.png)

Fig. 3. I/O stack with filter manager and three minifilter drivers

Advantages of Minifilter Drivers: The advantages of minifilter driver over the existing legacy filter driver are:

1. Filter load order control is easy
2. Unloading a minifilter is possible
3. Ability to process only necessary operations
4. Kernel stack is used more efficiently
5. Code redundancy and Less complexity
6. New operations can be easily added
7. Can support multiple platforms easily
8. Better support for user-mode applications

**III. SYSTEM DESIGN**

A. Overview of the system

The use of a dedicated standalone system for testing the malware is not an efficient solution. The dedicated system can be reinstalled after each dynamic test run is performed, but this induces very high cost. In order to overcome the disadvantages of a standalone system, the new design utilizes a VE in conjunction with the developed minifilter drivers, thus limiting the effect of the malware only to the virtual machine (VM) but not the real system. In case of a virtual system, the infected virtual image is replaced with a new one. A major drawback of the virtual solution is that the malware may detect that it is running in a VE and may change its behavior accordingly.

The alternative solution to the above problem is the use of an emulator. A PC emulator is a piece of software that emulates the functionality of a real system including all the real time resources of a system. There is a subtle difference between an emulator and a VM. Virtual system executes a statically dominant set of instructions directly on a real system, whereas an emulator simulates all the instructions in software [4]. Also there is a very important difference between the speed of execution on a real time system and the speed of execution on a virtual OS. This difference can be used by the malware to judge whether it is being run on a virtual OS or a real system. This problem can be solved by using a minifilter driver which can redirect operations and makes a malware believe that it is not being run in a test, virtual or emulator based environment.

B. Information Analyzed

It is possible to classify the types of information that is captured during the analysis phase of the system. Many systems concentrate on the communication between the application and the OS. This includes intercepting system calls and hooking API calls.

There are tools that can be used to list all the windows processes running in a system. Generally these tools are implemented as OS drivers that can intercept the native windows calls. Thus they are invisible to the application that is currently on analysis. There are also tools that can intercept arbitrary user functions, including all windows API calls. This requires some rewriting of the target function. This could again be detected by a malware and thus it may act differently to overcome the detection.

In order to overcome the problem, virtual OS is used along with a minifilter tool. It has the capability of analyzing both the native windows calls as well as the windows API calls, at the same time being unidentified by the malicious code. Because of the use of the minifilter which has complete control over the system, the analysis being performed is more fine grain. This functionality is similar to the debugger but the technique does not make use of break points, which are known to create problems when used for analyzing malicious code. The reason being the software break points can be detected using code integrity checks and the malware could act accordingly [7].

The minifilter tool is used for analyzing windows executables, especially the files corresponding to the portable executable (PE) file format. In this technique the program is tested in a VE and the valid native windows calls and windows API are logged for analysis.

C. Analyzing the reports and decision making

Once a report is generated by the tool, analysis can be performed by the analyst. Based on the operations performed by the program as listed in the report, the analyst can classify a program as a malicious program or a normal program. For example, the following is the analysis of a sample report. The report is from Symantec Antivirus [10]:

<table>
<thead>
<tr>
<th>Name: Auraax.c</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: Worm</td>
</tr>
<tr>
<td>Infection Length: 27,136 bytes</td>
</tr>
</tbody>
</table>

Whenever the worm executes, it replicates itself and infects all the machines that are prone to it. It copies into the system files of a machine as follows:

- `%Program Files%\Microsoft Common\wuauclt.exe`

Once a machine is infected, it creates several files on the infected machine. For e.g. the following files are created:

1. `%Windir%\Temp\D\SINGLE NUMBER\tmp`
2. `%System%\config\systemprofile\Local Settings\History\desktop.ini`
3. `%System%\config\systemprofile\Cookies\index.dat`
4. `%System%\config\systemprofile\Local Settings\Temporary Internet Files\desktop.ini`

After the creation of files, the worm alters the following system processes:

1. `svchost.exe`
2. `explorer.exe`

The worm creates new entries into the windows registry along with modifying existing entries. These entries run every time the system boots. The worm also creates registry entries that have the capability to bypass the Windows firewall. The worm also searches the kernel drivers for .sys extension files for the purpose of modifying them. These files are generally overwritten with a root kit so that the worm can hide itself. The worm also modifies the host
machine files to prevent the host from downloading new updates from Microsoft and other antivirus providers.

D. Testing Environment

The whole testing is performed on a virtual system. The developed tool’s two major components are Minifilter Driver and Analysis Tool.

Once the minifilter is installed and the service is running, the program can be tested. Figure 4 shows the architecture of the developed tool.

![Architecture of the Malicious Code Detection Tool](image)

**Fig. 4. Architecture of the Malicious Code Detection Tool**

The testing is performed in the following steps:

1. First the program name is sent to the driver
2. Driver monitors process operations
   a. driver monitors the process creation
   b. drivers monitors the activity and redirect any operation
3. Driver monitors the exit of the process
4. Detection tool receives notification from the driver and generates report for analysis

**Redirection using the detection tool:** Redirection explains how the detection tool controls the malware by having greater control over the system. The following redirection schemes are used:

1. If a program wants to read an existing registry key:

   `HKLM\Software\Microsoft\Windows\TaskManager\`

   Analyzer will let the program read the value and return to malware program.

2. Suppose malware wants to create a registry key

   `HKLM\Software\Microsoft\Windows\MalwareXXX\`

   Analyzer will create a registry key as:

   `HKLM\Software\Analyzer Sandbox\Malware\`

   The sub key will be:

   `HKLM\Software\Microsoft\Windows\MalwareXXX\`

   The tool makes the malware believe that it is not detected by sending wrong registry keys. It monitors the creation and deletion of registry keys.

3. Whenever malware tries to read a registry key, Analyzer will first check the sandboxed key and if found will return the value from there, otherwise will let the operation continue as usual.

4. What happens if malware modifies any registry key? This operation will be considered as same to creation of key.

5. What happens when file operation is requested? It treats registry names as some file names and file paths.

E. Analysis Process

The analysis process is started by allowing the given program to execute in an emulated environment. When the program executes, all the OS services that are requested by the program are noted. Every action that involves communication with the environment requires some OS services, and it cannot directly interact with the hardware. In a windows OS environment, the application cannot directly interact with the windows native API. They are supposed to make use of the functions provided by the OS to interact with the OS services.

Malware writers make direct use of these native API to avoid any kind of DLL dependencies or confuse the virus scanners. This tool takes care of both Windows API function calls by an application and native API calls of an application, thus making the probability of a malware escaping the analysis very low. The tool will track which OS services are used by a program. This tracking requires us to solve two problems:

1. We must be able to track the execution of a malware process and also distinguish between the instructions executed by a malware process and the instructions executed by a normal process. This is very important because the emulated environment does not only run the instructions of the malware process, but also the native OS instructions and instructions of the other supporting processes.

2. We need to make sure that the native API call or a Windows API call is invoked without any kind of modification to the malware sample.

The PDBR (Page Directory Base Register) can be used to track the execution of the instructions. For this purpose a tool has been designed and developed to makes use of virtual OS and minifilter drivers for detection of malicious code. Figure 5 shows the detection tool in operation.

![Overview of the Malicious Code Analysis Tool](image)

**Fig. 5. Overview of the Malicious Code Analysis Tool**
IV. RESULTS AND EVALUATION

To demonstrate the malware tool in successfully monitoring the actions of malicious code, the tool has been dynamically tested on currently existing malware samples. To evaluate the tools effectiveness, the results are compared against various anti-virus providers solutions.

The malware sample set is chosen from the Symantec’s published list of the most prevalent malware samples [10]. A set of different malware programs that represent a good mix of different malicious code variants currently popular are chosen. From this pool, we choose one working sample for each malware type. All samples chosen for the experiments are run against the online virus scanner provided by Symantec and Kaspersky to verify their integrity. The detection tool was also able to recognize many viruses that are enlisted neither on Symantec’s or Kaspersky’s anti-virus list. The technical detail of malware includes the files, registry entries, processes and services affected by the malware. Generally these changes by a particular virus are not the same on different computers. The reasons can be that the malicious code chooses random file names or a name from a list of options that are not exhaustively covered by the malware description, or it can be a malware variant rather than the malware about which the virus scanner has published the technical details. Table 2 presents some malwares that are analyzed by the tool and compared to the results of the Symantec anti-virus list.

In Table 2 ‘P’ refers to partial matches. The partial matches occur due to the malware dependency on the target system. Generally files are also dependent on the target system but the core files that are created or affected by the malware are always the same. ‘Yes’ represents malware infection and ‘No’ refers to no infection by malware. Also there are few malware samples listed in Table 2 for whom virus definitions are not found on the Symantec’s anti-virus list. These viruses are successfully detected by the developed tool. Whenever some normal process is analyzed, the tool displays the changes made by the program to the system.

<table>
<thead>
<tr>
<th>Malware Name</th>
<th>File</th>
<th>Registry</th>
<th>Process</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>W32.Storm.Worm</td>
<td>Yes</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>W32.HLL.W.Dooms juice</td>
<td>Yes</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>W32.Sality.AE</td>
<td>Yes</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>W32.Qquzlzb.exe</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>W32.Srvcp.exe</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Step 2: The Malprober service has to be started by going to the command prompt in the virtual OS and executing the command “sc start malprober”.

Step 3: The command “dir” has to be executed after step 2 for testing the sample program and generating reports.

Malprober.exe srvcp.exe

Step 4: The Malprober tool automatically generates a report which contains the list of opened files, newly created suspicious files since the last scan, registry keys added, changed and deleted. Figure 6 shows a report that is generated when a program is tested with the Malprober tool.

Step 5: Finally the analysis of report is performed. From Figure 7 it can be noticed that the program modifies the TCP/IP parameters. At the same time, it makes registry changes so that it can automatically get started every time the system boots.

A. Evaluation

The results obtained clearly show that the developed tool is successfully able to identify various categories of malwares as outlined in Table 2. The tool was successfully able to identify main categories of malwares, but due to space constraints a list of important malwares is provided in Table 2. The developed malware tool is able to detect W32.Sality.AE, W32.Qquzlzb.exe, and variants of polymorphic malwares which were not detectable by using the tool developed by duan et al. [4]. The TTanalyze tool developed by bayer et al. [7] was either not successful in detecting the entire list of malwares provided in Table 1 as well as polymorphic malwares. From these results it is clearly evident that our new tool outperforms the TTanalyze tool [7] and the tool by duan et.al. [4].

![Image](image.jpg)
V. CONCLUSION

Because of the time gap between the vulnerability that comes into existence as a result of new malware and the point where a solution for the new malware is generated by the anti-virus provider, every new malware poses a serious threat to computer systems. The developed tool dynamically analyzes the behavior of an unknown program by executing the code in a VE using minifilter drivers. One of the main advantages is the report generated by the tool is simple and easy for an analyst to analyze. All the events are listed in chronological order, which makes it easy to understand. Because the analysis is performed in a VE, the overhead is less.

This dynamic tool makes use of a minifilter driver for tracking the security-related OS events including Windows API functions and native kernel calls. Once a minifilter driver is provided with a program name, it notifies the tool about the activities of the program. The results successfully showed that this tool is able to identify the various categories of malwares provided by the Symantec database and also its ability to detect polymorphic malwares compared to previously available tools.

The tool can be improved to include more classified and detailed reports. It can be improved by creating a database to store the signatures of detected viruses and use this database whenever necessary.

REFERENCES


