Malware Detection Method Focusing on Anti-Debugging Functions

Kota Yoshizaki, Toshihiro Yamauchi
Graduate School of Natural Science and Technology, Okayama University
3-1-1 Tsushima-naka, Kita-ku, Okayama, 700-8530 Japan
yoshizaki@swlab.cs.okayama-u.ac.jp, yamauchi@cs.okayama-u.ac.jp

Abstract—**Malware has received much attention in recent years. Antivirus software is widely used as a countermeasure against malware. However, some kinds of malware can evade detection by antivirus software; hence, a new detection method is required. In this paper, we propose a malware detection method that focuses on Anti-Debugging functions. An Anti-Debugging function is a method that prevents malware analysts from analyzing an application program (AP). The function can form part of benign as well as malicious APs. Our method focuses on a behavioral difference between benign and malicious APs and detects malware by comparing the two behavioral patterns. Evaluation results with malware confirmed our method to be capable of successfully detecting malware.**

Keywords-**security; malware detection; anti-debugging**

I. Introduction

The amount of malware has been increasing in recent years [1], [2], [3]. This is evident from the damage that has been caused by malware threats all over the world. For example, in 2012, an incident relating to remote computer control [4] caused serious problems in Japan. Moreover, a few years ago, the Stuxnet worm [5], which exploited computer vulnerability, became a serious problem in overseas countries. These examples show that the damage caused by malware has become a serious problem all over the world.

Antivirus software vendors have developed antivirus software to detect and protect computers against malware. However, some malware evade detection by antivirus software [6], [7]. Agobot is one of the types of malware whose source code is propagated over the Internet. Its propagation and its capability to modify itself, have led to the creation of a large number of subspecies. Many antivirus software programs detects malware by using a provided signature. However, signature based detection is not an effective approach when the source code is propagated such as is the case with Agobot. Moreover, Agobot has a built-in Anti-Debugging function [8], [9] that is intended to disrupt analytical investigation. Therefore, a new malware detection method is required.

There are many types of malware with Anti-Debugging functions [10] and there are two kinds of these functions. The first kind prevents static analysis, while the other kind prevents dynamic analysis. Static analysis uses a disassembling method to analyze an application program's (AP's) source code; hence malware aimed at counteracting static analysis uses a packer, to obfuscate the execution code. Dynamic analysis uses a debugger to analyze the malware's approximate behavior, and malware intended to counteract this kind of analysis uses a method to detect the debugger. For our purpose, prevention of this nature is defined as evasion. Evasion is therefore considered preventive behavior with the purpose of evading dynamic analysis. Evasion complicates the analysis of malware.

To counteract malware that contains an Anti-Debugging function, Matsuki et al. [11] proposed a method to prevent the malware from running. This method operates by deactivating the malware by putting it into an analyzed state. However, there are two problems associated with this method. First, this method prevents the activity of all APs that contain an Anti-Debugging function, including benign ones. Second, this method dealt with only one Anti-Debugging function.

In this paper, we propose a malware detection method based on the Anti-Debugging function. Our method is designed to modify the return value of the API that is used by the Anti-Debugging function to either put the AP into an analyzed or a non-analyzed state. Our method is based on the following principles. Benign and malicious APs that are put into an analyzed state, take evasive measures to evade the analysis. Benign APs that are put into a non-analyzed state display benign behavior. In contrast, malicious APs that are put into a non-analyzed state, display malicious behavior. Thus, there is a behavioral difference between benign and malicious APs in the non-analyzed state. Our method is designed to detect malware dynamically by focusing on this behavioral difference. The contributions made in this paper are as follows.

The incorporation of Anti-Debugging functions in malware complicates the identification of malware analysts. Our method presents an efficient way to identify malware by detecting it dynamically without the need for static analysis.

II. Malware Detection Method Focusing on Anti-Debugging Functions

A. Anti-Debugging function

Table I shows a list of Anti-Debugging functions together with the APIs in which they are used. Anti-Debugging function is a method that is used by malware to prevent analysts from analyzing or reverse engineering APs.

Table I
LIST OF ANTI-DEBUGGING FUNCTIONS

Anti-Debugging Function	API Name
Detecting process	CreateToolhelp32Snapshot Process32First Process32Next
Detecting windows	EnumWindows GetClassName
Detecting device file	CreateFile
Measuring execution time	GetTickCount timeGetTime QueryPerformanceCounter
Reboot malware	CreateProcess
Obtaining BeingDebuggedFlag	IsDebuggerPresent
Attaching to process by the debugger	CheckRemoteDebuggerPresent
Obtaining context of the thread	GetThreadContext
Obtaining the return value of the API	OutputDebugString
Attaching by debugger	CreateProcess DebugActiveProcess

1) Anti-Debugging function against static analysis: Functions that prevent static analysis use a packer as a tool to compress the executable file while it is executable. An attacker can shorten the download time of malware by using a packer to compress the malware. In addition, a packer obfuscates and encodes execution code, thereby making it difficult for malware analysts to analyze malware. A packer, therefore complicates static analysis by malware analysts.

2) Anti-Debugging function against dynamic analysis: Malware uses the technique of evasion to prevent dynamic analysis from occurring. The use of this technique requires malware analysts to perform a static analysis instead of a dynamic analysis. However a static analysis requires abundant time to clarify the malware's behavior, which delays its termination. Therefore, any research effort aiming to address the problem of evasion relating to malware detection would have to focus on the Anti-Debugging function against dynamic analysis.

B. Concept of proposed method

Figure 1 and 2 present an overview of our method. Figure 1 shows the execution of a malware in the analyzed state. Figure 2 shows the execution of a malware in the non-analyzed state. Our method executes the AP in both of these two states and focuses on the behavioral difference between them to detect malware.

The process flow shown below represents the case in which the malware is placed in the analyzed state. This process flow corresponds to that shown in Figure 1.

1) An AP invokes an API which is used in Anti-Debugging function.
2) An API Hooker hooks that API and returns the value representing "analyzed".
3) An AP does evasion.
4) An API Hooker outputs the log of the evasion.
5) An AP resumes execution.

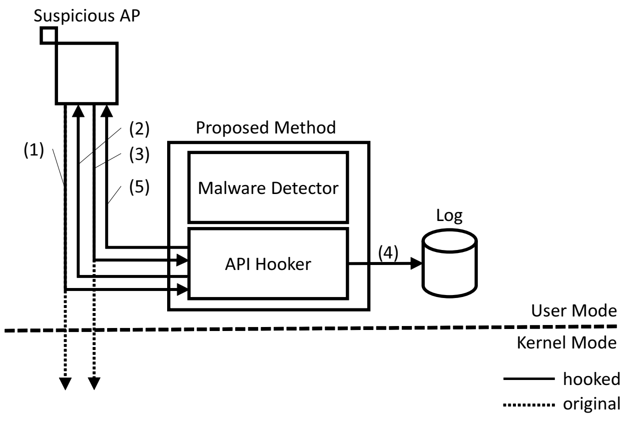

Figure 1. Executing malware in the analyzed state

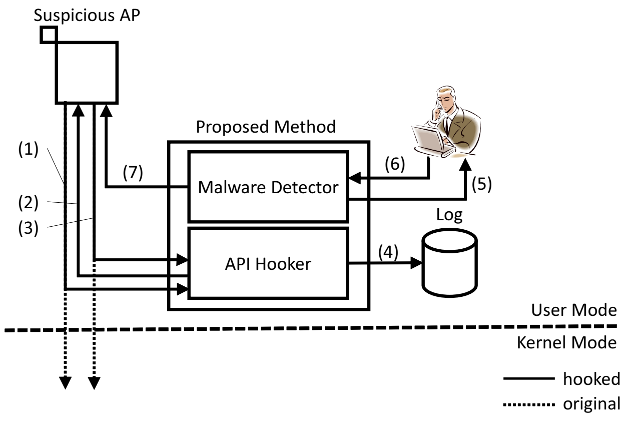

Figure 2. Executing malware in the non-analyzed state

The process flow shown below represents the case in which the malware is placed in the non-analyzed state. This process flow corresponds to that shown in Figure 2.

1) Malware invokes an API which is used in Anti-Debugging function.
2) API Hooker hooks the API and returns the value representing "non-analyzed".
3) The malware invokes an API for malicious behavior and API Hooker hooks this API.
4) API Hooker reads the logs of evasion.
5) The Malware Detector detects the malware and asks the user whether to terminate the malware or not.
6) The user returns a determination to the Malware Detector.
7) If the user specified termination, the Malware Detector terminates the malware. Otherwise, the Malware Detector resumes the malware.

Our method defines adding an entry for start-up to the registry as malicious behavior. This is because malware often creates a registry entry to enable itself to run automatically during computer start-up [12]. To add an entry to the registry, it is required to invoke API. We define the date, time, process ID, the name, and the arguments of the hooked API as malicious behavior information.

C. Requirements and Challenges

The detection of malware requires the observation of evasion and, consequently, malicious behavior. Observing evasion requires placing the AP into an analyzed state. The requirements of this paper are as follows:

Requirement 1. Place malware into analyzed state.

Requirement 2. Place malware into non-analyzed state.

Requirement 3. Observe evasion and output logs of evasion.

Requirement 4. Observe malicious behavior.

Requirement 5. Detect malware based on a behavioral difference between the benign AP and the malware.

To satisfy Requirement 1 and 2, it is necessary to hook the API that is used for the Anti-Debugging function and to modify the return value of the hooked API. For instance, if the return value of the IsDebuggerPresent API is TRUE, the AP recognizes that it is in the analyzed state, whereas if the return value of the IsDebuggerPresent API is FALSE, the AP recognizes that it is in the non-analyzed state. To satisfy Requirement 3, the API that is used for evasion has to be hooked and the outputting logs of the hooked API are required while satisfying Requirement 1. To satisfy Requirement 4, it is necessary to hook the API, which opens or creates the registry entry for start-up. Requirement 5 requires an assessment of the AP to determine whether it is malicious or benign based on the observation of evasion and the malicious behavior information. Moreover, the result of the judgment would determine whether malware is detected.

To satisfy the Requirements above, the following challenges are required.

Challenge 1. It is possible to hook the API used for the Anti-Debugging function.

Challenge 2. It is possible to hook the API used for evasion and the output logs of the hooked API.

Challenge 3. It is possible to hook the API used for malicious behavior.

Challenge 4. It is possible to detect malware based on the output logs of evasion and malicious behavior information.

D. API Hooker

The API Hooker hooks the following APIs: IsDebuggerPresent, RegCreateKeyExA, and ExitProcess. API Hooker creates logs of evasion as output and sends the malicious behavior information to the malware detector. Some Anti-Debugging functions invoke IsDebuggerPresent to detect a debugger and ExitProcess as evasion. To achieve Challenges 1 and 2, IsDebuggerPresent and ExitProcess are hooked, following which the output information of ExitProcess is used as a log. RegCreateKeyExA is used for malicious behavior. Therefore, Challenge 3 is achieved by hooking RegCreateKeyExA. Challenge 4 is achieved by generating an evasion logs and a malware detection log as output as described below. The API Hooker generates output containing the hooked API's date, time, process ID, name, and arguments as hooked API information.

E. Malware Detector

The Malware Detector is a function that uses the hooked API information from the API Hooker to detect malware. At first, the Malware Detector accepts as input the evasion information obtained from the logs. Second, the Malware Detector receives the malicious behavior information from the API Hooker. If the conditions below are met, the Malware Detector detects and declares AP to be malware.

Conditions 1. Evasion information is output in the form of logs.

Conditions 2. The API Hooker detected malicious behavior information.

When malware is detected, the Malware Detector uses a message box to notify users that malware has been detected.

III. Experimental results

A. Evaluation environment

Our experiment was performed by running Windows Vista SP2 on a VirtualBox 4.2.6 as a guest OS. The malware Agobot, which has an Anti-Debugging function, was used. The experiment involved Agobot attempting to detect a debugger by invoking IsDebuggerPresent once Internet connection is established. However, our machine would have been able to perform the role of a bot by executing Agobot if our machine were to be connected to the Internet. Therefore Agobot was modified to call the Anti-Debugging function before the Internet connection was established. Agobot was therefore executed in an environment that was isolated from the Internet.

B. Malware detection evaluation

At first, Agobot was executed in the analyzed condition. Specifically, the API Hooker hooks IsDebuggerPresent and changes its return value to TRUE. Figure 3 shows the log resulting from the evasion and shows that the time between the call of IsDebuggerPresent and the call of ExitProcess to be about one second. The Malware Detector detects Agobot if ExitProcess is called within five seconds. Therefore, in this case, Malware Detector detects Agobot and considers it malware. Figure 4 shows the logs of malicious behavior. This log shows a call of RegCreateKeyExA with its argument is *SOFTWARE¥Microsoft¥Windows¥Current¥Version¥Run*. This registry entry was created by Agobot for start-up and implies that the call of RegCreateKeyExA indicates malicious behavior. The Malware Detector accepts as input the

```
2013/2/7,16:01:21:294,1752,4028,IsDebuggerPresent,VOID
2013/2/7,16:01:22:567,1752,4028,ExitProcess,VOID
```

Figure 3. Log of evasion

```
2013/2/7,16:01:21:294,1752,4028,IsDebuggerPresent,VOID
2013/2/7,16:01:22:567,1752,4028,ExitProcess,VOID
2013/2/7,16:15:25:802,3780,236,IsDebuggerPresent,VOID
2013/2/7,16:15:26:673,3780,236,RegCreateKeyExA,SOFTWARE¥Microsoft
¥Windows¥CurrentVersion¥Run
```

Figure 4. Log of malicious behavior

logs of evasion information, before receiving the malicious behavior information that enables it to detect the malware.

C. Consideration

Our method executes the AP twice in order to form a judgment as to whether the AP is malicious or benign. The first execution enables evasion to be observed, while the second execution enables the detection of malicious behavior. However, the observation of both of these behaviors requires modification of the return value of the API for Anti-Debugging. This was accomplished by preparing two DLLs to change the return value. One of the DLLs returns the value TRUE for the IsDebuggerPresent API, whereas the other returns the value FALSE. The name of the DLL had to be changed each execution time the method was executed, and this was not efficient.

Our method was only tested with one Anti-Debugging function. Future efforts would have to consider more malware by including other Anti-Debugging functions.

IV. Conclusions

We proposed a malware detection method that focuses on the Anti-Debugging function. An evaluation of our method using malware showed it capable of successfully detecting the malware.

Our method executes and detects malware in two states, an analyzed state and a non-analyzed state. The method focuses on the behavioral difference between benign and malicious software, which has an Anti-Debugging function. In the analyzed state, the AP and the malware perform the action of evasion. In the non-analyzed state, the AP behaves benignly, but the malware exhibits malicious behavior. Our method focuses on this behavioral difference to detect malware.

The results of our experiments showed that our method could observe the Anti-Debugging function, evasion, and malicious behavior and is therefore capable of detecting malware. Furthermore, our work was able to verify a user's ability to terminate malware successfully when the user was presented with the possibility of doing so.

References

[1] AVTEST The Independent IT-Security Institute, "Statistics, Malware," http://www.avtest.org/en/statistics/malware/

[2] McAfee, "McAfee Threats Report: Fourth Quarter 2014," http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q4-2013.pdf

[3] Shadowserver Foundation, "Statistics - Malware," http://www.shadowserver.org/wiki/pmwiki.php/Stats/Malware

[4] J. Hamada, "Man Arrested in Relation to the "Remote Control Virus"," http://www.symantec.com/connect/blogs/man-arrested-relation-remote-control-virus

[5] S. Karnouskos, "Stuxnet worm impact on industrial cyber-physical system security," 37th Annual Conference on IEEE Industrial Electronics Society, pp.4490-4494 (2011).

[6] AV Comparatives, "Real World Protection-Test - March 2014," http://chart.av-comparatives.org/chart1.php

[7] I. You and K. Yim, "Malware Obfuscation Techniques: A Brief Survey," Proc. 2010 International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA), pp.297-300 (2010).

[8] Z. Qi, B. Li, Q. Lin, M. Yu, M. Xia and H. Guan, "SPAD: Software Protection Through Anti-Debugging Using Hardware-Assisted Virtualization," Journal of Information Science and Engineering, vol.28, pp.813-827 (2012).

[9] S. Gao, Q. Lin, M. Xia, M. Yu, Z. Qi and H. Guan, "Debugging classification and anti-debugging strategies," Proc. SPIE, vol.8350, pp.83503C-83503C.6 (2011).

[10] R. Branco, G. Barbosa, and P. Neto, "Scientific but Not Academical Overview of Malware Anti-Debugging, Anti-Disassembly and Anti-VM Technologies," Blackhat USA'2012.

[11] T. Matsuki, Y. Arai, M. Terada, and N. Doi, "Proposal of Malware Activity Control Method Turning Anti-analysis Function to Advantage," IPSJ Journal, vol.50, no.9, pp.2118-2126 (2009). (in Japanese)

[12] F-Secure, "News from the Lab," http://www.f-secure.com/weblog/archives/00001207.html