A Methodology for Hook-Based Kernel Level Rootkits

Chien-Ming Chen[1,2], Mu-En Wu[3], Bing-Zhe He[4], Xinying Zheng[1], Chieh Hsing[4], and Hung-Min Sun[4]

[1] School of Computer Science and Technology, Harbin Institute of Technology Shenzhen Graduate School, Shenzhen, China
dr.chien-ming.chen@ieee.org, xinying_15@163.com
[2] Shenzhen Key Laboratory of Internet Information Collaboration, Shenzhen, China
[3] Department of Mathematics, Soochow University, Taipei, Taiwan, R.O.C.
mnasia1@gmail.com
[4] Department of Computer Sciences, National Tsing Hua University, Hsinchu, Taiwan, R.O.C.
{ckshjerho,jhsing}@is.cs.nthu.edu.tw, hmsun@cs.nthu.edu.tw

Abstract. It is easy to discover if there are hooks in the System Service Dispatch Table (SSDT). However, it is difficult to tell whether theses hooks are malicious or not after finding out the hooks in the SSDT. In this paper, we propose a scheme that evaluates the hooks by comparing the returned results before hooking and after hooking. If a malicious hook which hides itself by the way of modifying the parameters passed to the Native API, we can easily detect the difference. Furthermore, we use a runtime detour patching technique so that it will not perturb the normal operation of user-mode programs. Finally, we focus on the existing approaches of rootkits detection in both user-mode and kernel-mode. Our method effectively monitors the behavior of hooks and brings an accurate view point for users to examine their computers.

Keywords: Security, SSDT, Rootkits.

1 Introduction

With the rapidly growth of computer system, more and more issues have been concerned. One of the most concerned issue is security [15,1,3,12]. Rootkits is a technique used by a malicious program to hide itself. It has been widely used in software, even in embedded systems. The rootkits have became a serious threat to our computer. These rootkits can be classified into two primary classes: (1) User-mode rootkits and (2) Kernel-mode rootkits. User-mode rootkits may hide itself through High-Level API intercepting and filtering. This kind of rootkits can easily be detected by existing anti-rootkits software. In this paper, we focus on Kernel-mode rootkits which are harder to detect. Kernel-mode rootkits are extremely dangerous because they compromise the innermost of an operation system.

X. Huang and J. Zhou (Eds.): ISPEC 2014, LNCS 8434, pp. 119–128, 2014.

Kernel-mode rootkits often use SSDT (System Service Dispatch Table) Hooking, or DKOM (Direct Kernel Object Manipulation) to achieve information manipulation. Although there is a lot of existing anti-virus softwares that can detect malicious code, when deal with rootkits, they cannot determine if its behavior is suspicious. Besides, several methods for detecting kernel-mode rootkits have been proposed [6,9,10,11,14,16]. However, if a user employs these softwares (e.g., Rootkit Unhookers [2], Rootkit Hook Analyzer [13]) to do the analysis and find out a suspicious driver, he can remove the driver immediately. However, a wrong decision may disable the functionality of some programs, such as anti-virus software, or some on-line games.

In this paper, we propose a scheme to evaluate the hooks by comparing the returned results before hooking and after hooking. Through this comparison, if a malicious hook which hides itself by modifying the parameters passed to the Native API, we can easily detect the difference. Besides, we use a runtime detour patching technique to not to perturb the normal operation of user-mode programs.

2 System Overview

According to our observation of the behavior of a hooked SSDT-based rootkit, these kinds of rootkits usually hook the SSDT to achieve information manipulation. Normally, a hooked SSDT-based rootkit hooks the SSDT to achieve information manipulation. Even though existing tools are sufficient to detect hooks in SSDT; however, we have to make a decision with caution whether to remove the hook or not. The decision we made will influence the usability of the computer.

In this section, we first describe design goals and assumptions. Then, we explain advantages of our scheme.

2.1 Design Goals and Assumption

Since we target at hooked SSDT-based rootkits, user-mode rootkits and other parts of kernel-mode Rootkit (e.g., DKOM, Inline Function Patch) are beyond our scope. Our scheme focuses on a machine which is probably infected with rootkits. TAN [17] proposed a framework to defeat kernel Native API hookers by SSDT restoration. We are inspired by his idea. In our scheme, we assume that in the beginning, there is no other kind of rootkits running on this machine.

Our scheme has the following two design goals: (1) Effectively analyzing the situation in the state before SSDT restoration $SSDT_{<before>}$ and the state after SSDT restoration $SSDT_{<after>}$ [17]. (2) This program should not perturb the normal operation of the program.

First, the SSDT restoration scheme [17] does not mention how to compare these two states accurately, because after we executed the SSDT restoration. We cannot reconstruct exactly the same parameters in the stack for the further comparison. In other words, when a program executes a non-specific Native API