Proposal of Kernel Rootkits Detection Method by Monitoring Branches Using Hardware Features

Yohei Akao, Toshihiro Yamauchi
Graduate School of Natural Science and Technology
Okayama University, Okayama, Japan
Email: akao@swlab.cs.okayama-u.ac.jp, yamauchi@cs.okayama-u.ac.jp

***Abstract*—Attacks on computer systems have become more frequent in recent years. Attacks using kernel rootkits pose a particularly serious threat. When a computer system is infected with a kernel rootkit, attack detection is difficult. Because of this, handling the attack will be delayed causing an increase in the amount of damage done to the computer system. This paper proposes a new method to detect kernel rootkits by monitoring the branch records in kernel space using hardware features of commodity processors. Our method utilizes the fact that many kernel rootkits make branches that differ from the usual branches. By introducing our method, it is possible to detect kernel rootkits immediately and, thereby, reduce damages to a minimum.**

***Keywords*-Security, kernel rootkit, last branch record**

I. Introduction

Rootkits are malicious programs that hide malicious behavior from the user of the computer where they are installed. There are two types of rootkits: user rootkits that run at the user level and kernel rootkits that run at the kernel level. Kernel rootkits modify operating system (OS) kernels and rewrite the data output by the OS. Therefore, detecting methods based on the output data of the OS are ineffective. For example, anti-virus software running at the user level cannot detect kernel rootkits. Thus, detecting kernel rootkits is difficult and various methods to detect them have been proposed. Ikegami et al. [1] mentioned that the existing methods do not resolve all of the following problems simultaneously: (1) cannot detect kernel rootkits immediately, (2) cannot keep the scalability of the OS kernel, and (3) cannot extend to different OS and OS versions. To resolve those problems, Ikegami et al. [1] proposed a method to detect kernel rootkits by checking the kernel stack. However, this method (4) cannot detect kernel rootkits that use instructions that do not push data into the kernel stack (e.g., the *jmp* instruction).

This paper proposes a new method to detect kernel rootkits. Our method detects kernel rootkits by monitoring the branch records in kernel space recorded by hardware features of commodity processors. Our method utilizes the fact that many kernel rootkits make branches that differ from the usual branches.

The contributions made in this paper are as follows: This paper proposed the efficient way to detect kernel rootkits using hardware features. Our method resolves all problems (1)-(4) simultaneously.

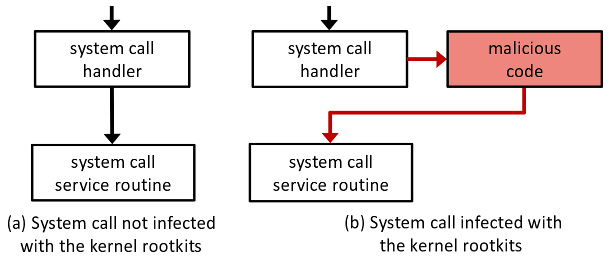
system call handler
system call service routine
(a) System call not infected with the kernel rootkits
system call handler
malicious code
system call service routine
(b) System call infected with the kernel rootkits

Figure 1. Changes in the control-flow when system call control-flow is modified

II. Kernel Rootkits Detection Method by Monitoring Branches Using Hardware Features

A. Concept of proposed method

Our method utilizes the fact that many kernel rootkits make branches that differ from the usual branch path. Previous research [2] indicates that 96% of all kernel rootkits employ control-flow modification, making branches different from usual. For example, Figure 1 shows the change in the control-flow when the system call control-flow is modified by kernel rootkits. Usually, after invoking a system call, the control moves from the system call handler to the each system call service routine. On the other hand, when a computer system is infected with kernel rootkits, the control moves from the system call handler to the malicious code prepared by the attacker before moving to each system call service routine. In the malicious code, the processing that hides attacks is executed. Our method detects kernel rootkits by monitoring branch records in kernel space and by detecting control-flow modification. Our method uses Last Branch Record, a recent feature of Intel processors for monitoring the branch record in kernel space.

B. Last Branch Record

Last Branch Record (LBR) is a recent feature of Intel processors that was introduced in the Nehalem architecture. When LBR is enabled, the CPU records the address of a branch instruction and its target instruction (branching data) on LBR stack registers. The LBR stack can store 16 entries.

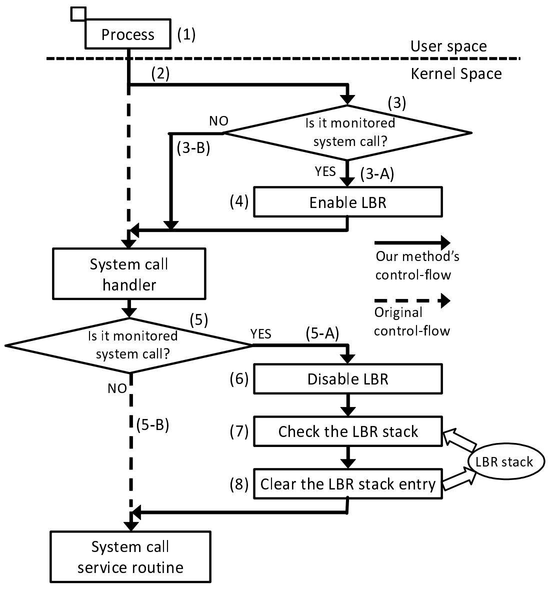

Figure 2. Processing flow of our method

When more than 16 branching data entries are recorded, the oldest stack data is overwritten. Monitoring branch records using LBR has the following advantages:

(1) It can record all branching data in the kernel. Therefore, it can monitor branch records recorded by instructions that do not push data into the kernel stack.
(2) It is transparent to the OS structure.
(3) It generates minimal overhead [3].

C. Overview of our kernel rootkits detection method

Our method detects kernel rootkits that modify control-flow of the system call by monitoring the branch records using LBR in Linux. Figure 2 shows a processing flow of our method. Our method detects kernel rootkits that modify the control-flow of the system call as follows:

(1) A user program invokes a system call.
(2) Our method hooks the transition to the system call handler.
(3) Our method judges whether the invoked system call is a watched system call and the following processing is executed.
    (A) If the invoked system call is a watched system call, then control is given to Step (4).
    (B) Otherwise, our method does nothing and control is given to the system call handler.
(4) LBR is enabled (to start monitoring branches) and control is given to system call handler.
(5) The following processing is executed.
    (A) If the invoked system call is a watched system call, our method hooks the transition to each system call service routine and control is given to Step (6).
    (B) Otherwise, control is given to each system call service routine.
(6) LBR is disabled (to stop monitoring branches).
(7) Our method checks branching data in the LBR stack. If branching data in the LBR stack is abnormal (see Case (2) described in II-D), our method alerts the user.
(8) Branching data in the LBR stack is cleared and control is given to each system call service routine.

Using these steps, our method monitors the branch records between the invoking system call and the transition to each system call service routine.

D. Checking branching data in LBR stack

Our method detects kernel rootkits based on the quantity of branching data in the LBR stack.

In our method, branching data recorded by LBR is classified in the following four ways:

(1) When the computer system is not infected with kernel rootkits, LBR records two pieces of branching data.
(2) When the computer system is infected with kernel rootkits, LBR records more than two pieces of branching data by processing the kernel rootkits.
(3) When the process is traced, LBR records more than two pieces of branching data by processing the trace.
(4) When an interrupt occurs, LBR records more than two pieces of branching data by processing the interrupt.

When quantity of branching data contained in the LBR stack is equal to two, our method determines that the computer system is not infected with kernel rootkits. When quantity is greater than two, our method verifies whether or not the process is traced intentionally by the user. When the process is not intentionally traced, our method determines that the computer system is infected with kernel rootkits. Handling the case in which an interruption occurs is an issue that we will consider in the future.

III. Conclusion

This paper proposed a method to detect kernel rootkits by monitoring branches in kernel space using LBR. Our method enables the detection of kernel rootkits that previous kernel stack comparison methods could not detect. In future work, we will handle the case in which interrupts occur and also evaluate the performance of our method.

References

[1] Y. Ikegami, and T. Yamauchi, "Proposal of Kernel Rootkits Detection Method by Comparing Kernel Stack," IPSJ Journal, Vol.55, No.9, pp.2047-2060, 2014 (in Japanese).

[2] N.L. Petroni Jr, and M. Hicks, "Automated Detection of Persistent Kernel Control-Flow Attacks," Proc. 14th ACM Conference on Computer and Communications Security (CCS'07), pp.103-115, 2007.

[3] V. Pappas, M. Polychronakis, and A.D. Keromytis, "Transparent ROP Exploit Mitigation using Indirect Branch Tracing," Proc. 22nd USENIX Security Synposium, pp.447-462, 2013.