VM-CFI: Control-Flow Integrity for Virtual Machine Kernel Using Intel PT

Donghyun Kwon, Jiwon Seo, Sehyun Baek, Giyeol Kim, Sunwoo Ahn, and Yunheung Paek(✉)

Department of Electrical and Computer Engineering, Seoul National University, Seoul, South Korea
{dhkwon, jwseo, shbaek, gykim, swahn}@sor.snu.ac.kr, ypaek@snu.ac.kr

Abstract. Nowadays cloud computing technology is used for a variety of services, such as the internet of things and artificial intelligence. However, as more data is being processed in the cloud, there is growing concern about security issues in the cloud computing environment. To solve this concern, many studies have been conducted to ensure the integrity of virtual machines in a cloud computing environment. However, in the case of the control-flow integrity for the virtual machine, existing studies are not only necessary to modify the kernel code, but also cannot protect it efficiently. In this paper, we propose VM-CFI which efficiently protects the control-flow integrity of VM kernel without modification of VM kernel in cloud computing environment. For this purpose, VM-CFI utilizes Processor Trace (PT), a hardware feature that is recently supported by Intel architecture. According to the experimental results, VM-CFI incurs on average 4.2% overhead.

Keywords: Control-flow integrity · Intel Processor Trace · Cloud computing

1 Introduction

Today, through cloud computing environments, many services are being offered to people from artificial intelligence services such as speech recognition technology to internet of things services such as a smart home. As the number of such services increases, much sensitive information is processed in the cloud environment, which is causing concern about security problems in the cloud computing environment [1]. Therefore, as a way to enhance security in cloud computing environment, a cloud administrator has to ensure the integrity of the kernel of a VM in which individual cloud users execute her application. To verify memory integrity of the kernel, previous studies have examined either the hardware events occurring in the VM through virtualization, or the information which extracted when kernel executes instrumented code [2, 3].

However, existing works do not guarantee Control-Flow Integrity (CFI) of the guest OS efficiently. CFI is to check whether the target program is executed as a legitimate control-flow. CFI studies [4–6] for kernels have modified the kernel code or binary to verify the control-flow of the kernel. However, there are two reasons why this approach is difficult to use in a cloud environment. First, there is a problem that it is difficult to apply to all types of guest OS supported in a cloud environment. In the cloud computing environment, cloud administrators support a variety of vendors' kernels, and even multiple versions of each kernel. Therefore, the approach based on kernel modifications is not practical because it requires the cloud administrator to modify the kernel code or binary each time a new kernel is introduced or the kernel is patched. Next, it may also be a problem in terms of Quality of Service (QoS) guarantee of cloud users. Cloud users contract with the cloud administrator to determine the hardware resources used in the virtual machine. And while the cloud user is using it, the cloud administrator must provide a certain QoS to the cloud user according to the contract. However, if the kernel code is modified for CFI, this can cause unpredictable performance overhead and it would be difficult to guarantee QoS.

In this paper, we propose VM-CFI which protects the control-flow integrity for existing guest OS efficiently, without modification of guest OS. For this purpose, VM-CFI use a Processor Trace (PT) provided by Intel architecture. PT provides the control-flow information of the currently executed processor as a stream of the packet through the hardware, so that VM-CFI can obtain the control-flow information without modifying the kernel code of a virtual machine. In addition, since the CFI verification of VM-CFI is independently operated with the monitored virtual machine, VM-CFI imposes small performance overhead on the monitored virtual machine.

VM-CFI makes the following contributions:

(1) **No kernel modification.** We enforce control-flow integrity on guest OS without guest OS modification, so user don't need to rewrite the source code of kernel to update it.
(2) **Low overhead.** We implement VM-CFI which has low overhead by using hardware features in the Intel processor called Intel PT.

2 Background

2.1 Intel Processor Trace

Intel PT is a hardware feature provided to enable the extraction of performance flow information of Intel processors with low overhead. Intel PT stores control-flow information in a memory buffer in the form of several packets. Table 1 summarizes the packets to be used for analyzing the control-flow of the virtual machine among these packets. Intel PT also provides packet-related settings and filter-related functions depending on the purpose of use. The details of this are described in the related manual [7].

2.2 Privilege Levels on Intel Architecture

Intel architecture has a privilege level called Ring. Ring 0 is the kernel privilege and Ring 3 is the privilege that the user application is running on. Ring 0 has higher privilege than Ring 3, so the various system configuration registers can be changed only in Ring 0.

Table 1. Intel PT packets analyzed by VM-CFI

Packet name	Packet description
Paging Information Packet (PIP)	This is a packet that occurs whenever there is a change in the CR3 register that contains the current paging information. In particular, it includes non-root (NR) bits that distinguish between execution of the guest OS and execution of the host kernel
Target IP Packet (TIP)	It is a packet that records the destination address when indirect branch, exceptions, and interrupts occur
Flow Update Packets (FUP)	It is a packet that records the source address when an interrupt and exception occurs

In addition to these ring-based privileges, the virtualization extension VT-x, which is provided by Intel architecture, has a separate CPU operating mode called root/non-root. Therefore, the application of the virtual machine (Ring 3) and the kernel of the virtual machine (Ring 0) is executed in the non-root mode, and the host application (Ring 3) and the host kernel including the virtual machine manager (VMM) (Ring 0) are executed in the root mode. Switching between these root and non-root modes will result in VM entry and VM exit events. Figure 1 is an abstracted figure summarizing these terms.

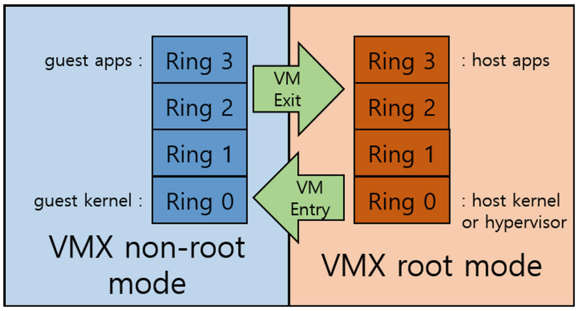

Fig. 1. Privilege levels on Intel architecture

3 Threat Model

In this paper, we assume that a remote attacker who attacks the victim's guest OS through a network connection, or a malicious user who attacks the guest OS to attack the cloud computing environment. Such an attacker does not have the right to modify the guest OS, but can perform a control-flow hijacking attack using a vulnerability existing in the guest OS.

VM-CFI, on the other hand, does not consider internal attacks which caused by a malicious cloud provider. Also, in order to protect the code integrity of existing guest OS, it is assumed that existing solutions are already adopted in the cloud environment [8], and VM-CFI only is designed to ensure CFI of the guest OS.

4 Design

4.1 Design Principles

(P1) Minimize Modification to Existing Software. A cloud provider supports various versions of the kernel for users in the cloud environment. If she need to modify the kernel to guarantee CFI, she have to revise it every time the kernel version is updated, which will result in less practicality of VM-CFI.

(P2) Minimize the Performance Overhead of Monitored Virtual Machines. Cloud users must be assured of the QoS they have contracted with the cloud provider. Therefore, if substantial performance overhead occurs in a cloud user's virtual machine to guarantee CFI, this may cause problems in terms of the QoS.

4.2 Design Overview

As shown in Fig. 2, the overall operation of VM-CFI can be divided into (1) offline binary analysis and (2) runtime integrity enforcement. In the offline binary analysis, the binary analyzer analyzes the guest kernel binary to generate the control-flow metadata to be used for the integrity verification before the monitored VM runs. In this control-flow metadata, information for each basic blocks existing in the guest kernel code is stored. At this time, as a binary of the guest OS is not modified, we can satisfy P1. Then, while the monitored VM is running, VM-CFI uses Intel PT to extract the control-flow trace of the system. During a packet decoding, only the trace corresponding to the guest kernel of the monitored VM is extracted and transmitted to the integrity checker. The integrity checker verifies CFI based on the received trace using control-flow metadata. Since this series of integrity verification process works as a process of the host kernel which is independent of the performance of the monitored VM, P2 can be satisfied.

4.3 Control-Flow Integrity Policy

In the case of a control-flow hijacking attack, the attacker tampers the control-flow and attempts to execute the malicious code. To do this, the attacker have to use a control transfer instruction. The control transfer instruction can be divided into two types. First, the direct control transfer instruction is an instruction in which the control transfer target address is stored in the code. So to exploit this instruction, an attacker should modify code memory region. However, it is well known that this code manipulating behavior can be easily defended through the existing W^X policy [8]. On the other hand, in the case of an indirect control transfer instruction, the control transfer target address is determined by the general purpose registers. Therefore, an attacker can manipulate the control-flow without modifying the code. To protect this, in VM-CFI, it is always enforced to jump to the basic block entry for the indirect control transfer instruction executed by the guest kernel.

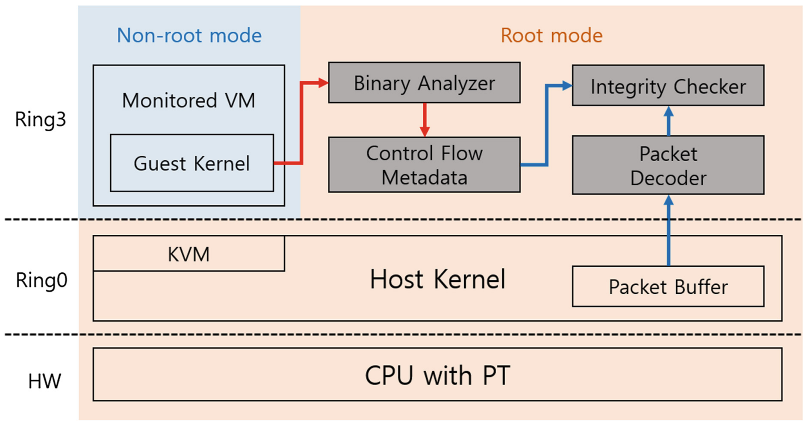

Fig. 2. System overview of VM-CFI. Modules marked with gray boxes correspond to VM-CFI, and red arrows indicate the operation of offline binary analysis. A blue arrow indicates the runtime integrity enforcement process. (Color figure online)

Since VM-CFI monitors CFI of the guest OS, in addition to branch behavior of such indirect control transfer instructions, branch behavior caused by interrupt and exception should be handled. This is because the control-flow is changed due to these events during the kernel execution, and the attacker can manipulate the control-flow in the guest OS by using this system events. For this purpose, we also have verified the return address of a handler code in hypervisor.

4.4 Offline Binary Analysis

In order to guarantee the CFI policy described above, it is necessary to know information about the start address of basic blocks existing in the guest OS code. Offline binary analysis is the process for doing this. We use *objdump* tool to disassemble the guest OS binary first, then extract the start addresses of all basic blocks in the binary. As shown in Fig. 3, we generate a valid target map that shows the valid indirect branch target address as 1 bit for each byte address of each code.

4.5 Runtime Integrity Enforcement

In VM-CFI, we used Intel PT to extract the indirect branch target address that occurred during execution of the guest kernel for integrity verification. At this time, we implemented a packet decoder to extract and analyze only the packets required for integrity verification among various kinds of packets coming from Intel PT. And the integrity checker verifies the integrity with the indirect branch target address extracted by the packet decoder.

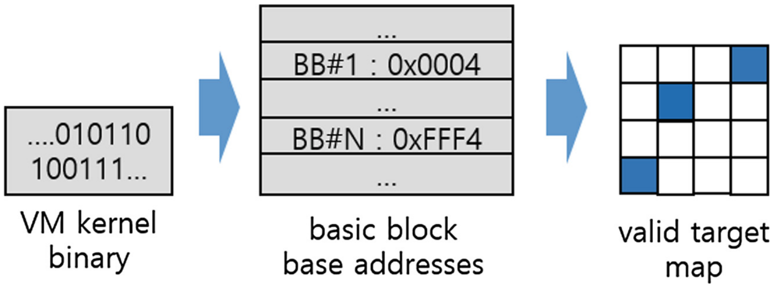

Fig. 3. Offline binary analysis

Packet Decoder. The packet decoder uses a privilege-level filter among the various filter functions provided by Intel PT. In VM-CFI, packets of guest or host user applications (programs executed in Ring 3) can be excluded. In addition, timing information and power events, which are not related to integrity verification, are set so that packets are not generated.

For the extracted packets, the packet decoder implements two functions to select only the information needed for integrity verification. First, the packet decoder distinguishes packets for the host kernel and the guest kernel. Fortunately, Intel PT knows the root/non-root mode through the PIP packet at the VM-entry/VM-exit, so that the packet decoder can distinguish it. Besides this method, there is a alternative method to enable trace packet generation in virtual machine only by using MSR load list in VM entry and VM exit respectively. However, VM-CFI does not use this method because it can only use Intel PT in non-root mode. Therefore, in VM-CFI, we design the packet decoder to identify the packet of the guest kernel. Next, in the packet decoder, only the TIP packet containing the target address information of the indirect branch among the control-flow related packets and the FUP containing the source address information are extracted and hand over to the integrity checker.

Integrity Checker. The Integrity Checker verified the integrity through the control information delivered by the packet decoder as shown in Fig. 4. We want VM-CFI to check two things: whether (1) the target address is in the valid target map (2) the address which the interrupt handler jumps from is the same as the address returned from interrupt hadler. First, if the packet transmitted by the packet decoder is a FUP packet (not a TIP packet), the corresponding source address is pushed onto the async-event stack, and the next packet is processed. If the packet is a TIP packet, check the corresponding bit of the valid target map for the target address to determine whether or not the base address of the basic block is valid. If the bit value is 1, the next packet is processed since it is a branch of the normal control-flow. If the bit value is 0, it confirms whether the control-flow branch is caused by the asynchronous event. That is, compare the top element of the async-event stack with the target address. To deal with the case where the target address points to the next instruction of the address indicated by the top element, the case is treated as a normal case when the difference is within 0xf, and then the corresponding value is popped on the stack before the next packet is processed. If it does not correspond to the top element, it is regarded as an attack.

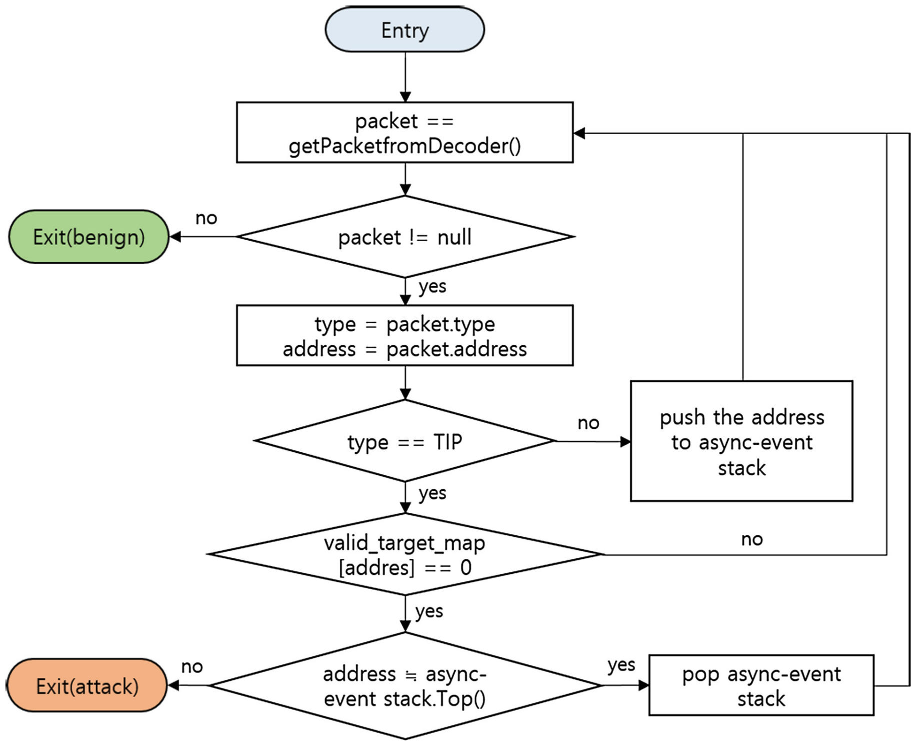

Fig. 4. Work flow of integrity checker

5 Evaluation

In this chapter, we will evaluate performance overhead due to VM-CFI and whether it can detect control-flow hijacking attacks targeting guest OS.

5.1 Experiment Environment

We have implemented a prototype of VM-CFI in real physical machine. The used machine has specifications of 4 Intel Core i5-6600 CPU @ 3.30 GHz and 4 GB RAM, and both the host kernel and the guest kernel use version 4.8.0 of linux. We used KVM as the hypervisor.

5.2 Prototype Implementation

As previously mentioned in the design, VM-CFI is implemented in the host kernel application without modifying the kernel. The binary analyzer is implemented by parsing the output file obtained by dumping the kernel binary with objdump, and implemented with 98 SLoC. The packet decoder and integrity checker are implemented by modifying the perf tool, which is provided for performance measurement and debugging in linux, and the number of modified lines is 244 SLoC. As such, VM-CFI guarantees virtual machine integrity by only writing a small amount of code in the user application without modification of the host and guest kernels.

5.3 Performance Evaluation

In order to measure the performance overhead of VM-CFI, we conducted two experiments. First, we measured the execution time of VM-CFI to analyze the control-flow of the virtual machine and to verify the integrity. Next we conduct an application benchmark experiment to conform that there is no performance overhead of the virtual machine despite the operation of VM-CFI.

Micro-benchmarks. In order to measure the time taken for VM-CFI to analyze the control-flow information and verify the integrity of the virtual machine during the execution of the virtual machine, we conducted a performance measurement experiment for the following cases.

- async-event-only: Validate only the async-event through the virtual machine's control-flow packet
- decode-only: Parse the control-flow packets of the virtual machine in VM-CFI
- vm-cfi: Parse the control-flow packet of the virtual machine through VM-CFI and verify integrity.

In the experiment, the virtual machine executes a user application that calls various system calls, and analyzes the resulting packets. As a result, it is confirmed that it takes about 10 s to process the packet of 20 MB in Fig. 5.

Application-Benchmarks. The following experiment was designed to measure the performance overhead on the virtual machine by VM-CFI. First, the virtual machine to be monitored is executed in core 0, and VM-CFI is executed in core 1. (This assumption is identical to the one used in previous Intel PT-based studies [9, 10].) Table 2 compares the performance of VM-CFI with the performance of Unixbench in the virtual machine, and summarizes the performance overhead. The result was an average performance overhead of 4.2%. This overhead is caused by memory virtualization during VM-CFI operation because memory is a computing resource used by VM-CFI and virtual machine simultaneously.

5.4 Security Evaluation

VM-CFI is designed to guarantee CFI of the guest OS. In this chapter, we have actually launched a rootkit attack which manipulate control-flow of guest OS. The rootkit attack modifies the data structure related to the virtual file system (VFS) managed by the guest OS. Specifically it modifies the function pointer associated with the file operation of the specific file. When the function pointer used later, the attack code injected by the attacker is executed instead of the normal file operation function. As a result, VM-CFI analyzes the indirect call packet generated at the moment when the attack code is executed by referring the manipulated function pointer, and confirms that it is not in the valid target map.

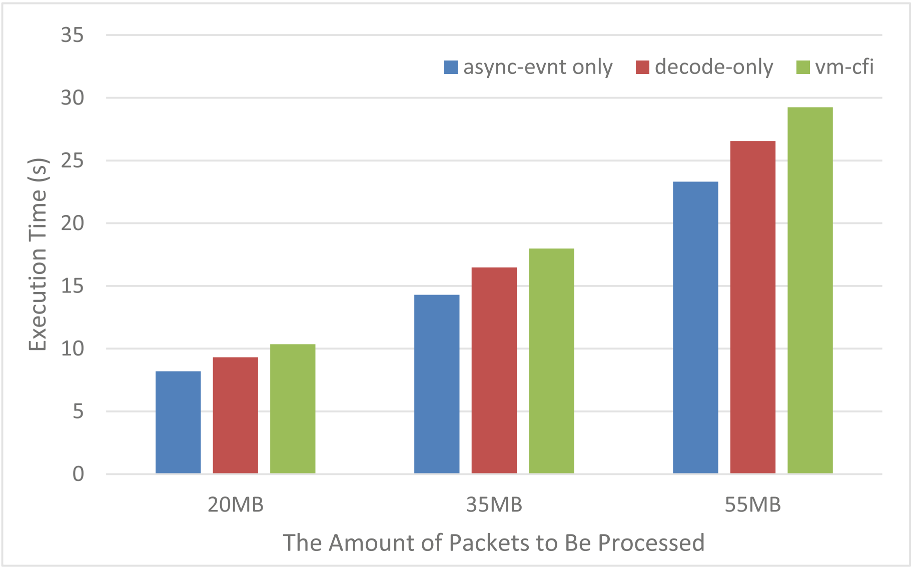

Fig. 5. Micro-benchmark experimental results. The horizontal axis indicates the amount of packets to be processed, and the vertical axis indicates the execution time (s).

Table 2. Unixbench performance overhead of virtual machines

Benchmark	Performance overhead
hanoi	2.5%
arith	2.5%
int	2.7%
syscall	7.8%
dhry	5.8%
context1	2.6%
spawn	5.4%
average	4.2%

6 Related Work

6.1 Virtual Machine Introspection

VMI researches have been examined to verify the state of each virtual machine in the cloud environment [2, 3]. These studies basically use virtualization technology or binary modification technology to monitor the hardware resources used by the virtual machine, or extract information about various events that occur when the virtual machine is running. However, there are no studies to efficiently extract the control-flow information of the guest OS. VM-CFI efficiently extracts and processes this information using Intel PT to ensure CFI for the guest OS.

6.2 Control-Flow Integrity Using Intel PT

Recently, Intel PT has been able to efficiently extract new execution flow information, and studies have been made to maintain execution flow integrity by using it [9–11]. However, all of these studies are different from run-time integrity studies for user applications, in that VM-CFI targets guest OS in the cloud environment. Specifically, there is a difference in processing Intel PT packets due to asynchronous events.

6.3 Kernel Control-Flow Integrity

KCoFI [4] compiles the kernel into a virtual instruction in a virtualized environment called Secure Virtual Architecture, and then verifies CFI in the process of converting the virtual command to the actual command in the virtual machine manager during kernel operation. Ge et al. [5] is implemented by inserting code that verifies CFI of each branch instruction of the kernel. At this time, kernel code modification is also implemented to protect the table storing the legitimate destination address from the attacker. On the other hand, krx [6] blocked the kernel code from being readable and defended against the Just-In-Time Code Reuse Attack for the kernel by arbitrarily randomizing the addresses of the codes. There is also a study to verify CFI of the guest OS in units of memory pages [12]. Although the previous technique has something in common in that it verifies CFI for the kernel, there is a limitation in practically applying all of the kernel code to recompile or modify a lot of parts. However, VM-CFI ensures CFI for the guest OS without these modifications.

7 Future Work

Our concern about VM-CFI is that the packet buffer could become full before integrity checks executed by VM-CFI are done. In this case, VM-CFI can only either protect guest OS in real time, or check all the packets generated by Intel PT. Even though this case did not occur in our experiment, the possibility still remains. We expect this problem to be solved by using multi-processor or GPU in decoding packets and integrity check, and it remains as the future work.

8 Conclusion

VM-CFI is a Control-Flow Integrity study for guest OS. Compared to kernel CFI studies with modifications to existing kernel code, VM-CFI uses hardware features called Intel PT to verify CFI of the virtual machine with a 4.2% lower overhead without modifying kernel code.

Acknowledgements. We thank anonymous reviewers for the support and insightful remarks that improved the paper. This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No. 2016-0-00078, Cloud based Security Intelligence Technology Development for the Customized Security Service Provisioning) and by the National Research Foundation of Korea

(NRF) grant funded by the Korea government (MSIP) (NRF-2017R1A2A1A17069478). This research was also supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2017-2015-0-00403) supervised by the IITP (Institute for Information & communications Technology Promotion).

References

1. Samarati, P., di Vimercati, S.D.C., Murugesan, S., Bojanova, I.: Cloud security: issues and concerns. Encycl. Cloud Comput. 1–14 (2016)
2. Zeng, J., Fu, Y., Lin, Z.: Pemu: a pin highly compatible out-of-vm dynamic binary instrumentation framework. In: ACM SIGPLAN Notices, vol. 50, pp. 147–160. ACM (2015)
3. Xiong, H., Liu, Z., Xu, W., Jiao, S.: Libvmi: a library for bridging the semantic gap between guest os and VMM. In: 2012 IEEE 12th International Conference on Computer and Information Technology (CIT), pp. 549–556. IEEE (2012)
4. Criswell, J., Dautenhahn, N., Adve, V.: KCoFI: complete control-flow integrity for commodity operating system kernels. In: 2014 IEEE Symposium on Security and Privacy (SP), pp. 292–307. IEEE (2014)
5. Ge, X., Talele, N., Payer, M., Jaeger, T.: Fine-grained control-flow integrity for kernel software. In: 2016 IEEE European Symposium on Security and Privacy (EuroS&P), pp. 179–194. IEEE (2016)
6. Pomonis, M., Petsios, T., Keromytis, A.D., Polychronakis, M., Kemerlis, V.P.: kR^ X: comprehensive kernel protection against just-in-time code reuse. In: Proceedings of the Twelfth European Conference on Computer Systems, pp. 420–436. ACM (2017)
7. Seshadri, A., Luk, M., Qu, N., Perrig, A.: SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes. In: ACM SIGOPS Operating Systems Review, vol. 41, pp. 335–350. ACM (2007)
8. Guide, P.: Intel® 64 and IA-32 Architectures Software Developer's Manual. Volume 3B: System programming Guide, Part 2 (2011)
9. Ge, X., Cui, W., Jaeger, T.: GRIFFIN: guarding control flows using intel processor trace. In: Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 585–598. ACM (2017)
10. Gu, Y., Zhao, Q., Zhang, Y., Lin, Z.: PT-CFI: transparent backward-edge control flow violation detection using intel processor trace. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, pp. 173–184. ACM (2017)
11. Liu, Y., Shi, P., Wang, X., Chen, H., Zang, B., Guan, H.: Transparent and efficient CFI enforcement with intel processor trace. In: 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 529–540. IEEE (2017)
12. Zhan, D., Ye, L., Fang, B., Zhang, H., Du, X.: Checking virtual machine kernel control-flow integrity using a page-level dynamic tracing approach. Soft Comput. 1–11 (2017)