http://dx.doi.org/10.5573/JSTS.2015.15.1.048 JOURNAL OF SEMICONDUCTOR TECHNOLOGY AND SCIENCE, VOL.15, NO.1, FEBRUARY, 2015

Efficient Kernel Integrity Monitor Design for Commodity Mobile Application Processors

Ingoo Heo[1], Daehee Jang[2], Hyungon Moon[1], Hansu Cho[3], Seungwook Lee[3], Brent Byunghoon Kang[2], and Yunheung Paek[1]

Abstract—In recent years, there are increasing threats of rootkits that undermine the integrity of a system by manipulating OS kernel. To cope with the rootkits, in Vigilare, the snoop-based monitoring which snoops the memory traffics of the host system was proposed. Although the previous work shows its detection capability and negligible performance loss, the problem is that the proposed design is not acceptable in recent commodity mobile application processors (APs) which have become de facto the standard computing platforms of smart devices. To mend this problem and adopt the idea of snoop-based monitoring in commercial products, in this paper, we propose a snoop-based monitor design called S-Mon, which is designed for the AP platforms. In designing S-Mon, we especially consider two design constraints in the APs which were not addressed in Vigilare; the unified memory model and the crossbar switch interconnect. Taking into account those, we derive a more realistic architecture for the snoop-based monitoring and a new hardware module, called the region controller, is also proposed. In our experiments on a simulation framework modeling a production-quality device, it is shown that our S-Mon can detect the rootkit attacks while the runtime overhead is also negligible.

Manuscript received Feb. 24, 2014; accepted Nov. 24, 2014

[1] Department of Electrical and Computer Engineering, Seoul National University
[2] Graduate School of Information Security, Korea Advanced Institute of Science & Technology
[3] DMC R&D Center, Samsung Electronics Ltd.
E-mail : Corresponding to ypaek@snu.ac.kr, brentkang@kaist.ac.kr

***Index Terms*—Security, application processor, smart mobile device, snoop-based integrity monitoring**

I. INTRODUCTION

As mobile devices including smartphones and tablets continue to gain popularity among the general public, they become our main devices for everyday communication such as emailing, social networking, processing financial information and transmitting personal data. Thus, the potential privacy and security risks associated with using these devices are also rapidly growing. Although many security solutions have been developed to protect the devices from the threats, unfortunately, most of them suffer from the *rootkit* attacks, which are the most threatening and common type of kernel-level malware that take privileges of operating system (OS) kernel to intercept and modify system events with the goal of hiding illicit activity [1]. Since rootkits themselves are located in the lowest kernel layer that has the highest privilege level in a system, they can trick any anti-malware solutions relying on the kernel, making them ineffective [11]. In other words, after rootkits compromise the OS kernel, any malicious behavior cannot be detected by the integrity monitors that have their root of trust on OS kernel.

To protect the OS kernel integrity from the rootkits, many security researchers have strived to make their security monitors independent from the host system that is being monitored. By separating the execution environment of monitoring from the host, the integrity of monitor could be protected from the rootkits. Recent efforts on this kernel integrity monitoring can be categorized into two groups: hardware-based approaches [2, 3] and hypervisor-based approaches [4, 5]. Although the hypervisor-based ones become popular, they have a critical weakness in that software vulnerabilities in them can be exploited by malwares to compromise the entire system. In the hardware-based approaches, they make an effort to get over the limitation with an independent hardware monitor that is inherently not reachable from the host. Since rootkit attacks running on the host cannot access the monitors, the hardware-based approaches are more secure than the conventional software-based solutions [6].

Most of the existing hardware-based solutions make use of *snapshot* analysis; they are usually assisted by some type of hardware component that enables saving of the memory contents into a snapshot, and then perform an analysis to find the traces of rootkit attack [2, 6, 7]. Nevertheless, the most critical weakness of the scheme is that it can inspect only the snapshots collected at a specific point of time, missing the evanescent changes which are not exposed in the captured snapshots. Thus, *transient* attack, which refers to attacks that do not leave persistent traces in memory contents but still achieves its goal by using only momentary and transitory manipulations [8], easily subverts the snapshot-based monitors. Moreover, the frequent snapshot-taking to increase the detection probability would inevitably degrade the host performance, because it consumes more memory bandwidth.

To overcome the limitations, Moon et al. [8] propose Vigilare, a kernel integrity monitor that makes use of snooping techniques for detecting the transient attack with low performance overhead. In order to achieve their goals, they take a fundamentally different approach; instead of taking snapshots, Vigilare monitors the operation of the host system by “snooping” the bus traffic of the host system from a separate independent system located outside the host system. This provides the monitoring system with the capability to observe all activities of the host which are revealed to the system’s shared bus, and yet being completely independent from any potential compromise or attacks in the host system. They demonstrates the effectiveness of this scheme by showing that their monitor is capable of effectively coping with transient attacks that violate the integrity of the immutable regions of the OS kernel and incurs negligible performance overhead on the host.

However, although Vigilare suggested a promising solution for rootkit detection, their prototype designs are too primitive to be deployed in recent commodity smart devices. In application processor (AP) platforms which are de facto the standard computing platforms for smart devices, there are many design rules to be complied with to manufacture the overall SoC system. Nevertheless, unfortunately, the monitor design proposed in Vigilare do not follow the general design constrainsts of recent APs, such as unified memory model or crossbar switch interconnect. Since the design restrictions are not considered, the previous monitor design of Vigilare cannot be directly applied to recent commodity products. Therefore, in order to leverage the deployment of the snoop-based monitoring in real machines, it is essential to consider a realistic design for commodity smart devices.

In this paper, we present a snoop-based integrity monitor, called S-Mon, which targets commodity smart devices. Unlike Vigilare, our monitor design strictly follows the design constraints on mobile AP platforms, in order to suggest the most realistic monitor design for commercial products. To derive a practical and realistic monitor architecture, the design constraints of recent commodity APs such as the unified memory model and crossbar switch interconnect are considered in designing S-Mon. Taking into account those, the architecture of snoop-based integrity monitor is restructured substantially, and a hardware module for secure monitoring in the unified memory model, called region controller, is also newly proposed. In our experiments on a simulation environment that models a realistic modern AP platform, S-Mon shows negligible performance impact on the host system, while transient rootkit attacks are successfully detected with our monitor design.

The rest of this paper is organized as follows. Section 2 introduces some backgrounds for OS kernel and the attack model which S-Mon targets for. After Section 3 introduces the design of Vigilare and its limitations, Section 4 will give the detailed design of S-Mon which considers commodity APs. Section 4 will report the experimental results and we will conclude this paper in Section 5.

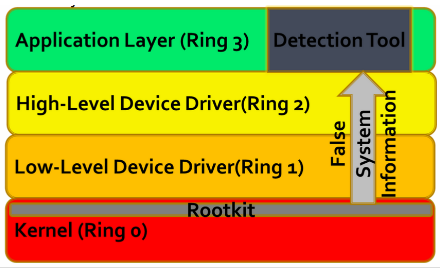

Application Layer (Ring 3)
Detection Tool
High-Level Device Driver(Ring 2)
Low-Level Device Driver(Ring 1)
False System Information
Rootkit
Kernel (Ring 0)

Fig. 1. OS Kernel Layers and Rootkit.

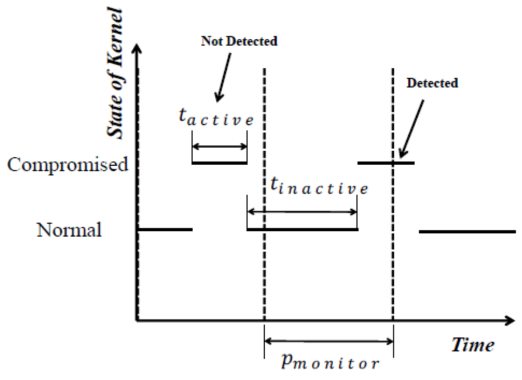


Fig. 2. The behavior of transient attack [8].

II. Attack Model

In this section, we will first explain several backgrounds which are necessary to understand our work. In particular, for some readers who are not familiar with system security, we will introduce a set of issues that are deeply related to the OS kernel integrity and the rootkit attacks. After that, we will present an attack model for this study, which aims at subverting the integrity of Linux kernel. Since most mobile OSes are based on the Linux kernel, we made our attack model be applicable to a majority of smart devices.

1. Rootkits and Transient Attack

As discussed in the previous section briefly, rootkit is a stealthy type of software that is designed to hide the existence of certain processes or programs from normal methods of detection and enable continued privileged access to a system [9]. As shown in Fig. 1, the rootkit is located in the lowest level of software layers and thus have the highest privilege to control the whole system. As a result, once installed, rootkits can modify the host's software to provide an attacker with the ability to hide the existence of chosen processes, files, and network connections from other users [2]. Therefore, rootkit can deceive anti-malware software by providing falsified information, and conventional anti-malware software that is dependable on the OS kernel's integrity never detects the malicious behaviors.

To get over the limitation, many security researches have tried to make their security monitors independent from the host system. With the separation from the host, rootkits cannot compromise the security monitors. Moreover, the existence of rootkit can be detected with the snapshot-based monitoring scheme when the rootkit leave permanent changes on the host.

However, transient attack which do not leave persistent traces in memory contents, still achieves its goal by using only momentary and transitory manipulations [8]. In such scenarios, the evidence of malicious modification is visible for a short time period so that detecting the modification becomes difficult. The soft-timer based rootkit presented by J.Wei et.al. is a representative example [1].

Fig. 2 shows the difficulties of detecting transient attacks with the snapshot-based monitoring. Imagine a snapshot-based integrity monitor with a snapshot period $P_{monitor}$ is launched to detect transient attacks. If the author of the kernel rootkit can properly adjust the duration of the attack t_{active} and the time of dormancy $t_{inactive}$, he could completely evade the snapshot-based monitors; by staying dormant at the time of memory snapshot and becoming active in between the snapshots, the rootkit can capably fool the snapshot-based monitor. The simplest solution to the limitation is to increase the rate of snapshot-taking but it will inevitably impose a high performance overhead on the host. Random snapshot timing is also not a proper solution because the detection rate would greatly depend on luck and not be consistent. In conclusion, the conventional snapshot-based integrity monitors cannot cope with the transient attacks and it is why the snoop-based monitors such as Vigialre were proposed.

2. Immutable Regions of Linux Kernel

Immutable region is the memory regions that are critical to the OS kernel integrity and thus any modifications on the regions are deemed malicious [8].

Protecting the integrity of the immutable regions should be the highest priority, since modification to the regions in the attacker's favor would be the most critical because the region constitutes core components in the OS and any compromise in this region would seriously affect all the application running on top of the OS.

For example, the immutable region of Linux OS includes the kernel code region and the system call table. The kernel code region is the most obvious example of the immutable region; since the basic functionalities of the kernel must not be changed after the bootstrap, the kernel code region should never be modified at runtime. The system call table is also a representative example of the immutable region because this table should not be modified after boot in order to provide many services consistently. Hijacking the kernel's system call often serves as an efficient way to control the kernel in the favor of an attacker. Modifying the system call table of the Linux kernel is a popular way to intercept the execution flow of the victimized system. The Linux system call table takes a form of an array of pointers. Each entry in the table points to a corresponding system calls such as *sys_read*, *sys_write*, and many more. The adversary could effortlessly hijack these system calls by inserting a function between the system call table and the actual system call handlers. Most user mode applications as well as kernel mode ones, rely on the basic system calls to communicate with file system, networking, process information, and other functionalities. Therefore, taking control of the system call table enables one to control the entire kernel from the bottom. For these reasons, it is important to focus on monitoring the immutable regions of the OS kernel. Thus, the attack model and corresponding monitor discussed in this paper also target the immutable regions of the OS kernel, as Vigilare did.

3. Assumptions and Attack Example

In this study, we assume that the host system is already compromised by an attacker with a rootkit attack. Therefore, the attacker already has the administrator's privilege on the host system and the memory protection of OS kernel is circumvented. However, the attacker's software cannot modify the hardware of the system and thus all hardware modules including our monitor are secure.

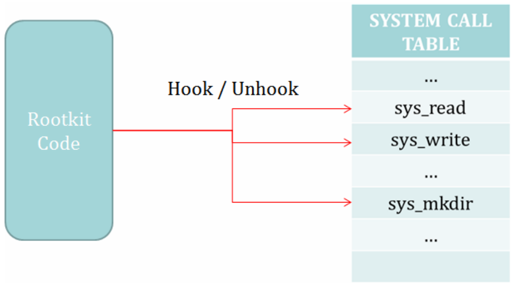


Fig. 3. Rootkit example (systemcall table hooking).

In these assumptions, to test our snoop-based integrity monitor (i.e., S-Mon), a transient rootkit attack example which hit the immutable region of Linux kernel is devised. The rootkit is very similar to the rootkit proposed in Vigilare [8] and acts as the traditional Linux kernel rootkits in the wild.

The rootkit repeatedly modify and reverse system call function pointers in the system call table to hijack the control flow of the system call, as shown in Fig. 3. To evade snapshot-based integrity monitoring, the rootkit repeat its function in a fixed time interval using the Linux timer as in Fig. 1. Particularly, the example rootkit hijacks *sys_read*, *sys_write* and *sys_mkdir* system calls to disturb standard IO or file system. As a side effect of the rootkit, the host system occasionally fails to handle standard IO and file system properly. The example rootkit simply performs the old-fashioned system call hooking, but the timer-triggered operation allows it to illustrate the transient rootkit characteristics. As shown in the experimental results, the capability of detecting the transient attack is the key difference of the snoop-based monitoring from the snapshot-based one.

III. ARCHITECTURE DESIGN OF VIGILARE AND CONSIDERATIONS FOR AP PLATFORMS

In this section, we will introduce the architecture design of Vigilare which is the first snoop-based monitor for the OS kernel integrity. Then, we will explain several design constraints of commodity AP platforms that should be complied with in order to derive a practical monitor in these platforms but are violated in Vigilare.

1. Vigilare Architecture Design

Fig. 4 shows the prototype design of Vigilare which

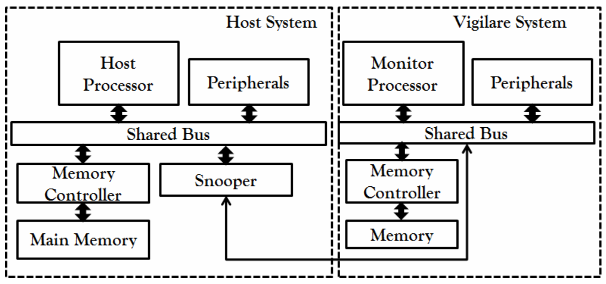


Fig. 4. Architecture design of Vigilare [8].

performs snoop-based monitoring. The overall system mainly consists of two subsystem, the host system being monitored and the Vigilare system that performs the monitoring. The Vigilare system is inherently separated from the host system so that any malicious access of malwares cannot be reachable to the monitor. As the figure shows, a dedicated memory module for the monitoring system is deployed and it contains all the programs and data used by the monitor processor. By using the separate memory for the monitoring system and memory controller inaccessible from the host system, the dedicated memory becomes physically tamper-free. In addition, since the host system cannot access any peripheral of the monitoring system, this configuration makes the monitoring system be secure from any potential compromise or attacks in the host system.

As shown in Fig. 4, a hardware module called the *snooper* is attached to the host system to detect malicious memory accesses on the host shared bus. In order to detect rootkit's behaviors that modify the immutable region of the OS kernel, Vigilare inspects all writes accesses revealed on the bus and check whether each write address is in the range of the immutable region. Since the snooper is connected to the host bus as a slave device, it can snoop all transactions on the bus because they are broadcasted to all components on the shared bus. By employing the snooping technique, Vigilare can detect all transient rootkit attack with negligible overhead while it is still secure from the rootkit's threat running on the host.

2. Design Constraints for AP Platforms

In AP platforms which are de facto the standard platforms for most smart devices, several design constraints should be followed for efficient design and rapid manufacturing. However, in the architecture design of Vigilare above, two design constraints for the platforms are violated; *unified memory model* and *crossbar switch interconnect*.

Firstly, the Vigilare system assumes a dedicated memory model that a dedicated memory module for the monitoring software code and data is given. Unfortunately, the memory model is generally not acceptable in AP platforms. In general, many commercial AP platforms contain a limited number of DDR DRAM modules as a main memory due to their expensive costs. Therefore, although the required memory capacity is small, requiring the additional memory module would hamper AP vendors to adopt the monitoring hardware. Thus, the memory space for the newly installed monitoring system should be allocated from the existing memory modules and it would consequently forms a unified memory model between the host and the monitoring system. Therefore, in order to adopt the snoop-based monitoring into AP platforms, this unified memory model should be employed.

Secondly, Vigilare assumes that the host processor is connected to the main memory via a shared bus such as AMBA AHB bus. In most commercial APs, however, the host processor and the main memory are connected through crossbar switch instead of shared bus [17]. Shared bus is rather old-fashioned interconnect that all components shares a communication channel and thus it might incur frequent bus contention when multiple transactions want to use the shared channel. On the other hand, crossbar interconnect provides multiple channels between masters and slaves in a matrix manner and therefore it can reduce the communication overhead significantly. It is the reason why the crossbar switch is widely used in modern AP platforms and thus the design of the snoop-based monitor should consider the interconnect type to be deployed in the realistic platforms. In the next section, we will discuss how these design constraints are considered in our S-Mon.

IV. S-MON : A PRACTICAL MONITOR DESIGN FOR AP PLATFORMS

In this section, the design of S-Mon, which is a snoop-based monitor that considers commodity AP platforms, will be introduced. After the overall design is shown in

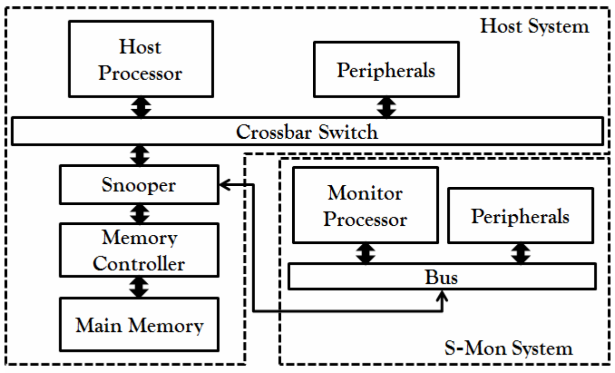


Fig. 5. The proposed design of S-Mon.

Section 4.1, the further detailed design considerations for modern AP platforms will be given in Section 4.2.

1. Overall System Design

The overall system design with S-Mon is presented in Fig. 5. As the figure shows, the overall system consists of the host system being monitored and S-Mon system that observes the host activities to detect malicious rootkit attacks. For the host system, an ARM Cortex-A9 dual processor [12] is used for the host processor and an ARM NIC-301 AXI crossbar switch interconnect [13] is integrated to connect the host processor, memory controller and other peripherals. It is assumed that the host processor runs at 1 GHz and has separate 32 KB instruction/data caches. The host system uses 1 GB DDR3 SDRAM as a main memory and has some peripherals such as UART and timer. The NIC-301 and DDR3 SDRAM commonly operate at 500 MHz. The host runs the OS kernel which is an embedded Linux kernel 2.6.38. It is noteworthy that the specifications for the host are very similar with the recent AP platforms such as Samsung Exynos 4 [14].

The design of S-Mon system is derived from Vigilare's prototype [8] which is described in the previous section. Fig. 5 shows that S-Mon is implemented as a separate system that consists of a monitor processor, the snooper and other peripherals such as UART. The monitor processor is also an ARM Cortex-A9 processor and has 32 KB instruction/data caches, running at 1 GHz. Unlike Vigilare, S-Mon system does not have its own dedicated memory module and thus a small section of main memory module is allocated for monitoring software, in order to reduce the production costs.

With the given hardware components, S-Mon also performs snoop-based monitoring that snoops the write memory traffic to find malicious behaviors of rootkits which hit the immutable regions of the OS kernel. To achieve this goal, the snooper module is located between the crossbar switch system interconnect and the memory controller. For each write request from the host, the snooper inspects the physical address of the transaction and check whether it is in the range of the addresses which corresponds to the immutable region. If it is, the write attempt is regarded as malicious behavior of rootkits.

After the snooper finds any attempt to corrupt the immutable region, it is reported to the monitor processor through the S-Mon system bus. In the current prototype, if the existence of rootkit is detected, it is announced via the UART interface in S-Mon system. Since the output interface is not reachable from the host, the reporting process is also secure and trustworthy. In the next subsection, the detailed design considerations of S-Mon, which is for adopting the snoop-based monitor to the recent AP platforms, will be discussed.

2. Design Considerations for AP Platforms

Although the basic functionality of S-Mon is the same as Vigilare, the detailed architecture for S-Mon differs from the Vigilare's prototype because the newly proposed design seriously considers realistic AP design environments as opposed to the previous work. As described in the previous section, there are two design restrictions that should be complied with to adopt the snoop-based monitor in AP platforms. The first is the unified memory model that the host and the monitoring system share the same memory module. The second is the crossbar switch interconnect which affects the location of the snooper. The following will discuss the detailed implementation issues.

As discussed earlier, for the AP design environments, the dedicated memory module for the monitoring system would not be acceptable due to the increase of production costs. For this reason, the proposed architecture in this paper adopts the unified memory model where the memory for monitoring is allocated from the main memory module. Nevertheless, although the memory

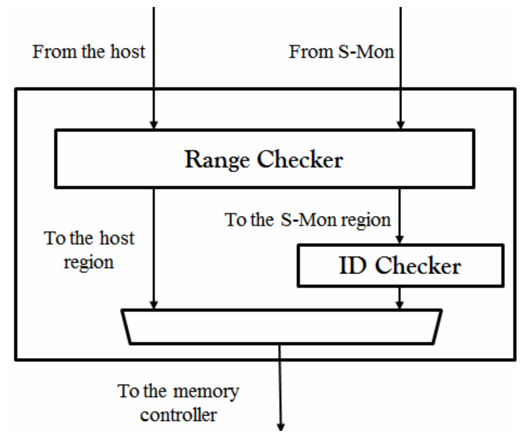


Fig. 6. The internal structure of region controller.

model meets the conventional AP design rules, it would make the monitoring system vulnerable. It is because the memory sections for S-Mon system reside in the main memory module and thus the host is now able to access the region to subvert the kernel integrity. Therefore, the memory region accessible from the host is no longer deemed as secure.

In order to keep the monitoring system secure from the rootkit attacks even when the unified memory model is employed, a hardware logic called the *region controller* is implemented in the snooper, as shown in Fig. 6. The region controller has been embedded in the snooper and specifically drops all memory operation requests from the host system to the memory region of S-Mon. At first, the *range checker* module checks whether each memory access address is in the range of S-Mon region. If it is not a transaction toward S-Mon region, the region controller passes it to the memory controller to access the host memory region. However, in the case of accesses to the S-Mon region, the transactions from the host should be denied. For this purpose, the region controller discriminate the requests of the host with transaction ID of AXI protocol, being supported by the *ID checker*. Since AXI interconnection protocol supports transaction IDs to identify the master of the transaction, the ID checker can classify the master of the memory access and thus drop the all accesses from the host. By using the region controller, the memory region of S-Mon is still physically temper-free and secure from the potential attacks of rootkits even in the unified memory model.

As shown in Fig. 4, the snooper hardware module in the prototype of Vigilare is connected to the main bus as a slave device observing all signals on the bus. By inspecting all write memory traffic on the bus, the snooper is able to detect any suspicious traffic that attempts to compromise the kernel. However, for a crossbar switch interconnect such as AXI NIC-301, the location of the snooper hardware module should be reconsidered. It is because communication traffics between two modules are not revealed to the other modules on the crossbar interconnect, which consequently would make the snooper hardware ineffective to detect the rootkit attacks.

To snoop the suspicious memory traffics on the crossbar switch interconnect, the snooper hardware of S-Mon is located at the front of the memory controller. The location of the snooper has an advantage in that, it does not depend on the interconnection types of the system. Because all memory requests are finally transferred to the memory, any malicious write transactions of the rootkits can be detected at the location. With the design consideration, the snooper hardware and the snoop-based monitoring scheme can be adoptable modern AP platforms where crossbar switch is usually used as main interconnect.

V. EXPERIMENTAL RESULTS

To evaluate the effectiveness of our S-Mon, we have developed a prototype on a simulation framework based on Carbon SoC Designer Plus [15]. This simulation tool provides cycle-accurate simulation and rapid prototyping for evaluating realistic AP platforms. The detailed specifications are the same as in Section 4. Commodity processors, interconnect and memory were given as library models that support cycle-accurate simulation for the corresponding modules. On the other hand, several customized hardware modules such as the snooper and the region controller had to be designed with Verilog HDL language and translated to simulation components with Carbon Model Studio [16]. By employing this accurate simulation framework, we have obtained reliable experimental results.

1. Performance

If a dedicated memory module is used for integrity monitoring as in Vigilare, this does not affect the performance of the host because there is no resource sharing between the host and the integrity monitor. In contrast, when the unified memory model is employed as in our S-Mon, the two system should share a memory module and it consequently incurs resource competition. For this reason, the performance impact of the unified model should be evaluated before adopting the snoop-based monitoring into AP platforms where the stringent performance requirements must be met.

To compare the performance impact of the memory models, both the unified memory model and the dedicated memory model are implemented. As described in the previous section, S-Mon is basically implemented with the unified memory model. For the dedicated memory model, an additional memory module is given to S-Mon and the monitoring software is allocated to the memory. Since the required memory capacity for S-Mon is under 1 MB, a 32 MB DDR memory is given for the dedicated memory model. To evaluate the performance impact on the host, the performance of the host system is measured with STREAM benchmark [10] which is widely used for measuring the memory bandwidth of a system. The data size for the experiments is 10000 and the iteration is 3. In addition, to better compare the snoop-based monitoring scheme with traditional snapshot-based method, a snapshot-based integrity monitor is also implemented. The snapshot monitor uses the DMA to get the snapshot of the immutable regions of the host kernel periodically and it is assumed that the region for snapshot is 4 KB.

Fig. 7 compares the performance of the host for six configurations. Three configurations for the snapshot monitors show that the snapshot-based monitoring scheme with shorter intervals slows down the host system significantly. Although the size of snapshot is minimized for brief comparison, the performance degradation is still substantial. Since the actual size of the immutable region is about 1 MB (we just assumed 4 KB), the slowdown must be worsened when the snapshot is performed on the entire immutable region. This provides a plausible reason why the snapshot-based monitoring is not promising in mobile devices.

In contrast, the snoop-based monitoring scheme brings the overhead down to negligible level. As expected, the dedicated memory model does not degrade the performance of the host at all. Meanwhile, the unified memory model shows 2.87% overhead compared to the

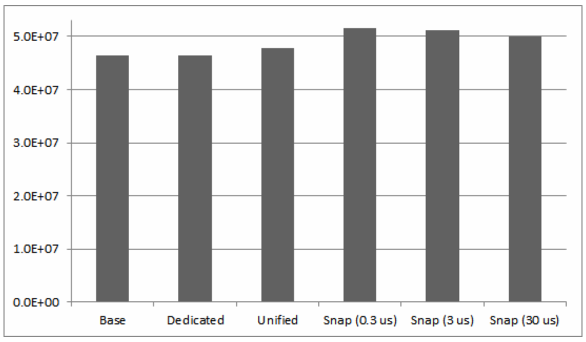


Fig. 7. Cycle counts for six configurations (Base : The host system without monitoring. Dedicated : S-Mon with the dedicated memory model. Unified : S-Mon with the unified memory model. Snap : Snapshot-based monitoring scheme and the numbers within parentheses indicate the period of the snapshot).

base case because the monitor processor requires the code and data in the main memory module. However, the number should be regarded as the worst case due to the characteristic of STREAM benchmark. Consider that most operations in STREAM benchmark require memory read/write so that the required memory bandwidth is maximized. In this case, the performance of the benchmark can be easily degraded by the small number of memory requests from the monitoring system. In fact, although the memory operations for monitoring are not so frequent and most of them are handled within the cache of the monitor processor, the performance of host is slightly slowed down. Moreover, since generally the mobile devices do not always require the peak memory bandwidth, the small 2.87% performance overhead can be amortized in real applications.

To justify our claim, we have measured the performance of the host when the unified memory model is employed, for eight applications from SPEC CPU 2006 benchmark [19]. Fig. 8 shows the performance comparison between S-Mon and the base case for the applications. This results show that, in most applications, the performance overhead of S-Mon with the unified memory model is negligible and it is about 0.8% in average. Therefore, we can conclude that the runtime performance overhead of S-Mon is acceptable in the realistic unified memory model.

2. Security Evaluation

Another objective of our experiments was to show the

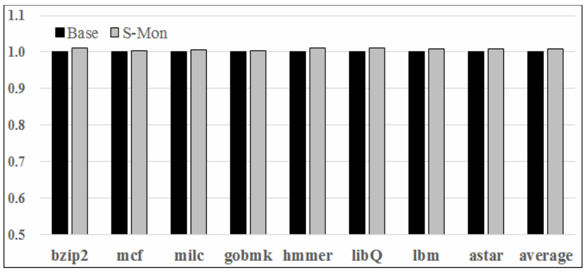


Fig. 8. The execution time of the host normalized to the base for eight applications from SPEC CPU 2006 benchmark.

detectability of our S-Mon on transient rootkit attacks. To demonstrate the ability to detect the attacks, the rootkit sample described in Section 2 was built and the probability of detecting a pulse of an attack is measured when S-Mon is employed. The rootkit example for this experiment is implemented to meet the definition of transient attack in Section 2 and Fig. 2 shows how the rootkit example works. It modifies the system call table of the Linux kernel to hook some of the system calls. After t_{active}, it removes its hooks by modifying the system call table as it has been before. After $t_{inactive}$, it hooks the system call as it did before. If the host system is compromised by this sample attack, it cannot service appropriate system calls related to user's standard I/O. In this experiment, 100 pulses are generated and how many of them were detected by S-Mon is measured. We varies the duration of the attack t_{active} from 2 ms to 50 ms, while the time of dormancy $t_{inactive}$ is changed from 10 ms to 1 s.

In this experiment, for all timing configurations, S-Mon detects all the pulses of attacks by snooping the write memory traffics to the system call table regions. However, the timings that S-Mon detects the rootkit behaviors are not consistent. The reason why S-Mon cannot detect all purse at the same timing is that the realistic design in this paper considers write-back cache type. In case of write-back cache, the write attempts from the host may not be seen immediately on the bus, because the cache does not commit all the memory updates immediately to the memory [8]. However, as Vigilare stated, because most caches write back dirty cache lines even when it is restored to original value, any write attempts on the immutable region generates the corresponding write traffic on the links between caches and memory, where the snooper snoops. The experimental results confirms that the snoop-based monitoring is still valid when the host system uses write-back cache system.

Table 1. Synthesis Results for Region Controller

Combinational Area (μm^2)	46.39
Buffer/Inverter Area (μm^2)	5.82
Non-Combinational Area (μm^2)	33.52
Total Area (μm^2)	85.73

3. Additional Overhead for S-Mon

In this paper, we propose the region controller to securely protect the memory region for S-Mon. To estimate the area overhead, we have measured the size of the newly proposed module with Synopsys Design Compiler [18] and a commercial 45 nm process library. Table 1 shows the estimated area for the region controller. As shown in the table, the region controller only occupies 85.73 μm^2 in 45 nm process, which is acceptable overhead in recent AP platforms.

When S-Mon is employed in an AP, the software code and data for monitoring should be given to the monitor processor. It includes the codes for the management of S-Mon such as the configuration of the snooper and the information related to the immutable regions. Although they occupy the additional memory space in the main memory, the required space is relatively small. In the current prototype, the additional memory space for the software code and data for S-Mon is about 273 KB. Since most commercial APs have a main memory module which can contain more than 1 GB, the additional space can be acceptable in most APs.

VI. CONCLUSIONS

This paper proposes S-Mon, a design of snoop-based kernel integrity monitor that considers commodity AP design environment. To adopt the snoop-based monitoring scheme to commodity APs, the unified memory model and crossbar switch interconnect are considered in designing our S-Mon. To this end, several design changes are discussed for realistic implementation and the resulting architecture is prototyped on a cycle-accurate simulation environment. In the experiments in this paper, it is demonstrated that S-Mon is capable of effectively coping with transient attacks that violate the integrity of the immutable regions of the OS kernel, while the performance degradation of S-Mon is still acceptable even though the unified memory model is imposed. As this paper shows, S-Mon is not only promising against threatening rootkit attacks, but also acceptably designed for modern smart devices manufactured with AP SoC platforms where practical design constraints exists.

ACKNOWLEDGMENTS

This work was partly supported by the Engineering Research Center of Excellence Program of Korea Ministry of Science and ICT & Future Planning (MSIP) / National Research Foundation of Korea (NRF) (Grant NRF-2008-0062609), the IT R&D program of MSIP/KEIT [K10047212], the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No. 2014R1A2A1A10051792) and the Brain Korea 21 Plus Project in 2014. Also, this research was supported by MOTIE (The Minister of Trade, Industry and Energy), Korea, under the Brain Scouting Program (HB609-12-3002) by the NIPA (National IT Promotion Agency) and Samsung DMC R&D Center (IO120903-05075-01).

REFERENCES

[1] J. Wei, B. Payne, J. Giffin, and C. Pu, “Soft-timer driven transient kernel control flow attacks and defense,” In Computer Security Applications Conference, 2008. ACSAC 2008. Annual, pages 97-107, dec.2008.USENIX Security Symposium.

[2] N. L. Petroni, Jr., T. Fraser, J. Molina, and W. A. Arbaugh, “Copilot - a coprocessor-based kernel runtime integrity monitor,” In Proceedings of the 13th conference on USENIX Security Symposium – Volume 13, SSYM'04, pages 13-13, Berkeley, CA, USA, 2004.USENIX Association

[3] X. Zhang, L. van Doorn, T. Jaeger, R. Perez, and R. Sailer, “Secure coprocessor-based intrusion detection,” In Proceedings of the 10th workshop on ACM SIGOPS European workshop, EW 10, pages 239-242, New York, NY, USA, 2002. ACM.

[4] T. Garfinkel and M. Rosenblum, “A virtual machine introspection based architecture for intrusion detection,” In Proceedings of Network and Distributed Systems Security Symposium, Feb 2003. Internet Society

[5] J. Rhee, R. Riley, D. Xu, and X. Jiang, “Defeating dynamic data kernel rootkit attacks via vmm-based guest-transparent monitoring,” In Availability, Reliability and Security, 2009. ARES '09. International Conference on, pages 74-81, march 2009. IEEE.

[6] A. M. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. C. Skalsky, “Hypersentry: enabling stealthy in-context measurement of hypervisor integrity,” In Proceedings of the 17th ACM conference on Computer and communications security, CCS '10, pages 38-49, New York, NY, USA, 2010. ACM.

[7] J. Wang, A. Stavrou, and A. Ghosh, “Hypercheck: A hardware-assisted integrity monitor,” In S. Jha, R. Sommer, and C. Kreibich, editors, Recent Advances in Intrusion Detection, volume 6307 of Lecture Notes in Computer Science, pages 158-177. Springer Berlin /Heidelberg, 2010.

[8] H. Moon, H. Lee, J. Lee, K. Kim, Y. Paek and Brent B. Kang, “Vigilare: toward snoop-based kernel integrity monitor,” Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 2012.

[9] Rootkits, part 1 of 3: A growing threat, April 2006. MacAfee AVERT Labs Whitepaper.

[10] J. D. McCalpin, “Memory bandwidth and machine balance in current high performance computers,” IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter, pages 19-25, Dec.1995.

[11] Lee, Hojoon, et al., “KI-Mon: A Hardware-assisted Event-triggered Monitoring Platform for Mutable Kernel Object,” Presented as part of the 22nd USENIX Security Symposium. USENIX, 2013.

[12] LTD ARM co., “a9 processor,” 2011.

[13] LTD ARM co., “AMBA Network Interconnect (NIC-301) Technical Reference Manual,” 2009.

[14] LTD Samsung Electronics co. Exynos 4, 2011, http://www.samsung.com/global/business/semiconductor/

[15] Carbon Design Systems, Carbon SoC Designer Plus., http://www.carbondesignsystems.com/soc-designer-plus

[16] Carbon Design Systems, Carbon Model Studio., http://www.carbondesignsystems.com/carbon-

model-studio/
[17] Na, Sangkwon, Sung Yang, and Chong-Min Kyung, "Low-power bus architecture composition for AMBA AXI," Journal of Semiconductor Technology and Science 9.2 (2009): 1.
[18] Synopsys, Inc., Synopsys Design Compiler, http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler/Pages/default.aspx
[19] J. L. Henning, "Spec cpu2006 benchmark descriptions," ACM SIGARCH Computer Architecture News, vol. 34, no. 4, pp. 1–17, 2006.

Ingoo Heo received a B.S. degree in Electrical Engineering from Seoul National University, Korea, in 2009. He is currently working towards the Ph.D degree in electrical and computer engineering from Seoul National University. His recent research interests are security hardware and data leak prevention using dynamic information flow tracking.

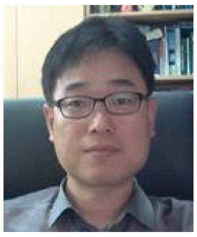
Daehee Jang received the B.E. degree in Computer Science from Hanyang University, South Korea, in 2011. He also received the M.E. degree at Graduation School of Information Security from Korea Advanced Institute of Science and Technology (KAIST), South Korea, in 2014. He is currently working toward the Ph.D. degree at the Graduation School of Information Security from Korea Advanced Institute of Science and Technology (KAIST), South Korea, since 2014. His main research interests include reverse engineering, rootkit analysis.

Hyungon Moon received B.S. degrees in Electrical Engineering and in Mathematical Science from Seoul National University, Korea, in 2010. He is currently working towards the Ph.D degree in electrical and computer engineering from Seoul National University. His research interests include designing operating systems and computer architectures for system security.

Hansu Cho is a senior engineer at Samsung DMC R&D Center. Dr. Cho has received his Ph. D. in Electrical Engineering and Computer Science from the University of California, Irvine. His research areas include system level design methodology, ESL simulation, High Level Synthesis, ASIP Design, and SoC Security.

Seungwook Lee is a principal engineer at Samsung DMC R&D Center. Dr. Lee has received his Ph. D. in Information & Communication Engineering from Sungkyunkwan University in 2006. He researched on secure processor architecture and security performance analysis for his thesis. His current research interests include system level design, SoC architecture optimization, high speed SoC simulation, SoC power and thermal simulation, ASIP Design and secure SoC architecture.

Brent Byunghoon Kang is currently an associate professor at the GSIS (Graduate School of Information Security) at KAIST (Korea Advanced Institute of Science & Technology). Before KAIST, he has been with George Mason University as an Associate Professor in the Volgenau School of Engineering. Dr. Kang received his Ph.D. in Computer Science from the University of California at Berkeley, and M.S. from the University of Maryland at College Park, and B.S. from Seoul National University. He has been working on systems security area including OS kernel integrity monitor, advanced botnet malware defense, server ghosting and DNS analytics. He is currently a member of the IEEE, the USENIX and the ACM.



Yunheung Paek received the B.S. and M.S. degrees in computer engineering from the Seoul National University, Korea in 1988 and 1990, respectively. He received his Ph.D. degree in computer science from University of Illinois at Urbana-Champaign in 1997. Currently he is a professor at the department of electrical and computer engineering, Seoul National University, Korea. His research interests include system security with hardware and hypervisor, secure processor design against various types of threats, and encryption hardware. He is also working on mobile cloud computing and re-targetable compiler.