IOCheck: A Framework to Enhance the Security of I/O Devices at Runtime

Fengwei Zhang

Center for Secure Information Systems
George Mason University
Fairfax, VA 22030
fzhang4@gmu.edu

***Abstract*—Securing hardware is the foundation for implementing a secure system. However, securing hardware devices remains an open research problem. In this paper, we present IOCheck, a framework to enhance the security of I/O devices at runtime. It leverages System Management Mode (SMM) to quickly check the integrity of I/O configurations and firmware. IOCheck does not rely on the operating system and is OS-agnostic. In our preliminary results, IOCheck takes 4 milliseconds to switch to SMM which introduces low performance overhead.**

Keywords*—*Integrity, Firmware, I/O Configurations, SMM

I. Introduction

With the increasing complexity of hardware devices, firmware functionality is expanding, exposing new vulnerabilities to attackers. The National Vulnerabilities Database (NVD [1]) shows that 119 firmware vulnerabilities have been found since 2010. An attacker can exploit these vulnerabilities in firmware [2] or a tool for updating firmware [3].

After compromising the firmware of an I/O device (e.g., NIC card), attackers alter memory via DMA [2], [4], [5] or compromise surrounding I/O devices [6], [7]. Fortunately, the Input Output Memory Management Unit (IOMMU) mechanism can protect the host memory from DMA attacks. It maps each I/O device to a specific area in the host memory and any invalid access fails. However, this mechanism relies on correct IOMMU configurations (e.g., the base address of the root entry table). These configuration registers and tables can be modified by an attacker as well [8].

A Trusted Platform Module (TPM [9]) can protect the integrity of firmware and IOMMU configurations while booting, but it cannot protect them at runtime. Recently, Intel introduced Trusted eXecution Technology (TXT), providing a trusted way to load and execute system software (e.g., OS or VMM). This is achieved by performing software measurements with the help of the TPM. Intel TXT does not make any assumptions about the system state, and it provides a dynamic root of trust. Thus, the TXT can be used to check the runtime integrity of I/O configurations and firmware. AMD has a similar technology called Secure Virtual Machine (SVM). However, both TXT and SVM introduce a significant overhead on the late launch operation (e.g., TPM quote in [10]).

In this paper, we present IOCheck, a framework to enhance the I/O devices' security at runtime. It leverages System Management Mode (SMM), a CPU mode in the x86 architecture, to quickly check the integrity of I/O configurations and firmware. Unlike previous systems [11], [12], IOCheck enumerates all of the I/O devices on the motherboard, and checks the integrity of their corresponding configurations and firmware. IOCheck does not rely on the operating system, which significantly reduces the Trust Computing Base (TCB). In addition, IOCheck is able to achieve better performance compared to TXT or SVM approaches. For example, the SMM switching time is much faster than the late launch method in Flicker [10], [13]. We will demonstrate that IOCheck is able to check the integrity of array of I/O devices including the BIOS, IOMMU, network card, video card, keyboard, and mouse.

Contributions. The purpose of this work is to make the following contributions:

- We provide a framework to enhance the security of I/O devices by checking their integrity at runtime.
- IOCheck is implemented in the hardware layer and does not rely on the operating system, which introduces a minimal TCB.
- IOCheck is able to enumerate all of the I/O devices on motherboard and check the integrity of each one.
- We demonstrate that our system can detect all of the tested firmware exploitations and I/O configuration modifications.
- We show that our system introduces low operating overhead.

II. Background

A. Computer Hardware Architecture

Figure 1 shows the hardware architecture of a typical computer. The Central Processing Unit (CPU) connects to the Northbridge via the Front-Side Bus. The Northbridge has Memory Management Unit (MMU) and IOMMU, collectively called the memory controller hub. The Northbridge also connects to the memory, graphics card and Southbridge. The Southbridge, also called the I/O controller hub, connects a variety of I/O devices including USB, SATA, and Super I/O, among others. Note that the BIOS is also connected to the Southbridge. IOCheck aims to check and verify the static configurations and firmware of I/O devices.

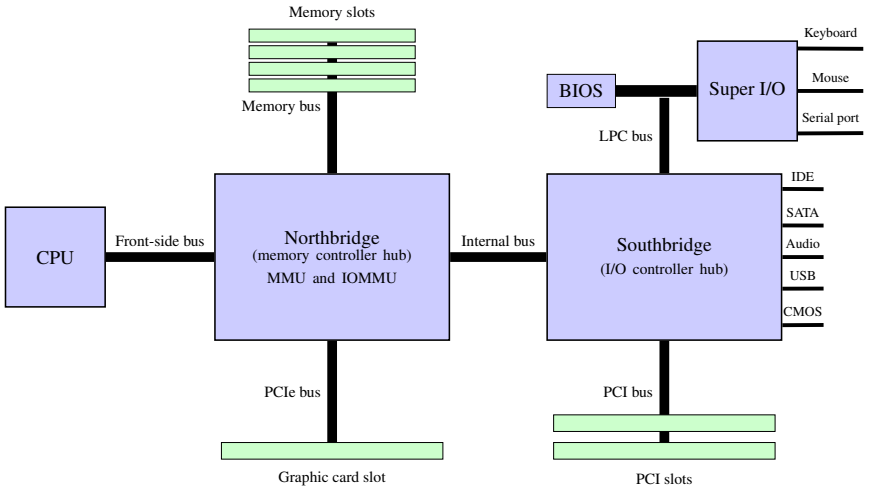

Fig. 1. Typical Hardware Layout of a Computer

B. System Management Mode

System Management Mode (SMM) is a CPU mode in the x86 architecture. It is similar to Real and Protected Modes. It provides an isolated execution environment for implementing system control functions such as power management.

SMM is implemented by the BIOS. Before the system boots up, the BIOS loads SMM code into System Management RAM (SMRAM), a special memory region which is inaccessible from any other CPU modes. SMM is triggered by asserting the System Management Interrupt (SMI) pin on the motherboard. Both hardware and software are able to assert this pin, although the specific method depends on the chipset. After assertion, the system automatically saves its CPU states into SMRAM, then executes the SMI handler code. A RSM instruction is executed at the end of the SMI hander to switch back to Protected Mode.

III. Related Work

To identify the malware running in I/O devices, Li et al. proposed VIPER [11], a software-based attestation method to verify the integrity of peripherals' firmware. VIPER runs a verifier program on the host machine, and it trusts the operating system. NAVIS [12] is an anomaly detection system checking the memory accesses performed by the NIC's on-chip processor. It builds a memory layout profile of the NIC, and raises an alert if any unexpected memory access is detected. NAVIS program runs inside of the operating system, and assumes the OS is trusted. Compared to VIPER and NAVIS, IOCheck is a hardware-based method for firmware integrity checking, which significantly reduces the TCB. In addition, IOCheck checks the configurations of I/O devices, which provides a further protection on the I/O devices.

Sang et al. discusses IOMMU vulnerabilities [8] and several I/O attacks [7] on Intel-based systems. Additionally, other researchers use an embedded microcontroller in the chipset to check the integrity of system memory [14], implement DMA malware [4], and introduce Ring -3 rootkits [15]. IOCheck is a framework to mitigate these attacks to enhance the security of I/O devices.

Recently, SMM-based systems have been brewing in the security area. HyperCheck [16], HyperSentry [17], SPECTRE [13] are SMM-based systems introspecting host memory for integrity checking or malware detection, while IOCheck aims to enhance the security of I/O devices. In addition, researchers use SMM to implement stealthy rootkits [18], which requires an unlocked SMRAM to load the rootkit into SMM. IOCheck locks the SMM in the BIOS so that SMRAM is inaccessible after system boots. Wojtczuk and Rutkowska [19] demonstrate an attack to bypass SMM lock using cache poisoning. It configures Memory Type Range Registers (MTRR) to force the CPU to execute code from cache instead of SMRAM. Fortunately, this architecture vulnerability has been fixed by the vendor.

IV. Threat Model and Assumptions

A. Threat Model

1) I/O Configuration Attacks: Recently, Intel introduced the virtualization technology for directed I/O (VT-d), which implements the IOMMU mechanism for Intel-based platforms. IOMMU is able to block DMA attacks by remapping the addresses accessed by hardware devices. However, an attacker can bypass IOMMU by manipulating configuration registers and tables [8]. Additionally, an attacker can also manipulate the PCI configuration space or Super I/O configurations for malicious purposes. IOCheck aims to check these configurations at runtime.

2) Firmware Attacks: The firmware of I/O devices contains vulnerabilities which can be exploited by attackers. For instance, Duflot et al. demonstrate that a remote attacker can compromise a Broadcom NIC firmware by sending crafted UDP packets [2]. Additionally, an attacker with ring 0 privilege can reflash the firmware to implement a persistent rootkit (e.g., BIOS rootkit). IOCheck is capable of verifying the runtime integrity of firmware.

B. Assumptions

IOCheck uses SMM to check the integrity of hardware configurations and firmware. An attacker is able to tamper

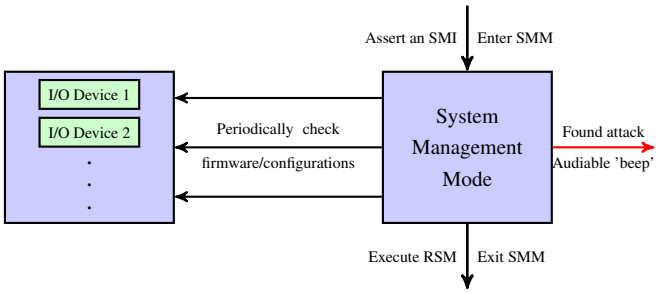

Fig. 2. Architecture of IOCheck

with the firmware by exploiting zero-day vulnerabilities. Since IOCheck does not rely on the operating system, we assume the attacker has ring 0 privilege. We assume the system is equipped with a TPM which measures the integrity of the initial BIOS image, which guarantees the SMM code is securely loaded into SMRAM and SMRAM is locked in the BIOS. Cache poisoning attacks [19] modifying the SMI handler is out of the scope of this paper. IOCheck does not consider Denial of Service (DoS) attack because an attacker with ring 0 privilege can block the SMI assertion. Furthermore, we assume the attacker does not have physical access to the machine.

V. System Framework

IOCheck is a framework that enhances the security of I/O devices at runtime, and it checks the static configurations and code of I/O devices. Figure 2 shows the architecture of IOCheck. The box on the left lists all of the I/O devices on a motherboard, and System Management Mode on the right periodically checks the integrity of I/O configurations and firmware. If an attack is found, IOCheck alerts with an audible beep to notify the user.

A. Configurations of I/O Devices

Before the system boots up, the BIOS initializes all of the hardware devices on the motherboard and set corresponding configurations to them. These devices rely on the configurations to operate correctly. Next, we use the PCI configuration space and IOMMU configuration as examples.

1) PCI Configuration Space: Each PCI or PCI Express controller has a configuration space. Device drivers read these configurations to determine what resources (e.g., memory mapped location) have been assigned by the BIOS to the devices. Note that the PCI configurations should be static after the BIOS initialization. However, an attacker with ring 0 privilege can modify the PCI configuration space. For example, the attacker can relocate the device memory by changing the Base Address Register in the PCI configuration space. Additionally, PCI/PCIe devices that support Message Signaled Interrupts (MSI) contain registers in the PCI configuration space to configure MSI signalling; Wojtczuk and Rutkowska demonstrate that the attacker in the driver domain of a VM can generate malicious MSIs to compromise a Xen hypervisor [20]. Note that IOCheck assumes the PCI configuration remains the same after BIOS initialization, and does not consider "Plug-and-Play" PCI/PCIe devices.

2) IOMMU Configurations: IOMMU restricts memory access of I/O devices. For example, it can prevent a DMA attack from a compromised I/O device. The IOMMU is composed of a set of DMA Remapping Hardware Units (DRHU). They are responsible for translating addresses from I/O devices to physical addresses in the host memory. The DRHU first identifies a DMA request by BDF-ID (Bus, Device, Function number). Next, it uses the BDF-ID to locate the page tables associated with the requested I/O controller. Lastly, it translates the DMA virtual Address (DVA) to host physical address (HPA), which is similar to the MMU translation.

Although IOMMU gives us a nice protection from DMA attacks, it relies on correct configurations to operate appropriately. [8], [20] have demonstrated several ways to bypass IOMMU. IOCheck aims to mitigate these attacks by checking the integrity of the critical configurations of IOMMU at runtime.

For example, the DMA Remapping (DMAR) Advanced Configuration and Power Interface (ACPI) table should never change after booting. The DMAR ACPI table describes the number of DRHUs present in the system and I/O controllers associated with each of them. It is set by the BIOS before the system boots up. In addition, the base address of the configuration tables for DMA remapping unit should be static. IOCheck aims to check the integrity of these static configurations. Table I shows the static configuration of the IOMMU.

B. Firmware Integrity

IOCheck aims to check the firmware of all I/O devices including the network card, graphics card, keyboard, and mouse. Next, we use a NIC and BIOS as examples.

1) Network Interface Controller: First, we discuss how IOCheck checks the integrity of a NIC's firmware. Modern network cards continue to become more and more complex. It usually has its own on-chip processor and memory to support various functionalities. Typically, a NIC loads the firmware from Electric Erasable Programmable Read-Only Memory (EEPROM) to flash memory, and then executes the code on the on-chip processor. To check the integrity of NIC firmware at runtime, IOCheck stores a hash value of original firmware image in the SMRAM while the system executes BIOS code. After the operating system boots up, IOCheck periodically reads the NIC firmware code from the flash memory and calculates the hash value of current image. If the hash value does not match, an attack against NIC may have occurred. In addition, IOCheck monitors the Program Counter of the on-chip CPU through the NIC's debugging registers, which can restrict the instruction pointer to the code section of the memory region. For instance, if the instruction pointer points to a memory region which stores heap or stack data, then a code injection and control flow hijacking may have happened.

Monitoring the integrity of the static code and instruction pointer can prevent an attacker from injecting malicious code into a firmware, but it cannot detect advanced attacks such as Return Oriented Programming (ROP) attacks manipulating the stack without any code injection. To detect these attacks, we can implement a shadow stack to protect the control flow integrity of the NIC firmware. Duflot et al. implemented similar work in NAVIS [12].

TABLE I. IOMMU CONFIGURATIONS

Register/Table Name	Description
Root-entry table address register	Define the base address of the configuration tables
Domain mapping tables	Define mapping between I/O controller and page tables
Page tables	Define memory regions and access permissions of I/O controller
DMAR ACPI table	Define the number of DRHUs and I/O controllers assigned to each of them

2) Basic Input Output System: IOCheck can also check the runtime integrity of the BIOS. As mentioned, a TPM can check the integrity of the BIOS at boot time, which helps us securely load the SMM code from the BIOS to the SMRAM. After the system boots up, attackers with ring 0 privilege can flash the BIOS using various tools (e.g., flashrom [21]). However, they are not able to access locked SMRAM. Thus, we can use SMM code to check the runtime integrity of the BIOS. The checking method is similar to other firmware verification techniques. IOCheck stores a hash value of initial BIOS, and checks if any alterations occur while the system is running.

Although the flashed BIOS with malicious code cannot be executed until the system resets and a TPM will detect this BIOS attack before booting, IOCheck can detect this attack earlier than a TPM, which provides a runtime detection and serves as a complementary defense. Earlier detection of such attacks can also limit the damage they wreak against the system.

VI. FUTURE IMPLEMENTATION AND EVALUATION

A. Programming the SMM

IOCheck leverages SMM to check the integrity of I/O configurations and firmware. It stores the checking code in the SMI handler. In order to program the SMI handler, we use an open source BIOS, Coreboot [22]. Coreboot performs some hardware initialization and then executes additional payload (e.g., SeaBIOS, UEFI). The Coreboot project is written mostly in C, which we use to implement IOCheck.

B. Evaluating IOCheck

The evaluation of IOCheck is composed of two parts: security and performance evaluations. First, we will evaluate the security of IOCheck by comparing with other systems such as VIPER [11] and NAVIS [12]. Next, we will evaluate the performance of IOCheck. From the preliminary experiment result, the SMM switching time takes about 4 milliseconds, which is significantly faster than the late launch method in Flicker [10].

VII. ACKNOWLEDGEMENT

The author would like to thank all of the reviewers for their valuable comments and suggestions. The author also wish to thank Kevin Leach, Kun Sun, Jiang Wang, Quan Jia for their comments on the early draft. Finally, the author would like to thank Angelos Stavrou for his advising and support.

REFERENCES

[1] National Institute of Standards, NIST. National vulnerability database, http://nvd.nist.gov. Access time 2013.04.08.

[2] L. Duflot and Y.-A. Perez, "Can you still trus your network card?" in *CanSecWest*, 2010.

[3] K. Chen, "Reversing and exploiting an Apple firmware update," *Black Hat*, 2009. [Online]. Available: http://www.blackhat.com/presentations/bh-usa-09/CHEN/BHUSA09-Chen-RevAppleFirm-PAPER.pdf

[4] P. Stewin and I. Bystrov, "Understanding DMA Malware," in *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'12)*, ser. Lecture Notes in Computer Science, U. Flegel, E. Markatos, and W. Robertson, Eds. Springer Berlin Heidelberg, 2012, vol. 7591, pp. 21–41. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-37300-8_2

[5] D. Aumaitre and C. Devine, "Subverting Windows 7 X64 Kernel With DMA Attacks," in *HITBSecConf Amsterdam*, 2010. [Online]. Available: http://esec-lab.sogeti.com/dotclear/public/publications/10-hitbamsterdam-dmaattacks.pdf

[6] A. Triulzi, "Project Maux Mk.II," in *CanSecWest*, 2008.

[7] F. Sang, V. Nicomette, and Y. Deswarte, "I/O Attacks in Intel PC-based Architectures and Countermeasures," in *SysSec Workshop (SysSec), 2011 First*, July, pp. 19–26.

[8] F. Sang, E. Lacombe, V. Nicomette, and Y. Deswarte, "Exploiting an I/OMMU vulnerability," in *5th International Conference on Malicious and Unwanted Software (MALWARE'10),*, 2010, pp. 7–14.

[9] Trusted Computing Group, "Trusted Platform Module main specification. version 1.2, revision 103, 2007." [Online]. Available: http://www.trustedcomputinggroup.org/resources/tpm_main_specification

[10] J. McCune, B. Parno, A. Perrig, M. Reiter, and H. Isozaki, "Flicker: An execution infrastructure for TCB minimization," in *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*, 2008.

[11] Y. Li, J. McCune, and A. Perrig, "VIPER: verifying the integrity of PERipherals' firmware," in *Proceedings of the 18th ACM conference on Computer and communications security (CCS'11)*, 2011.

[12] L. Duflot, Y.-A. Perez, and B. Morin, "What if you can't trust your network card?" in *Recent Advances in Intrusion Detection (RIAD'11)*, ser. Lecture Notes in Computer Science, R. Sommer, D. Balzarotti, and G. Maier, Eds. Springer Berlin / Heidelberg, 2011, vol. 6961, pp. 378–397.

[13] F. Zhang, K. Leach, K. Sun, and A. Stavrou, "SPECTRE: A Dependable Intropsection Framework via System Management Mode," in *Proceedings of The 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'13)*, 2013.

[14] Y. Bulygin and D. Samyde, "Chipset based approach to detect virtualization malware a.k.a. DeepWatch," *Blackhat USA*, 2008.

[15] A. Tereshkin and R. Wojtczuk, "Introducing Ring -3 Rootkits," 2009. [Online]. Available: http://invisiblethingslab.com/itl/Resources.html

[16] J. Wang, A. Stavrou, and A. Ghosh, "HyperCheck: A hardware-assisted integrity monitor," in *Proceedings of 13th International Symposium On Recent Advances In Intrusion Detection*, 2010.

[17] 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*, 2010.

[18] S. Embleton, S. Sparks, and C. Zou, "SMM rootkits: a new breed of OS independent malware," in *Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*, 2008.

[19] R. Wojtczuk and J. Rutkowska, "Attacking SMM Memory via Intel CPU Cache Poisoning," 2009. [Online]. Available: http://invisiblethingslab.com/resources/misc09/smm_cache_fun.pdf

[20] ——, "Following the White Rabbit: Software Attacks against Intel VT-d," 2011. [Online]. Available: http://invisiblethingslab.com/itl/Resources.html

[21] "Flashrom utility." [Online]. Available: http://www.flashrom.org/

[22] "Coreboot." [Online]. Available: http://www.coreboot.org/