Microelectronics Reliability 100–101 (2019) 113413

Contents lists available at ScienceDirect

Microelectronics Reliability

journal homepage: www.elsevier.com/locate/microrel

Hardware-based approach to guarantee trusted system boot in embedded systems

Lucas Correa, Fabian Vargas*, Letícia Poehls

Electrical Engineering Dept., Catholic University – PUCRS, Av. Ipiranga 6681, 90619-900 Porto Alegre, Brazil

ARTICLE INFO

Keywords:
I/O attack
Malicious peripheral
Peripheral Component Interconnect Express (PCIe)
Trusted system boot
Dynamic integrity checking
Secure embedded system

ABSTRACT

In recent years, computer systems spread in our daily lives have been experiencing an increasing wave of attacks that disrupt their normal operation or leak sensitive data. Some of them, so-called I/O attacks, are performed by malicious peripherals that perform read or write accesses to DRAM memory through unauthorized DMA (Direct Memory Access) requests during the system boot. During this period the system is particularly vulnerable, which allows hackers to make use of peripherals to perform malicious accesses to DRAM memory. As consequence, it degrades system security. In this context, this paper presents a hardware-based secure-system boot sniffer (SBS) Watchdog to prevent I/O attacks during system boot. The SBS Watchdog is tightly connected to the PCIe communication bus to prevent peripherals from accessing DRAM memory during the BIOS execution and operating system loading, i.e., during the system boot. Compared to existing approaches, the proposed technique is fully transparent to the user, can be applied to any operating system, induces no performance degradation since the SBS Watchdog is turned-off after system boot, and cannot be unintentionally disabled by users during system configuration procedure. The proposed technique was implemented in a commercial FPGA. Practical experiments have demonstrated the high effectiveness of the proposed technique to prevent I/O attacks during system boot.

1. Introduction

The need to include security mechanisms in electronic tiny devices has dramatically grown with the widespread use of such devices in our daily life [1–3]. In this scenario this work presents a hardware-based approach, here denoted as a watchdog connected to the Peripheral Component Interconnect Express (PCIe) bus, which aims at preventing I/O attacks during system boot. The approach is based on a dedicated secure-system boot sniffer (SBS) Watchdog, which is tightly connected to the PCIe bus to prevent peripherals from accessing DRAM memory during the BIOS execution and operating system loading, i.e., during the system boot. During this period, the system is particularly vulnerable because configuration tables of the operating system (which are located in a DRAM region that is not protected from Direct Memory Access (DMA) accesses) are initialized during the system boot. These configuration tables contain information concerning the memory region and the type of memory access (read and/or write) that are allowed for each of the peripherals connected to the system board chipset. Since such configuration tables are not ready for use by the hardware before the end of the system boot process, the firmware is not able to filter DMA accesses from the peripherals, allowing them to read and/or write system main memory with no protection. See Fig. 1 for details. (Note: the I/O Memory Management Unit – IOMMU will be described in the next section). Thus, a hacker behind a malicious peripheral may benefit from this vulnerable moment to modify these tables in memory just before the hardware is properly configured by the operating system during the system boot. In this scenario, system main memory can be initialized with a chunk of code that can be executed by the hacker afterwards to control the system and/or to leak any information stored therein. As a consequence, the system security is degraded.

2. Related works

Several approaches found in the literature have been used with different degrees of success, as it will be described in the following paragraphs:

1) To cope with such attacks, Intel developed a hardware protection component: I/O Memory Management Unit (IOMMU), which has been included in many modern computers. However, it is shown that even if this component has been introduced 10 years ago, some serious security concerns may be raised about its actual efficiency to

* Corresponding author.
E-mail address: vargas@computer.org (F. Vargas).

https://doi.org/10.1016/j.microrel.2019.113413
Received 15 May 2019; Received in revised form 25 June 2019; Accepted 5 July 2019
Available online 23 September 2019

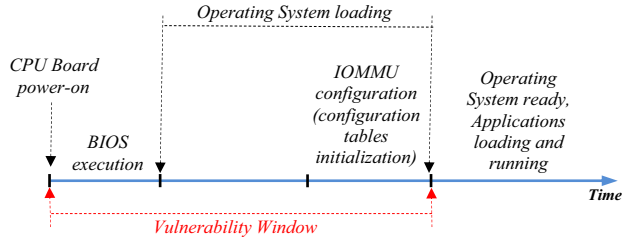


Fig. 1. System boot timeline and vulnerability window.

prevent malicious I/O attacks due to some design weaknesses in its configuration [4]. In more detail, the IOMMU component is able to implement a DMA Protected Range (DPR) segment, which is specified by the processor and whose Protected Memory Ranges (PMR) are implemented by the IOMMU in the system memory. However, the Linux operating system does not use these memory segments placing IOMMU structures outside the protected ranges [4]. Peripheral devices are consequently able to read and write the IOMMU configuration before its activation.

2) Another hardware protection mechanism is the Bus Master Enable (BME) bit [5], which can prevent the peripherals from sending messages during BIOS and operating system loading. If this bit is set on in the configuration of the different PCIe bridges, the peripherals that are connected to these bridges cannot send any DMA requests by themselves anymore and thus cannot perform memory attacks. This bit is correctly set at the startup of the system. However, in many chipset configurations, firmware disables this protection before giving the control to the bootloader.
3) Also, another efficient protection against this type of attack is to use the Intel Trusted Execution Technology (Intel TXT) to configure the hardware of each computer [6,7]. Intel TXT works by creating a Measured Launch Environment (MLE) that enables an accurate comparison of all the critical elements of the launch environment against a known good source. Intel TXT creates a cryptographically unique identifier for each approved launch enabled component and then provides hardware-based enforcement mechanisms to block the launch of code that *does not match approved code* (i.e., *the cryptographic identifier)*. This Intel hardware-based solution provides the foundation on which trusted platform solutions can be built to protect against the software-based attacks that threaten integrity of systems. However, setting such hardware/software infrastructure protection is quite tricky, difficult for an end user which is not a hardware and operating system expert. As a consequence, up to our knowledge, Intel TXT is not a solution that is largely deployed so far. It is therefore necessary to find solutions that are efficient without being too restrictive.
4) In [7] authors developed an efficient hardware package to wrap around all outsourced hardware (intellectual property – IP) cores that are able to perform DMA via system bus interface. Such a technique aims at guaranteeing secure boot for specific System-on-Chips (SoCs) known as FPGA SoCs. This type of SoCs are known to include in addition to a conventional CPU architecture (containing entities like processor, memory and bus), a configurable hardware logic (to map the outsourced cores) in the same chip. In this scenario if IP cores are sourced from third-parties, they are hence questionable whether they could contain (in addition to the logic that performs their desirable functionality) unwanted logic devoted to perform unauthorized access to the system memory and so, leak/corrupt sensitive data stored in memory during system boot. The hardware wrapper demonstrates to be efficient to guarantee secure system boot, but note that one hardware wrapper must be added to every outsourced IP core mapped in the FPGA. This could result in an unacceptable area overhead for systems containing a large number of IP cores that are able to perform DMA via system bus interface.
5) Finally, if we consider tiny devices such as IP cameras, smart-TVs, smart-phones and multimedia central in autonomous vehicles for instance, they typically do not run with the latest full-version of Windows or Linux operating systems and in the majority of the cases they do not even run with these operating systems at all. So, they do not benefit from the native security mechanisms embedded into these operating systems as previously described. Instead, they use home-tailored, optimized-code operating systems devoted to their target application. So, they are extremely vulnerable to I/O attacks during system boot.

Interested readers on the issue of secure boot of embedded systems can also address additional references on [8–11]. Considering the aforementioned approaches, the proposed technique presents the following advantages: (a) it is fully transparent to the user (i.e., does not need/depend on any user configuration). (b) can be applied to any operating system. (c) induces no performance degradation since the SBS Watchdog is turned-off after system boot. (d) results in a negligible area overhead since a unique SBS Watchdog is designed to handle any number of peripheral ports connected to the PCIe bus, and (e) cannot be unintentionally disabled by users during system configuration procedure.

In order to improve the drawbacks previously described, the next section presents the proposed approach. It is worth mentioning that such approach can be used as a stand-alone solution or even in conjunction with any of the previous approaches.

3. The proposed approach

The approach is based on a dedicated secure-system boot sniffer (SBS) Watchdog, which is tightly connected to the PCIe bus to prevent peripherals from accessing DRAM memory during the BIOS execution and operating system loading, i.e., during the system boot.

The proposed approach can be applied to any embedded system, assuming the condition that the Intel IOMMU hardware protection component has been incorporated in the system board chipset. IOMMUs are designed to virtualize the memory space and the interrupts of the peripherals [5,12]. Memory virtualization is implemented in the so-called DMA remapping units (DMAR) of the IOMMU. In this scenario, the DMAR units are configured based on the Global Command Register (GCMDR) and in particular on the Translation Enable Bit (TE Bit), which is part of the GCMDR. Both, TE Bit and GCMDR are stored in the system memory and are updated by the BIOS firmware during its execution. The existence of the Intel IOMMU hardware component on the board chipset is mandatory because the SBS Watchdog needs to be configured with information stored in the TE Bit. During system start up, TE Bit is set to “0” by the BIOS firmware, and set to “1” by the operating system when it finishes loading and is ready to start running user tasks.

For embedded systems that do not support the Intel IOMMU hardware component, the equivalent bit to the TE Bit must be identified and its address used to configure the SBS Watchdog. By doing so, the SBS Watchdog will be able to recognize when the system is ready to start running user application and then, release peripherals to freely access system memory.

Fig. 2 depicts details of the general architecture. As can be seen, the SBS Watchdog is set between the PCIe Host and the peripherals (via a PCIe Switch). In such connection, the SBS has as function to monitor all communication traffic from the peripherals towards the PCIe Host Block during the period when the BIOS is being executed and the operating system is being loaded. After this critical period (CP), the peripherals are released to communicate at any time, with no restriction with the DMA Controller. In more detail, while TE Bit = “0”, the SBS Watchdog allows the peripheral to send to the DMA Controller *only* the data that are solicited by the PCIe Host to configure the board chipset

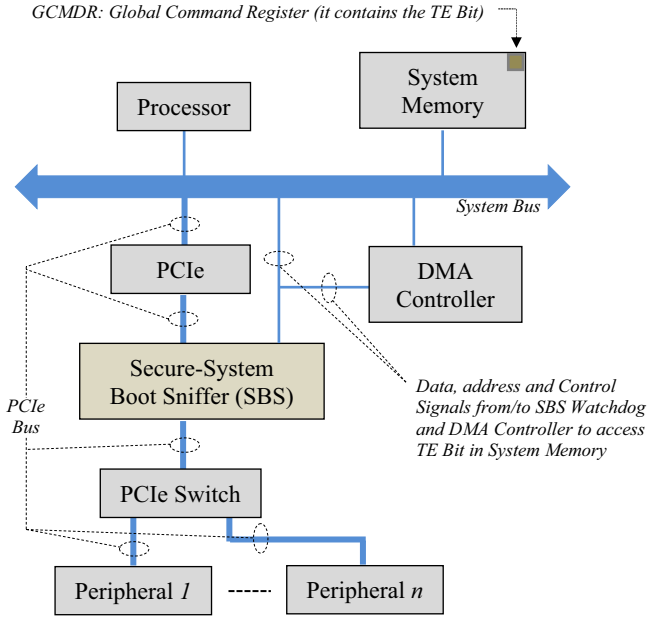


Fig. 2. General architecture of the SBS Watchdog.

during system boot. In this scenario, any request from the peripheral to the PCIe Host to write into the system memory is promptly blocked by the SBS Watchdog. When TE Bit assumes logical value "1", peripherals are released to communicate at any time, with no restriction with the DMA Controller.

Fig. 3 depicts in more detail this situation. Note that data coming from the PCIe Host pass directly to the peripheral without scrutiny of the SBS Watchdog. Nevertheless, the data coming from the peripheral to the PCIe Host are checked by the SBS Watchdog. The communication on the PCIe bus is given according to a specific pattern, which is depicted in Fig. 4. As seen in this figure, data is formatted in a 4 × 32-bit word, with 18 fields in a total. When the SBS Watchdog receives data from the peripheral formatted in such way, it checks if it concerns on a request from the peripheral to the PCIe Host to write in the system memory. If it is the case, the SBS Watchdog interrupts immediately such communication traffic. Otherwise, it allows the communication process to proceed till the end.

Fig. 5 depicts the connections between the SBS Watchdog and the DMA Controller. When the SBS Watchdog has to update the TE Bit Register the *System Boot Monitor and Bus Interface* Block (Fig. 6) starts communicating with the DMA Controller by sending a *DMA Req* signal, which is answered by the latter by sending a *DMA Ack* signal. Then, the SBS Watchdog indicates to the DMA Controller that it needs to read the content of the TE Bit in the system memory. This is done by sending a *DMA Read/Write* signal to the DMA Controller while writing the

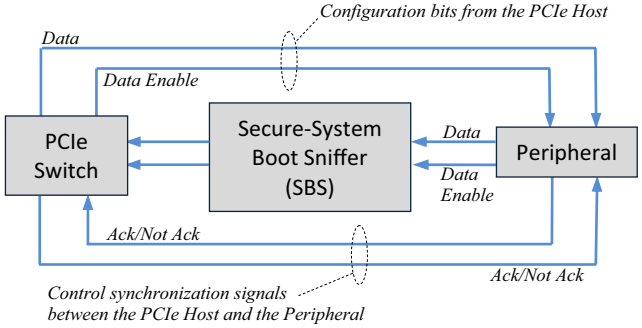


Fig. 3. Details of the connection between SBS and PCIe Host by one side and the SBS and Peripherals via PCIe Switch by the other side.

GCMDR (i.e., the TE Bit) address in the *Address Bus*. When the data stored in the GCMDR is read out from the memory and is available in the Data Bus, the DMA Controller replies with a *DMA Data Enable* signal in order for the SBS Watchdog to read the *Data Bus*.

Fig. 6 depicts the internals of the SBS Watchdog. During system boot the TE Bit is set to "0" in the System Memory by the BIOS firmware (Fig. 2). When the SBS Watchdog is powered on, the *System Boot Monitor and Bus Interface* Block requests to the DMA Controller to read the status of the TE Bit in the System Memory. Then the *System Boot Monitor and Bus Interface* Block initializes the *TE Bit Register* = "0". The *System Boot Monitor and Bus Interface* block repeats periodically this communication with the DMA Controller in order to keep updated the *TE Bit Register* with the most recent system boot status. While the TE Bit in the System Memory (and so the *TE Bit Register*) is equal to "0", the selected *Decoder* output is A. When the *TE Bit Register* switches to "1", the *Decoder* output changes to B (this means that the operating system has already been loaded and the SBS Watchdog is bypassed).

While *TE Bit Register* = "0", the SBS Watchdog operates as follows (Fig. 7 presents the SBS flow chart for details):

a) First, the Receiver Block detects that there is a communication from the peripheral to the PCIe Host, receives and stores temporarily the requested data in the 4 × 32-bit word format as depicted in Fig. 4, while forwarding it to the Buffer and Packet Analyzer Blocks.

b) In the Packet Analyzer Block, the word is decoded and checked if it is a write operation solicited by the peripheral to the PCIe Host. If it is not the case (for instance, the peripheral is sending its identification number, ID, to the DMA Controller), the word stored in the Buffer Block is released to the PCIe Host. However, if it is a write request, then the contents of the fields Fmt and Type (Fig.4) are as follows: Fmt = "10" or "11": it means a memory write request from the peripheral to the PCIe Host for a 32 or 64 bits address, respectively; and Type = "00000". In this case, the Packet Analyzer Block understands that the peripheral is attempting to perform an unauthorized write operation into the System Memory and so, it interrupts immediately the communication (i.e., the word stored in the Buffer is flushed).

4. Experimental results

The proposed technique was validated by a set of experiments realized with the GHDL compiler [13] and GTKWave waveform [14], both open source simulation software packages. With this purpose, two blocks describing the PCIe Host and a peripheral emulator were implemented in VHDL language and prototyped into a Virtex-7 Xilinx FPGA (Part number xc7vx485tffg1761-2). Fig. 8 depicts these blocks and their interconnections.

Note that the peripheral emulator is a *generic* block implemented only with the PCIe functionality. With this purpose, other native functions necessary to implement a smart device were dismissed since they were not the goal and were not even necessary to validate the proposed approach. Note also that the *PCIe Host* Block was implemented in a minimized version containing only the logic necessary to guarantee a transaction between the PCIe Host and the peripheral. Examples of such a transaction can be a *read request* from the PCIe Host to the peripheral (Fig. 9) or even a request from the peripheral to the PCIe Host to *write into the system memory* (Fig. 10).

Simulation signals seem in Fig. 9 are described as follows: *a)* System clock; *b)* System reset; *c)* Data stream sent by the peripheral to the SBS Watchdog; *d)* Data stream received by the SBS Watchdog from the peripheral and forwarded to the PCIe Host; *e)* Data stream received by the PCIe Host from the SBS Watchdog. As observed, data stream is received and forwarded by the SBS Watchdog. This process can be guaranteed because after analyzing fields *Fmt* and *Type* of the 4 × 32-bit word format (Fig.4) the SBS Watchdog decides to release data since

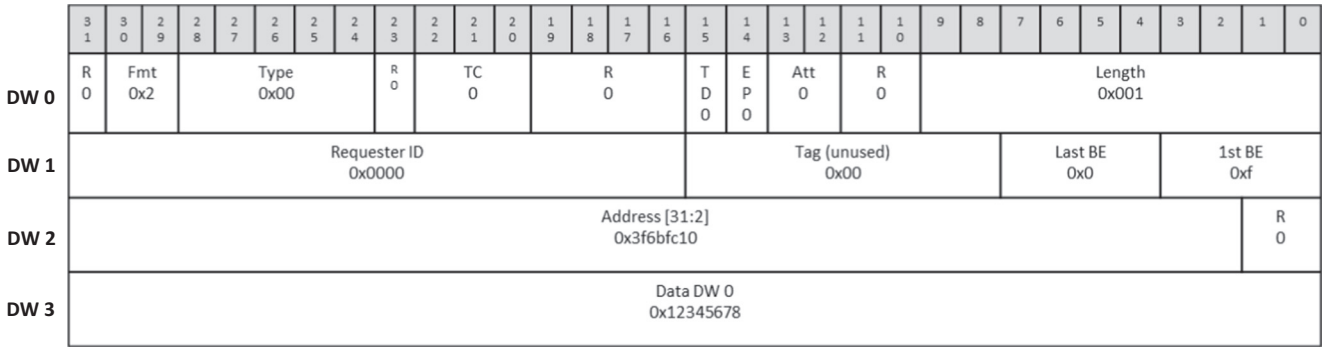


Fig. 4. Data format for communication between peripheral and PCIe Host.

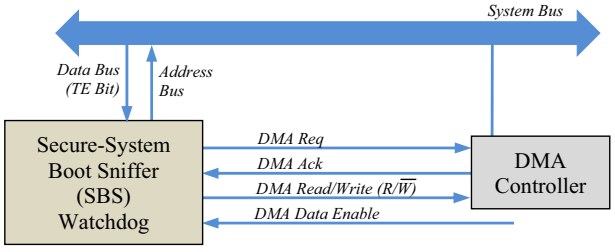
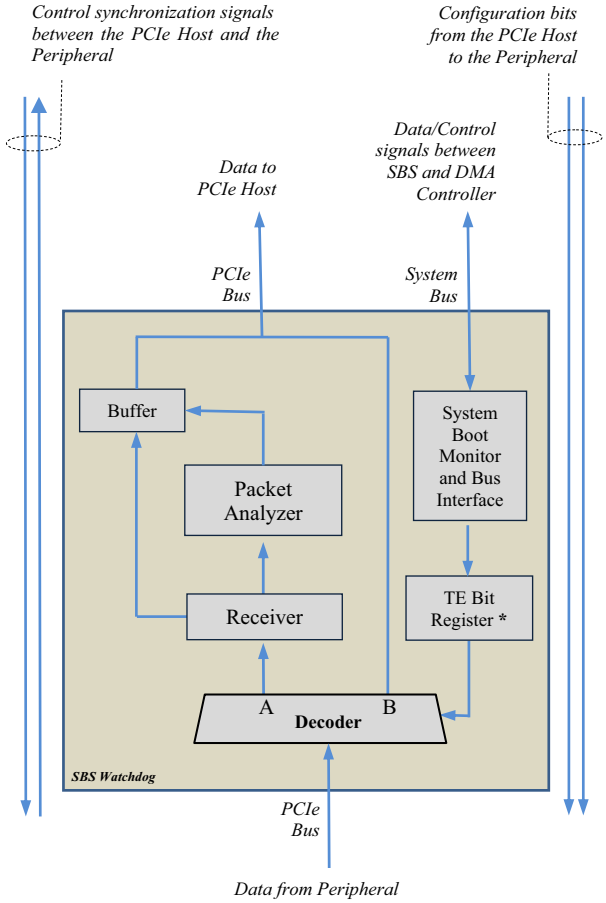


Fig. 5. Details of the connection between SBS and DMA Controller Block.



** **TE Bit**: Translation Enable Bit. It contains the system boot status:*
TE Bit = "0": during BIOS and Operating System loading/memory configuration.
TE Bit = "1": IOMMU mounted and ready for use.

Fig. 6. Secure-System Boot Sniffer (SBS) internal blocks.

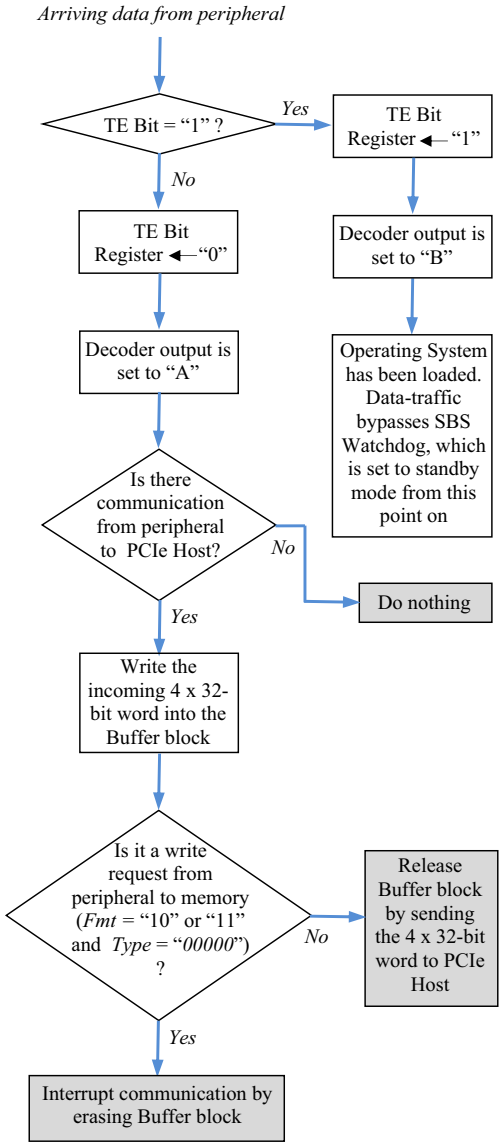


Fig. 7. SBS Watchdog flow chart.

it does not represent a *memory write request* from the peripheral into the system memory. Nevertheless, Fig. 10 describes a *memory write request* transaction from the peripheral to the PCIe Host. And as expected, the SBS Watchdog interrupts such communication by cutting down the data stream.

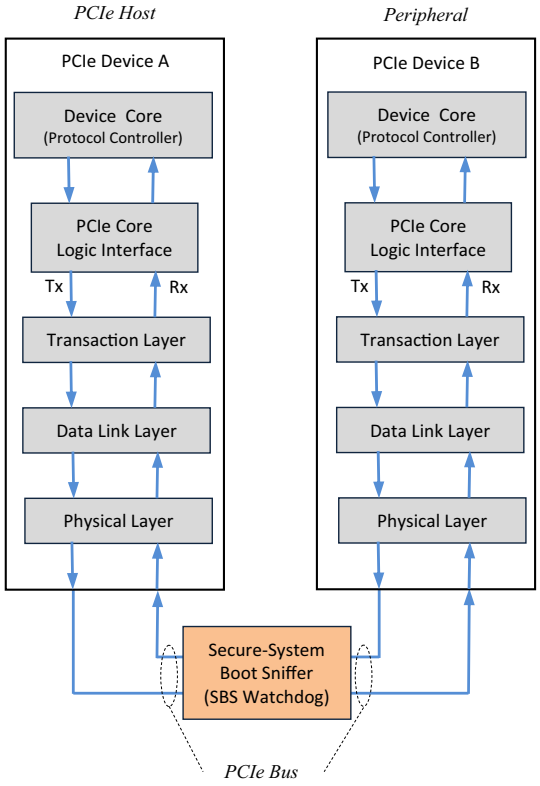


Fig. 8. PCIe Host and Peripheral Emulator Blocks described in VHDL language and simulated in ModelSim to validate the proposed approach.

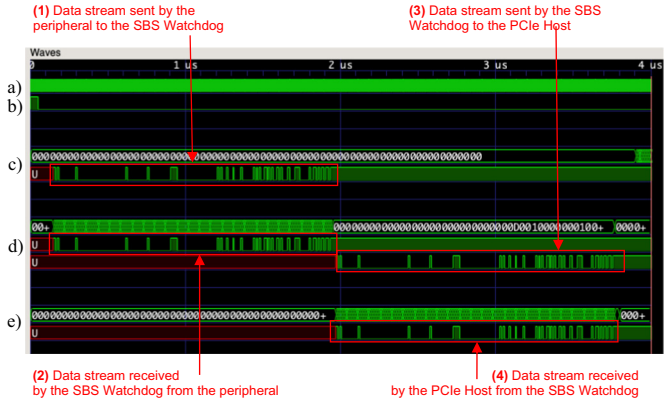


Fig. 9. ModelSim-based VHDL simulation implemented to validate the proposed technique: simulation of a *data read request* from the PCIe Host to the peripheral.

5. Final considerations

This work presented a hardware-based approach to prevent I/O attacks during system boot, here denoted as a dedicated secure-system boot sniffer (SBS) Watchdog. This watchdog is tightly connected to the Peripheral Component Interconnect Express (PCIe) bus to prevent peripherals from accessing DRAM memory during the BIOS execution and operating system loading, i.e., during the system boot. During this period, the system is particularly vulnerable because configuration tables of the operating system (which are located in a DRAM region that is not protected from Direct Memory Access (DMA) accesses) are not ready for use by the hardware before the end of the system boot process. Therefore, the firmware is not able to filter DMA accesses from the peripherals, allowing them to read and/or write system main memory indiscriminately, with no protection. Thus, a hacker behind a malicious

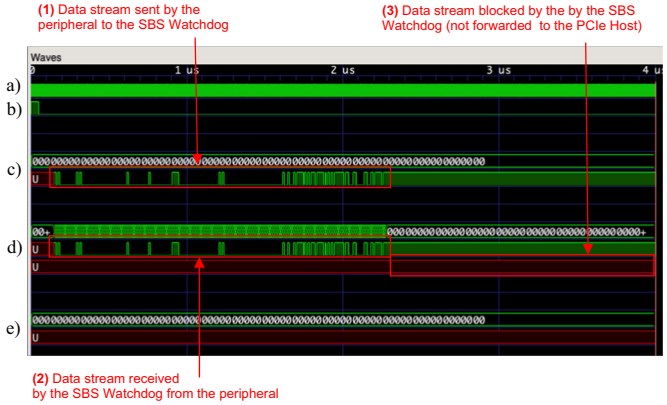


Fig. 10. ModelSim-based VHDL simulation implemented to validate the proposed technique: simulation of a *memory write request* from the peripheral to the PCIe Host.

peripheral may benefit from this vulnerable moment to modify these tables in memory just before the hardware is properly configured by the operating system. In this scenario, system main memory can be initialized with a chunk of code that can be executed by the hacker afterwards to control the system and/or to leak any information stored therein. As consequence, the system security is degraded.

In order to validate the proposed approach, the SBS Watchdog, a simplified version of the PCIe Host and an entity that mimics the functionality of a PCIe-compliant peripheral connected to the PCIe bus of a system board chipset have been implemented in VHDL language and prototyped into a Virtex-7 Xilinx FPGA (Part number xc7vx485tffg1761-2). The obtained results confirm that the SBS Watchdog is able to interrupt any attempt from the peripheral to write into the system memory during the system boot period. All the other communication between the PCIe Host and the peripheral is allowed.

Considering the existing approaches, the proposed technique presents the following advantages: (a) it is fully transparent to the user (i.e., does not need/depend on any user configuration). (b) can be applied to any operating system. (c) induces no performance degradation since the SBS Watchdog is turned-off after system boot. (d) results in a negligible area overhead since a unique SBS Watchdog is designed to handle any number of peripheral ports connected to the PCIe bus, and (e) cannot be unintentionally disabled by users during system configuration procedure.

6. Current work

Currently a full embedded system is under construction, where the SBS Watchdog and the simplified PCIe Host are being coupled with the Leon3 softcore processor and a DMA Controller. The whole VHDL-based system is being mapped to the aforementioned FPGA device. The hardware implementation of the SBS Watchdog indicated preliminarily an area usage of the FPGA in the order of 2051look-up tables (LUTs) and 562 registers. With the purpose of comparison, the area occupied by the Leon3 softcore processor implemented in the same FPGA was 15,393 LUTs and 10,794 registers. In this scenario, the area overhead induced by the SBS Watchdog is only 9.97%. Note that the final LUT count, after physical optimizations and full implementation, is typically lower. Then, running the Xilinx *opt_design* tool after synthesis will give us a more realistic (and even more optimistic) count.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work has been supported in part by CNPq (National Science Foundation, Brazil) under contract n. 306619/2015-6 (PQ) and Project Suspensys-Randon Company (TA 04/2016).

References

[1] Common Vulnerabilities and Exposures - The Standard for Information Security Vulnerability Names. URL: https://cve.mitre.org/, last access: Jan 2018.
[2] R.S. Ferreira, F. Vargas, ShadowStack: a new approach for secure program execution, Microelectron. Reliab. J. 55 (9) (2015) 2077–2081. August.
[3] Yong-Joon Park, Zhao Zhang, Gyungho Lee, "Microarchitectural protection against stack-based buffer overflow attacks", IEEE Micro., vol. 26 , issue 4, July-Aug. pp. 62–71, 2006.
[4] B. Morgan, É. Alata, V. Nicomette, M. Kaâniche, Bypassing IOMMU protection against I/O attacks, 2016 Seventh Latin-American Symposium on Dependable Computing (LADC), Oct 2016, Cali, Colombia, 2016, pp. 145–150, , https://doi.org/10.1109/LADC.2016.31.
[5] Intel White Paper, A Tour Beyond BIOS: Using IOMMU for DMA Protection in UEFI Firmware, https://firmware.intel.com/sites/default/files/Intel_WhitePaper_Using_IOMMU_for_DMA_Protection_in_UEFI.pdf , Accessed date: January 2019.
[6] Intel White Paper, Intel Trusted Execution Technology: Hardware-based Technology for Enhancing Server Platform Security, URL www.intel.com/content/www/us/en/architecture-and-technology/trusted-execution-technology/trusted-execution-technology-security-paper.html , Accessed date: January 2019.
[7] N. Jacob, J. Heyszl, A. Zankl, C. Rolfes, G. Sigl, How to break secure boot on FPGA SoCs through malicious hardware, 19th Int. Conference on Cryptographic Hardware and Embedded Systems – CHES, Taipei, Taiwan, 2017, pp. 425–442 Sept. 25-28.
[8] Kuan-Jen Lin, Chin-Yi Wang, "Using TPM to improve boot security at BIOS layer", 2012 IEEE International Conference on Consumer Electronics (ICCE), pp, 376–377.
[9] T. Kai, X. Xin, C. Guo, The secure boot of embedded system based on mobile trusted module, International Conference on Intelligent Systems Design and Engineering Application, 2012, pp. 1331–1334.
[10] Yuan Liu, Jed Briones, Ruolin Zhou, Neeraj Magotra, "Study of secure boot with a FPGA-based IoT device", 2017 IEEE 60th Int. Midwest Symposium on Circuits and Systems (MWSCAS), pp. 1053–1056.
[11] R.V. Rashmi, A. Karthikeyan, Secure boot of embedded applications – a review, 2nd Int. Conference on Electronics, Communication and Aerospace Technology, ICECA, 2018, pp. 291–298.
[12] Benoît Morgan, Éric Alata, Vincent Nicomette, Mohamed Kaâniche, IOMMU protection against I/O attacks: a vulnerability and a proof of concept, Springer Open Access J. Braz. Comput. Soc. 24 (2018) 2, https://doi.org/10.1186/s13173-017-0066-7.
[13] GHDL, http://ghdl.free.fr/ , Accessed date: January 2019.
[14] Welcome to GTKWave, http://gtkwave.sourceforge.net/ , Accessed date: January 2019.