하드웨어 Trojan 사례 연구: 캐시 일관성 규약을 악용한 DoS 공격

공선희, 홍보의, 서태원
고려대학교 컴퓨터학과
e-mail : nickong@korea.ac.kr

A Case Study on Hardware Trojan: Cache Coherence-Exploiting DoS Attack

Sunhee Kong, Bo-Uye Hong and Taeweon Suh
Dept. of Computer Science and Engineering, Korea University

Abstract

The increasing complexity of integrated circuits and IP-based hardware designs have created the risk of hardware Trojans. This paper introduces a new type of threat, the coherence-exploiting hardware Trojan. This Trojan can be maliciously implanted in master components in a system, and continuously injects memory read transactions on to bus or main interconnect. The injected traffic forces the eviction of cache lines, taking advantage of cache coherence protocols. This type of Trojans insidiously slows down the system performance, incurring Denial-of-Service (DoS) attack. We used Xilinx Zynq-7000 device to implement and evaluate the coherence-exploiting Trojan. The malicious traffic was injected through the AXI ACP interface in Zynq-7000. Then, we collected the L2 cache eviction statistics with performance counters. The experiment results reveal the severe threats of the Trojan to the system performance.

1. Introduction

In tandem with the market demand and the Moore's law, the hardware complexity of modern computers increases twice every 18 months. For example, the Intel's latest processor, Haswell, boasts 1.7 billion transistors integrated in a single chip [6]. The number of engineers in a design team easily exceeds a few hundreds. The silicon product goes through many steps to be finally introduced to the market: Planning, Architecting Hardware Design, Manufacturing, and Validation. A few thousands of engineers would be involved in the whole processes. Thus, it would not be feasible to audit the work from every single engineer as long as the product functions as intended. It indicates that hardware components are not free from malicious modifications and insertions via evil insiders. Even worst, there are some speculations that some products are intentionally modified to include a kill switch [7].

In the stiff market competition, embedded device vendors are forced to reduce their production cost and the outsourcing of hardware components in a production has become a norm to reduce the time-to-market delay; There are an increasing number of companies that design and sell embedded gadgets but do not manufacture the hardware components by themselves. Even though a vendor may manufacture their devices on their own, it is more than likely that the hardware components in the devices are integrated with numerous IP cores outsourced or licensed by diverse third parties.

This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2012R1A1A2008231)

Moreover, there is a report that counterfeit hardware components are used not only in social infrastructure (such as high-speed trains) but also in military products (such as fighter jets) [7].

This tendency has brought worries that outsourced components may not be very trustworthy. Potentially, anyone who are involved in the manufacturing process might make some malicious modifications to the original chip design, which is known as Hardware Trojan. The hardware Trojans are insidious and hard to detect much more than the software viruses, posing the critical threats to the system security. Even when detected, it is not possible to remove or cure the infected system since it is fused into the hardware components. Especially, when used in military products, the national security could be at risk, so extra care and screening should be taken in adopting off-the-shore components.

Among the various types of hardware Trojans, this paper focuses on a Trojan incurring the Denial-of-Service (DoS) attack. More specifically, we investigate the malicious traffic injections from a master component in a hardware system, taking advantage of the memory coherence mechanisms. This type of attack does not alter or crash the system. It simply slows down the system performance, making it even harder to detect its existence. Nevertheless, it could make a critical impact on systems such as real-time systems or military product such as anti-missile and radar systems. We used Xilinx Zynq-7000 device on ZedBoard for the experiment to measure the impact of the coherence-exploiting Trojan. We also suggest countermeasures to mitigate its impact on a system.

The rest of this paper is organized as follows: Section 2 presents the recently proposed taxonomy and reviews various types of Trojans demonstrated in previous studies. Section 3 introduces some potential places in a chip where DoS attack can occur. Our experiment on coherence-exploiting Trojan is explained in Section 4. Section 5 gives an analysis on the results and is followed by Section 6 with the conclusion.

2. Related Work

The insertion, detection, prevention, and countermeasures of hardware Trojans have become one of the most critical contemporary issues in IC security. There are numerous research works published in academia [1, 2, 3, 4, 5, 7]. Wang et al. [1] classified the hardware Trojans according to their types [2]. Karri et al. extended the Wang's taxonomy [3] based on the following five areas;

1) *Insertion phase:* The typical development cycle of an IC goes through from "Specification" phase to "Design" and then "Fabrication" followed by "Testing" and ends with "Assembly" phase. Although the malicious modification could possibly be inserted at any of these 5 phases, there are few Trojans inserted at the phases other than "Design" stage. [3] demonstrated zero-overhead modification at the fabrication phase which enables privilege escalation attacks on modern microprocessors.

2) *Abstraction level:* Undesirable alteration to a circuit could be made at various hardware abstraction levels. At low levels the circuit can be modified maliciously by thinning a wire or weakening a transistor [1]. At high levels, Trojan horses can be implanted during preprocessing through tampering with CAD tools or scripts [5].

3) *Activation mechanism:* Some attackers may want their Trojans to be activated only under a specific condition whereas the others want theirs to be always on. *Trojan type VIII* of [4] is designed to remain dormant until "Caps Lock" or any undefined key is pressed while *Trojan type III* is always in operation.

4) *Effects:* Destructive behaviors and harmful influences that maliciously-modified logics can introduce vary from a small downgrade in system performance to an extremely destructive one like Trojan type II in [4] which causes the whole system to stop working.

5) *Location:* A malicious circuit can also be characterized by the location where it is embedded.
Some Hardware Trojans such as Time-Bomb [8] could create the DoS effect in a system through the system crash. Our paper differs from the previous works in that the coherence-exploiting Trojan insidiously slows down the system performance. To the best of our knowledge, this type of DoS attackingTrojan has not been investigated before.

3. Potential Places of DoS Attacks

The DoS attacks could happen anywhere in a chip if the implanted Trojan could influence the system performance in any form. This section introduces the most noticeable places of DoS attacking Trojans.

A. *Memory Transactions*

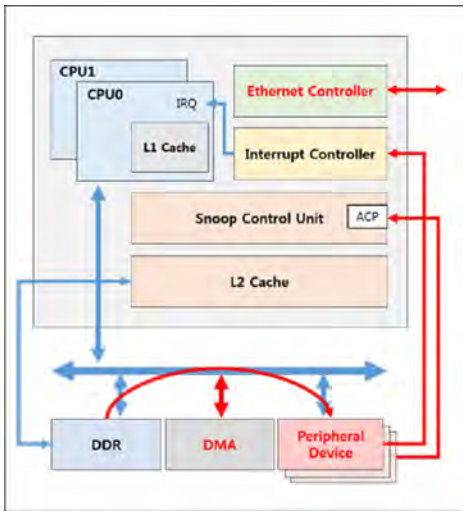
Accessing memory plays a critical role in system performance due to the performance gap from CPU. Thus, the malicious traffic injection could incur the system perturbation in a great scale. The multi-core configuration of the modern systems aggravates the threat of the malicious traffic injections because the number of master components in a system increases. There are two kinds of transactions to memory: Write and Read. Injecting a write transaction to memory could change the system state or would more likely incur the system crash if not carefully injected since the kernel and/or user space are most likely to be spoiled. Thus, the Trojan with write injection would more easily be discovered. Whereas, a read transaction is harmless and yet insidiously influences the system performance. Maliciously moving data (from memory to peripheral devices) around via Direct Memory Access (DMA) is one example of this type.

B. *Coherence Transactions*

Multi-core systems require cache coherence protocol for data consistency. A write transaction in one master which modifies shared data incurs cache line evictions in the others. There are some cases where even a read traffic could trigger coherence transactions in caches. For example, suppose a dual-core system with private L1 caches and a shared L2 cache which are strictly inclusive. If one of the cores injects a read transaction which causes a line marked shared to be replaced at the L2 cache, the victim will also be evicted from both L1 caches in order to satisfy the inclusive property. Our paper focuses on this type of Trojan, referred to as coherence-exploiting Trojan.

C. *Interrupts*

For computer operations, the interrupt is a fundamental mechanism for communication between CPU and peripheral devices. Falsely generating spurious interrupts from malicious peripherals perturbs the whole system, stopping CPU every now and then. This type of Trojans is also insidious and hard to detect.



(Figure 1) Potential places of DoS attacks

D. Network

Hardware Trojans could reside in network interface cards (NICs) to delay sending a packet to the network output, or periodically overrun the incoming packets in the local buffer, triggering retransmission of the packets. This type of Trojans are also stealthy.

(Figure 1) describes potential locations where DoS attacking Trojans can reside with possible paths the attacks can take place. Both are marked in red.

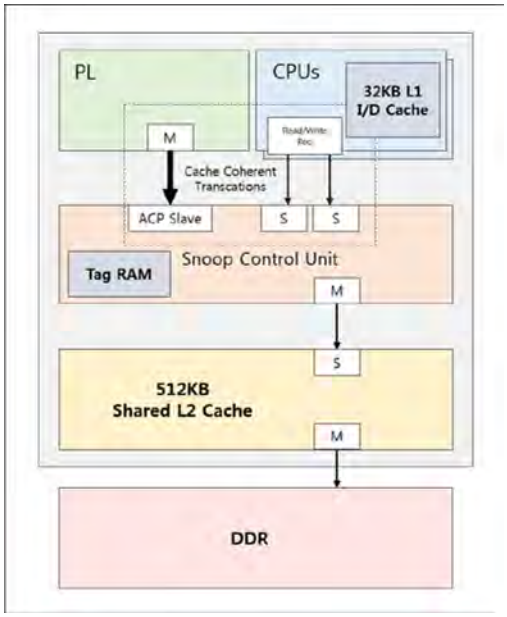
4. Experiment

A. Experimental Setup

We utilized Xilinx Zynq-7000 on ZedBoard to design, implant, and evaluate the coherence-exploiting Trojan. Zynq-7000 is composed of two sections: Processing System (PS) with two Cortex-A9s fused and Programmable Logic (PL) with a typical FPGA fabric. These two sections can communicate through several interfaces Zynq-7000 SoC provides. Statistics on L2 cache and system performance were collected using the event monitoring bus of L2C-310 (AMBA Level 2 Cache Controller) and the performance monitoring unit (PMU) in Cortex-A9. Collected data were transmitted to PC via UART connection. The Xilinx Vivado 2013.3 was used with SDK for the experiments.

B. Memory Hierarchy and Coherence Support of Zynq-7000

As mentioned, Zynq-7000 has two fused masters (Cortex-A9). Each core has a separate 32KB L1 data cache and an instruction cache of the same size. The two processors share a unified 512KB L2 cache for instruction and data. Both L1 and L2 caches have the line length of 32-bytes. Zynq-7000 provides a master interface called accelerator coherency port (ACP). A master component designed in the PL can access the L2 cache through the ACP with 64-bit AXI interface. This AXI 64-bit port is channeled through Snoop Control Unit (SCU), providing an interface for the communication between the PL and the PS. The SCU maintains the data cache coherency between the two processors, the L2 cache, and system accelerators implemented through ACP interface. For the experiment, we implemented a master component in the PL section, which continuously injects burst read transactions. The malicious master is connected to SCU through the ACP interface, as shown in (Figure 2).



(Figure 2) Memory hierarchy in Zynq-7000

C. Hardware DoS Attack via ACP

1) *Description of Attack*

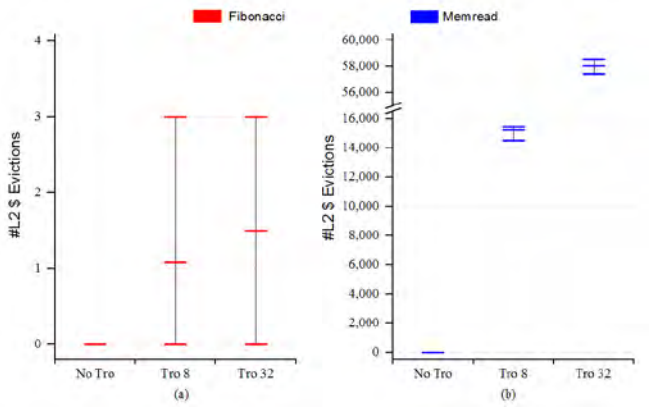
As explained in Section 3-B, continual read requests to a specific set of the L2 cache might result in repeated occurrence of cache line replacement not only in the L2 cache but in the L1 cache, if the caches are strictly inclusive. Once a line is evicted from the L1, CPU has to access memory to repopluate its cache. Since memory access takes a considerable amount of clock cycles, CPU has to sit and wait until the data is fetched.

2) *Implementation*

To effectively cause line evictions in the L2 cache, its structural characteristics should be taken into account in Trojan design. With 512KB 8-way set-associative structure, the size of each way is 64KB indicating that each way can accomodate upto 2048 lines. Thus, accessing an entire 1MB address space with a 32-byte stride will idealy make all 2048 lines (that is, whole 512KB L2 cache) be evicted assuming the cache supports true LRU as its replacement strategy. We first implemented a Trojan (*Tro8*), which accesses a 512KB memory space. But since the L2 cache in Zynq-7000 employs pseudo-random victim selection policy, we designed another Trojan (*Tro32*) which accesses a larger space (2MB) in order to increase the chance of L2 cache line replacement.

5. Evaluation and Analysis

We designed two synthetic benchmarks: Fibonacci and memread. Fibonacci takes an integer *n* as an argument and returns the *n*th element in the Fibonacci sequence by recursion. In our experiment we gave 40 as the input. Memread simply reads data from scattered memory spaces in a regular pattern. (Figure 4) shows the number of L2 cache line evictions while each of two synthetic benchmarks runs. We executed the benchmarks 10 times and computed the average with the minimum and maximum in the graphs. In (Figure 4) (b), the line eviction in the L2 cache dramatically increases from 0 to 15,218 on average, which would greatly disturb the CPU's memory access for the benchmark execution. One more noticeable thing is that there is a significant difference in the number of L2 cache evictions in the two benchmarks. We strongly believe that it comes from these two factors: execution time and memory access pattern. Fibonacci takes relatively a short amount of time compared to memread. When it comes to the memory access, memread is much more memory-intensive. To make sure the effect of our Trojan, we measured the number of read requests sent to L2 cache for the two benchmarks. The result is summarized in <Table 1>.



(Figure 4) The number of L2 cache eviction

<Table 1> The number of read requests to L2 cache

	Trojans		Reqs/Cycle
	No Tro	*Tro32*	
fibonacci	60	57,794,578	0.636
memread	311,137,489	1,010,678,207	0.643

Based on the number of L2 cache read requests, we derived the number of requests per cycle. The numbers in the second column with *No Tro* come from the legitimate CPU requests while executing the benchmarks. The numbers in the third column with *Tro32* report the combined effect of the legitimate requests and Trojan requests. As shown in the last column of <Table 1>, there is no notable difference in the number of L2 read requests per cycle between the two benchmark runs. It implies that *Tro32* injects malicious traffic consistently regardless of the characteristics of applications.

We then made some modifications in memread benchmark to shorten its runtime to be similar with that of fibonacci. <Table 2> shows the average number of L2 cache evictions so that the execution cycles are 90,862,698 for fibonacci and 90,594,632 for memread. Only 1.2 lines were evicted on average while executing fibonacci whereas there were 4,905 evictions in the case of memread. As *Tro32* injects malicious traffic consistently in both cases, this implies that the impact of *Tro32* is greatly influenced by the memory access pattern of applications.

<Table 2> Average number of L2 cache line evictions

	Trojans	
	No Tro	*Tro32*
fibonacci	0	1.2
modified-memread	0	4,905

This experiment shows that malicious traffic injections can incur line evictions in the L2 cache. This undesirable occurrence of line evictions can be a great threat as it might cause significant performance degradation. To investigate the subsequent impacts of evictions, our next plan is to build up a real embedded system with Linux and/or Androids ported on. Standard benchmarks such as SPEC 2006 are going to be used to measure the impact with metrics like the number of L1 cache evictions and execution time.

One more thing we are going to focus on is the countermeasures to mitigate the impact of the malicious injections of read traffic. Implementing a programmable register between the ACP slave port and the Trojan can be a way to avoid malicious injections of read transactions through the ACP. Although the implanted Trojan cannot be removed from the chip, the user can avoid the attack by configuring this register to block the injection of traffic from the malicious hardware. The effectiveness of this method will be evaluated in further studies.

6. Conclusion

This paper introduced the coherence-exploiting hardware Trojan which can be maliciously implanted in master components, and continuously injects memory read transactions. This type of attack can be made very insidiously as read transactions do not alter or crash the system. The result of our experiment shows the injected traffic causes the eviction of cache lines. And this reveals severe threats of the Trojan to the system performance. To mitigate the impacts of this attack, a programmable register can be added between the ACP slave port and its master. This register provides a way for the user to avoid injection of read traffic from the hardware Trojan, although the Trojan is not removed from the chip.

References

[1] X. Wang, M. Tehranipoor, and J. Plusquellic, ‘‘Detecting Malicious Inclusions in Secure Hardware: Challenges and Solutions,’’ Proc. IEEE Int’l Workshop Hardware-Oriented Security and Trust (HOST 08), IEEE CS Press, 2008, pages. 15-19.

[2] Tehranipoor, M; Koushanfar, F. IEEE DESIGN & TEST OF COMPUTERS; JAN-FEB, 2010; 27; 1; p10-p25I.S. Jacobs and C.P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G.T. Rado and H. Suhl, Eds. New York: Academic, 1963, pages. 271-350.

[3] N Tsoutsos, and M Maniatakos, “Fabrication attacks: Zero-overhead malicious modifications enabling modern microprocessor privilege escalation”. Emerging Topics in Computing, IEEE Transactions on (Volume:2 , Issue: 1), pages 81-93, March 2014.

[4] Y. Jin, N. Kupp, and Y. Makris, “Experiences in hardware trojan design and implementation.” In Proc. IEEE Workshop on Hardware-Oriented Security and Trust, pages 50–57, June 2009.

[5] J. A. Roy, F. Koushanfar, and I. L. Markov. Extended abstract: Circuit cad tools as a security threat. In Proc. IEEE Workshop on Hardware-Oriented Security and Trust, pages 65–66, June 2008.

[6] Intel Haswell, http://www.intel.com/.

[7] Sally Adee, “The Hunt for the Kill Switch”. IEEE Spectrum. Vol. 45, Issue 5, pp 34-39, May 2008.

[8] Celia Gorman, “Counterfeit Chips on the Rise”, IEEE Spectrum. May 2012.