Hardware Based Security Enhanced Direct Memory Access

Marcel Eckert, Igor Podebrad, Bernd Klauer

Hardware Based Security Enhanced Direct Memory Access

Marcel Eckert[2], Igor Podebrad[1], and Bernd Klauer[2]

[1] Group Security, Threats Defense
Commerzbank AG, Frankfurt am Main
igor.podebrad@commerzbank.com,
[2] Computer Engineering, Department of Electrical Engineering
Helmut Schmidt University, Hamburg
{marcel.eckert,bernd.klauer}@hsu-hh.de

Abstract. This paper presents an approach to prevent memory attacks enabled by DMA. DMA is a technique that is frequently used to release processors from simple memory transfers. DMA transfers are usually performed during idle times of the bus. A disadvantage of DMA transfers is that they are primarily unsupervised by anti malware agents. After the completion of a DMA activity the transfered data can be scanned for malicious codes. At this time the malicious structures are already in the memory and processor time is necessary to perform a malware scan. The approach presented in this paper enhances the DMA by a watchdog mechanisms that scans the data passing by and interrupts the processor after the detection of a malicious data or instruction sequence. Configurable hardware based on FPGAs is used to overcome the problem of frequently changing malware and malware signatures.
Key words: Hardware Security, FPGA, Direct Memory Access, Malware

1 Introduction

The security of modern computing systems is mainly based on software, such as anti malware agents or intrusion detection systems. Modern attack vectors consider hardware and software leaks for intrusion purposes. The main target today is software on all software abstraction levels in a computer. Hardware as a target is also moving into the focus of the "Dark Side" (configurable hardware is also infectable hardware) but this fact is not focus of this paper. Although traditional (software based) anti malware approaches are improving daily by enhanced snooping procedures and new malware signatures, Rutkowska has shown, that attacks are possible, which are undetectable by software [4].

Especially exploits taking advantage from the Direct Memory Access unit (DMA) showed the simplicity of malware injections directly into the main memory of computing systems, bypassing all software based security mechanisms. Rutkowskas attacks imposingly show, that the hardware engineering paradigm "Software writers should provide security; Hardware should just be as fast as possible" [2] is definitively outdated. To design secure systems in future, security

needs to become an issue to be adressed on all abstraction levels of computing systems.

A general introduction into the field of hardware based security is given in [6], with an excellent elaboration on DMA and related components like memory and interfaces in chapter 5. A comprehensive analysis of related work in the area of processor security is given in [1].

In the remainder of this paper we present our hardware based security enhancement for the DMA-functionality as an example for how to cut down selected hardware originated attack vectors. It's basic functionality is a tamper-proof hardware based, highly parallel executed snooping functionality on the data bus, that scans for signatures of malicious code structures interrupting if a signature is found.

2 Problem

DMA is a well known technique to release processors from time consuming workload caused by simple data transfers. The transfers are performed without supervision. Data and instruction are communicated between the memory as sink and source or between the memory and mass storage or interfaces. They reach their destination block-wise completely before beeing checked by anti malware agents. The DMA reports DMA completions by setting bits in status registers by interrupting the processor or by other status signals [5]. An anti malware agent can then check the result if the memory was the DMA target.

With the unsupervised DMA transfers stealth features can be implemented. Malicious code and data can be transferred. To complete success the attack pattern needs to launch the code before it has been checked by the anti malware agent.

Another opportunity to exploit the unsupervised DMAs is to infect the anti malware software directly.

3 Solution

3.1 Concept

The vulnerability of a system, based on the security leak imposed by the DMA-functionality will be solved by the introduction of a DMA-Watchdog (watchdog). The watchdog itself resides between the DMA-controller and the memory controller (of the main memory). The watchdog supervises the data part of the memory-bus with a number x of sensors (S_0 to S_x in figure 1).

The sensors provide a pattern matching functionality to identify malware. The detection algorithm of a single sensor can be complex in any order and is variable in principle. If one of the sensors is detecting his pattern it is signaling to the watchdog. The watchdog itself will now block the current DMA-transfer and signals the processor a "bad" transfer where it can be handed over to the operating system and appropriate software (e.g. anti virus software).

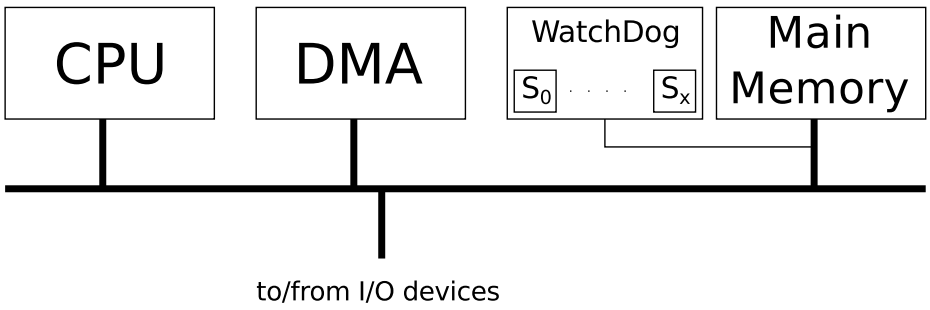

Fig. 1. watchdog residing between DMA-controller and main memory

The detection of different patterns is possible by the different sensors and is performed in parallel. For proving the effectiveness of our solution, the implemented proof of concept demonstrator is presented in the next section.

3.2 Proof of Concept

This section presents the proof of concept demonstrator for an hardware based DMA-Watchdog as described in the previous section. It is based on the *Partially Reconfigurable Heterogeneous System (PRHS)* framework as shown in subsection 3.2. The purpose of the proof on concept demonstrator is to prove the following theses:

1. A hardware based DMA-watchdog is able to detect malware infected DMA-to-Memory transfers.
2. There is no performance loss for the system processor if a hardware based DMA-watchdog is used.

PRHS Framework Before starting to work on hardware based DMA-watchdogs, it is necessary to have a freely configurable framework for investigations. Already available frameworks, e.g. Xilinx Microblaze, suffer one big problem: Their hardware is either hardwired (hardcores) or the available soft-cores have closed sources. To gain full flexibility for future research and development, it was necessary to have control even over the configuration and architecture of the hardware. Therefore the *PRHS* framework has been developed. It is presented in the remainder of this section.

The framework has been designed for *Field Programmable Gate Array (FPGA)* usage only, with the intention to have a platform for research and education. Hence it consists of several modules allowing reuse and adaptivity for different *FPGA*s and Boards (it has already been used for Spartan3, Virtex5, Spartan6, Virtex6, Virtex7 *FPGA*s and evaluation boards hosting such devices (i.e. ML505, ML605, VC707). All necessary hardware-sources (VHDL) have been implemented from scratch. Therefore the system is completely open. The software

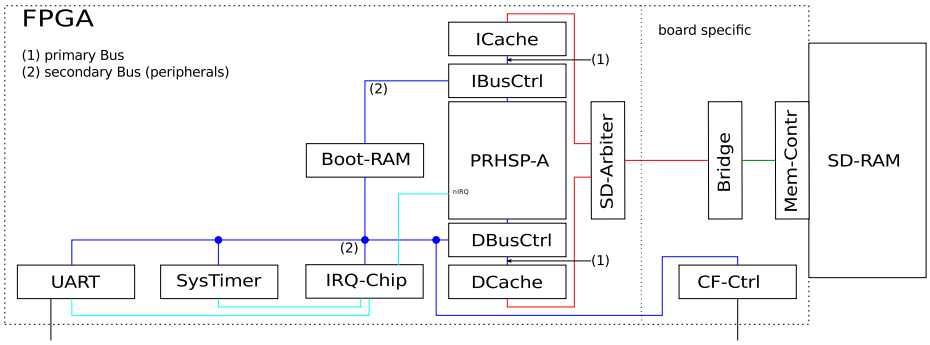

Fig. 2. schematic overview for base system of PRHS framework (hardware, detailed explanation of components is given in section 3.2)

parts are adapted Open Source projects (Linux as OS, gcc and ucLibC for Cross Compiler toolchain) and therefore also fully accessible.

Hardware For allowing flexibility, the *PRHS* framework comes with three fundamental systems:

- Small system: Combines the *PRHS Core - ARM Instruction Set (PRHSC-A)* with Block-Ram and an UART. This might serve as starting point for small embedded systems.
- Base system: Extends the small system to be able to run a customized Linux (*Linux for PRHS (L4PRHS)*) including an SD-RAM interface. Figure 2 gives a schematic overview.
- Reconfiguration system: Extends the base system with a partial reconfigurable area. (This feature is not used for the DMA-watchdog proof of concept demonstrator)

Description of the board independent components:

PRHSp-A A self implemented processor, consisting of a core, which is instruction set compatible with the ARM8 instruction set, a system co-processor and a memory management unit.
Boot-Ram On-chip Block Ram that contains the stage 1 boot-loader.
ICACHE/DCACHE Instruction/Data Caches, different implementation variants exist, ranging from a simple bridge mechanism to a 32k Cache.
IBusCtrl/DBusCtrl Bus controller to separate fast accesses to Caches/Memory (on primary Bus) and slow accesses to peripheral devices (on secondary Bus).
SD-Arbiter Bus Arbiter to prevent mixing of SD-RAM memory access.
SysTimers In-system programmable Timers.
UART Provides input/output functionality over a RS232 Line.

Between the caches and the SD-Arbiters, a fast, board and *FPGA* independent protocol for SD-RAM access is implemented. It is mapped by board specific components to the appropriate SD-RAM protocol of a board.

Description of the board specific components:

Bridge This device maps the *FPGA* independent SD-RAM protocol to the appropriate board specific SD-RAM protocol.

Mem-Contr Board specific SD-RAM memory controller.

CF-Ctrl Board specific (Compact) Flash controller. This device implements harddrive functionality.

Software The *PRHS* framework also contains a software part including the following components:

L4PRHS An adapted Linux kernel (version 3.8), including all device driver modules to run properly on the hardware presented in the previous section.

cross compiler toolchain gcc based toolchain (including uClibC as Standard Library) to compile the adapted Linux kernel and develop software for the Framework on a high-level language base.

3.3 Proof of Concept Demonstrator

The general architecture for our proof of concept demonstrator, based on the base system of the *PRHS* framework is given in figure 3.

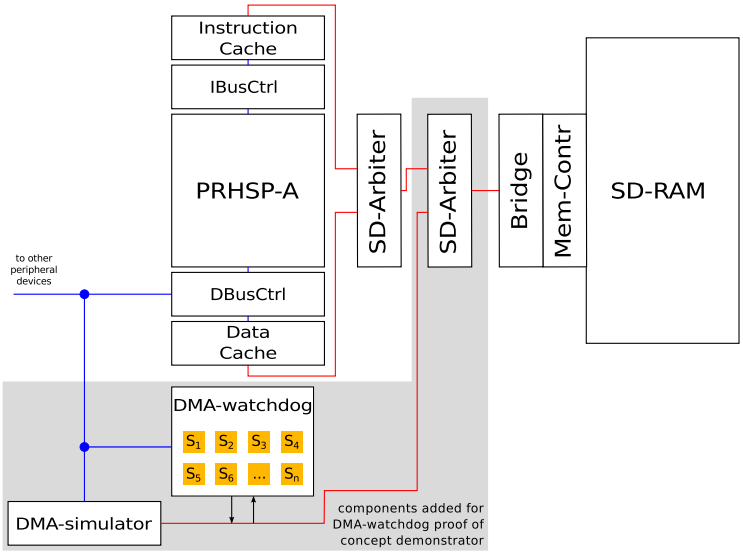

Fig. 3. general architecture for proof of concept demonstrator

To simulate and initiate DMA-functionality, a new device (DMA-simulator) has been added to the secondary data bus of the system. It is also connected to RAM via an SD-Arbiter. Its task is to copy the content of internal Block-Ram (8kByte, contents programmable via software) to a given RAM address (programmable via software) on demand. A write request to RAM contains 256 Bits (32 Bytes) of Data. The 8 Kbyte content of the Block RAM is therefore transfered with 256 write requests to RAM.

A DMA-watchdog as described in the previous section has also been implemented. The sensors implement only a very simple pattern matching algorithm without any fuzziness. It simply compares its programmed (via software) pattern (256 bit wide) with the actual data signal (256 bits wide) of the DMA-to-memory transfer.

As the watchdog and the sensors itself are working independently from the system processor by design, there is no performance loss for the processor. Nonetheless, performance measurement has been done with dhrystone[7] and ramspeed[3] benchmarks and has proven: watchdog functionality isn't consuming processor time.

Detection functionality has been tested with a proof of concept demonstrator based system including 40 sensors. The two possibilities, DMA data contains (positive test) or doesn't contain "bad" data (negative test) were extensively tested. In both cases, DMA data and sensors patterns were generated randomly. For the negative test cases, one of the sensor patterns was randomly selected and inserted at a random position in the DMA data at each iteration. For both cases 1.000.000 iterations were carried out. In all iterations of both cases, the watchdog worked correctly.

4 Evaluation

Our proposal has shown, that a security mechanism can be implemented on a very low level, which doesn't cost any processor performance by design, because it doesn't use any computational resources of the processor. The design of the memory controller and the DMA-controller are also not affected by the introduction of a DMA-watchdog. Nevertheless, additional resources are necessary to implement the sensors and the watchdog, which can bee seen as a very special kind of (co)-processor, consuming their own processing time.

The introduced approach is able to cut down only a certain part of the overall attack tree, namely the hardware related ones. This means that a hardware based solution is powerful but definitively not able to fulfill all requirements concerning detection at all levels. Nonetheless with our solution we are for the first time in the position to detect malicious code structures at a very low level in a tamper-proof way.

The proposed solution is "signature" based and therefore lacks the problem of actuality. Additionally, the number of detectable "signatures" is limited by the implemented number of sensors.

As mentioned in section 3 it would also be imaginable to implement a more complex behaviour based detection mechanism [8]. In this case, the sensors will only signal at the ”end” of a behavior. This results in data already transfered to RAM, but the entire transfer, or more exactly the ”final” transfer will be prevented.

The current solution lacks the problem of a static detection algorithm. This is related to the hardware implementation of this algorithms. Some already available commercial systems like Convey HC-series or Xilinx Zynq and the academic *PRHS* framework (section 3.2) introduce the possibility to modify hardware components at system runtime via (re)configurable hardware based on *FPGA*. These technologies offers the possibility to exchange the hardware based detection algorithms for our proposed solution.

5 Conclusion and future work

In this paper we presented our solution for the problem of unsupervised DMA-transfers. The proposed solution introduced a hardware based DMA-watchdog. A proof of concept demonstrator , based on an ARM-system running Linux, has also been implemented to prove the feasibility of our solution and to underline one of the big advantages: this approach doesn't consume processor performance at all.

For the future we want to introduce the ability to also modify the hardware based detection algorithm by means of partial and dynamic reconfiguration. This might include also a behavior based detection algorithm.

References

1. Siddhartha Chhabra, Yan Solihin, Reshma Lal, and Matthew Hoekstra. An analysis of secure processor architectures. *Transactions on Computational Science*, 7:101–121, 2010.
2. S. Gueron, G. Stronqin, J.-P. Seifert, D. Chiou, R. Sendag, and J.J. Yi. Where does security stand? new vulnerabilities vs. trusted computing. *Micro, IEEE*, 27(6):25–35, 2007.
3. Rhett M. Hollander and Paul V. Bolotoff. RAMspeed, a cache and memory benchmarking tool.
   http://alasir.com/software/ramspeed/, 2009.
4. Joanna Rutkowska. Beyond The CPU: Defeating Hardware Based RAM Acquisition.
   http://www.first.org/conference/2007/papers/rutkowska-joanna-slides.pdf, 2009.
5. Patrick Stewin and Iurii Bystrov. Understanding dma malware. In *DIMVA2012 Proceedings of the 9th Conference on Detection of Intrusions and Malware & Vulnerability Assessment*, Lecture Notes in Computer Science, Vol. 7591, pages 21–41. Springer, 2012.
6. Shuangbao Wang and Robert S. Ledley. *Computer Architecture and Security: Fundamentals of Designing Secure Computer Systems*.
7. Reinhold P. Weicker. Dhrystone: a synthetic systems programming benchmark. *Commun. ACM*, 27(10):1013–1030, October 1984.

8. D. Ye, M. Moffie, and D. Kaeli. A Benchmark Suite for BehaviorBased Security Mechanisms.
http://www.ece.neu.edu/groups/nucar/publications/SSATTM05.pdf, 2005.