Design of a Generic Security Interface for RISC-V Processors and its Applications

Hyunyoung Oh, Junmo Park, Myonghoon Yang, Dongil Hwang and Yunheung Paek
ECE and ISRC, Seoul National University
Seoul, Republic of Korea
{hyoh, jmpark, mhyang, dihwang}@sor.snu.ac.kr, ypaek@snu.ac.kr

Abstract— **In this paper, we propose the design of a generic security interface for RISC-V. This interface increases flexibility of security modules by creating an environment that can operate independently on a host processor. We also present an application using this interface for the memory protection. To check the feasibility of our idea, we implement an early prototype where a RISC-V processor is connected with the proposed hardware components using our interface. The empirical results show that our security interface has enabled a security module operated independently of the processor with no performance and low area overhead.**

Keywords; RISC-V, security interface, memory protection

I. INTRODUCTION

As the security threats become increasing steadily, several studies have proposed various security solutions that protect the underlying system against security attacks. Most of the early security solutions were implemented in software, and often suffered from a substantial performance overhead. To reduce the overhead, many researchers have proposed hardware-based security solutions. Conventional hardware-based solutions, however, have a drawback in that they cannot be utilized extensively since they require a permanent modification to the microarchitecture of the host processor [1]. Thus, without a permanent modification, it becomes necessary to have an interface that enables security modules to be used independently of a host processor. In this sense, we have developed a *security interface* that can extract the internal information of the host processor needed for security modules. This interface is connected to components of the pipeline structure inside the RISC-V Rocket core, which can obtain the internal information to recognize the control flow and data access patterns of the host. In addition, we present an application that can effectively perform memory protection by using the security interface.

II. ASSUMPTION AND THREAT MODEL

In this work, we target embedded systems employing processors available in today's market, which cannot afford MMUs for sophisticated virtual memory managements. We assume that the target system comes in the form of an SoC and therefore any external hardware can be attached during implementation.

As our adversaries, we assume that untrusted software modules can be installed in the target devices. These untrusted

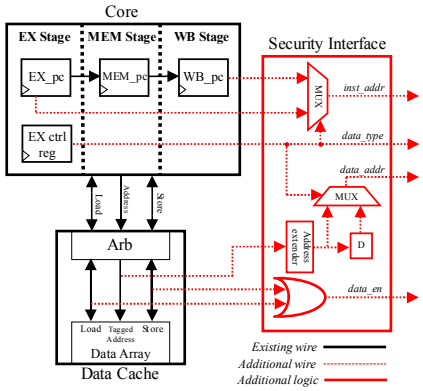

Figure 1. Information extraction through Security Interface

modules can lead to compromising an entire program when (1) the modules have vulnerabilities which can be exploited by an attacker or (2) the modules themselves are malicious. This is due to the fact that, without address protection between modules, a software module would be able to corrupt the data used by another module.

III. SECURITY INTERFACE

Security interface, as shown in Fig. 1, extracts the RISC-V processor's internal information such as instruction/data address and data access pattern. The security interface probes pipeline stages to get an instruction address and internal signals of data cache to get a data address. But synchronizing between those extracted addresses is not trivial. We observed that there exist some delay relations between them and founded that those relations are correlated to control signal of pipeline stages. Thus, we carefully multiplex each delayed address by using the control signal as a selector. And we also create the enable signal to indicate the validity of the output.

IV. EXEMPLARY APPLICATION FOR SECURITY: MEMORY PROTECTION

In this section, we propose a security module integrated together the security interface. This module, called *memory region protector* (MRP), supports the security function that is to establish the software isolation environment by checking the occurrence of illegal memory accesses. Fig. 2 shows the overall architecture of our MRP. To detect an illegal memory access, MRP utilizes a register file, called *access permission*

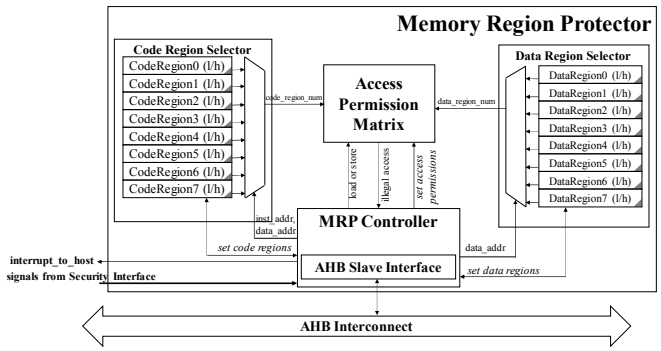

Figure 2. Memory Region Protector architecture

matrix that indicates legal access permission. The access permission matrix maintains two types of relationship: code versus code region and code versus data regions, as described in TABLE I. After receiving an instruction address and a data address via the security interface, code region selector and data region selector find each region index respectively. And then as considering those region indexes with the received access type, MRP checks whether or not the access can be found in the access permission matrix. If not, the MRP controller interrupts the host CPU to report the illegal memory access.

During the boot-up, the code stored in PROM initializes the access permission matrix. When the code stored in PROM is executed, the processor launches an instruction. Over the system bus, the instruction is encoded by the bus protocol. On receiving the command, the *APB slave interface* in the *MRP controller* module decodes the command and sends the decoded information to the access permission matrix for configuring the registers. The MRP controller also updates the registers to define protected code and data regions in the *code region selector* and *data region selector*, respectively. To define the range of each code or data region, two registers are provided to set the low and high addresses of the region.

V. EXPERMINETAL RESULTS

In our prototype, we use the Xilinx Zynq-7000 board and use a version 1.7 of RISC-V Rocket core parameterized by FPGA configuration DefaultFPGAConfig, as the host processor. In our implemented MRP, the number of code and data region are both configured to be eight. The bus compliant with AMBA AHB2 protocol [2] is used to interconnect all the modules in our prototype system.

We quantified the resources necessary for our hardware components in terms of lookup tables for logic (LUTs) and FFs with synthesis constraint of 33MHz clock rate. The design statistics show that, compared to the baseline Rocket core, our components incur the resource overhead of 6.07% and 2.96% for FFs and LUTs, respectively. We also estimated the gate count of our hardware components using Synopsys Design Compiler. With a commercial 45nm process library, the total gate-count of the proposed modules is 16,586(1,712 gates for the security interface, 12,828 gates for MRP and 2,046 gates for the access permission matrix).

The security interface incurs zero performance overhead because it extracts the internal information without changing the critical path of the host CPU. Using the interface, our MRP runs in parallel with the functional execution of the host. Hence, the access permission check of MRP also does not impact the performance of the target system.

TABLE I. ACCESS PERMISSION MATRIX

OBJECT / SUBJECT	Code Region0	Code Region1	Code Region2	Data Region0	Data Region1	Data Region2
Code Region0	RX	-	R	RW	-	RW
Code Region1	-	RX	-	-	R	-
Code Region2	-	R	RX	RW	R	RW

Access Permissions

R : Readable, W : Writable, X : eXecutable
- : No access is permitted

As a hardware component that plays a role similar to MRP designed in this paper, there are MPU [3] of ARM Cortex series and PMP [4] which can be optionally attached to RISC-V processor recently. However, MRP can arbitrarily set the size of the area compared to these, providing more flexibility in setting up the system. Also, MRP provides at least equal or less overhead because it requires the same amount of registers to set the lower and upper bounds of an area.

VI. CONCLUSION

We present a security interface that makes it possible to extract the internal information of RISC-V needed for the security modules. We have also demonstrated an exemplary application for security called MRP that can effectively implement a memory protection mechanism using a security interface. MRP plays the role of deciding the existence of invalid memory accesses by observing the pattern of data transfers outside the host. The experimental results showed that MRP which is our hardware-based solution successfully operated using a security interface and provides enhanced memory protection with low area and virtually no performance overhead. As a result, the proposed generic security interface has the potential to design more advanced security modules based on the internal information of RISC-V that has been successfully extracted.

ACKNOWLEDGMENT

This work was partly supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2018-0-00230, Development on Autonomous Trust Enhancement Technology of IoT Device and Study on Adaptive IoT Security Open Architecture based on Global Standardization [TrusThingz Project]) and supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIT) (No.2017-0-00213, Development of Cyber Self Mutation Technologies for Proactive Cyber Defense) and supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (NRF-2017R1A2A1A17069478)." and supported by IDEC.

REFERENCES

[1] Patrick Koeberl, Steffen Schulz, Ahmad-Reza Sadeghi, and Vijay Varadharajan. 2014. TrustLite: a security architecture for tiny embedded devices. In Proceedings of the Ninth European Conference on Computer Systems. ACM, 10.

[2] ARM 1999. AMBA Specification. ARM.

[3] ARM 2014. ARM Cortex-M7 Processor. ARM.

[4] Electrical Engineering and Computer Sciences University of California 2016. The RISC-V Instruction Set Manual Volume II: Privileged Architecture. Electrical Engineering and Computer Sciences University of California.