Experiment on Replication of Side Channel Attack via Cache of RISC-V Berkeley Out-of-Order Machine (BOOM) Implemented on FPGA

Anh-Tien Le*, Ba-Anh Dao*, Kuniyasu Suzaki†, Cong-Kha Pham*

Anh-Tien Le, Ba-Anh Dao, Kuniyasu Suzaki, Cong-Kha Pham
*The University of Electro-Communications (UEC), Tokyo, Japan
†Technology Research Association of Secure IoT Edge application based on RISC-V Open architecture (TRASIO)
Email: {leanhtien,daobaanh}@vlsilab.ee.uec.ac.jp, k.suzaki@aist.go.jp, phamck@uec.ac.jp

***Abstract*—In this work, we start by describing the implementation and benchmark of the BOOM processor (RISC-V Berkeley Out-of-Order Machine) on an FPGA board ZC706. Then compare the result with the RISC-V in-order scalar processor the Rocket Core. Subsequently, we demonstrate a side-channel attack that exploits some characteristics of an Out-of-Order processor in general and the BOOM processor in particular. The experiment would be a premise for constructing a custom heterogeneous processor.**

***Index Terms*—RISC-V, side-channel attack, Out-of-Order processor**

I. Introduction

In recent years, RISC-V [1] is developed as a new instruction set architecture that is available under open, free and non-restrictive licenses. It is designed to encourage computer architecture researches, educations and be supported by large industries including chip and device makers. The RISC-V ecosystem enables a new level of innovation in processor architecture that will be a crucial driver for the needed gains in performance and power efficiency over the next decade. One of the advantages of RISC-V is its simplicity and much smaller than other commercial general-purpose ISAs but still captures significant details. It also supports both 32-bit and 64-bit address space variants for applications, operating system kernels and hardware implementations. Consequently, both in-order and out-order processors have been studied and designed based on the RISC-V architecture.

The Out-of-Order processor provides a dynamically rescheduling mechanism that utilizes the full chip's resources to improve the computer's performance. The University of California, Berkeley, has been developing and maintaining an open-source synthesizable parameterized out-or-order RISC-V processor [2], called "BOOM". It contributes a practical environment for researching the execution and security of the RISC-V micro-architecture. In Fig.1, the BOOM pipeline is divided into ten stages. The fetch stage gets instructions from the processor's memory and pushes into the Fetch Buffer, a FIFO queue. From this queue, the commands are pulled out and converted into the "micro-ops" in the decode stage. The rename stage converts the ISAs into "physical" register specifiers. In the dispatch stage, the "micro-ops" are written into the Issue stage including a queue which they would stay on and wait for all of the instructions are ready. The Out-of-Order execution begins with the "micro-ops" loading their register operands from the Physical Register File and then be executed to write/read the memory. The efficiency of the BOOM's architecture will be evaluated in the latter part of this paper.

In contrast, the Rocket core [3] is a micro-architecture of an in-order scalar RISC-V processor with a 32-bit instruction format. It contains a six-stage pipeline, has one integer ALU and an optional FPU. An accelerator or co-processor interface, called RoCC, is also provided. It is able to produce a set of processor core designs, with different configuration parameters. The Rocket core is also provided with a library of processor components, which make it suitable for study to design a custom RISC-V processor.

In early 2018, the computer industry has convulsively discussed two newly discovered serious security flaws that have been identified within computer processors called meltdown [4] and spectre [5]. These dangerous security holes permit adversaries to abduct the critical information which could currently be processed on the memory of modern high-performance processors. This information might include a private key from a cryptography program performing encryption or a lot of user's private information such as passwords, messages or important documents. On the one hand, Meltdown exploits the vulnerability in Intel processors. They are compromised when implementing the Out-of-Order execution which could allow the hackers to get through the hardware-wall between user-level software and the core memory of the computer. On the other hand, the spectre takes advantage of the speculative execution mechanism of many modern processors to delude an error-free application into giving up the private information. Since the Berkeley Out-of-Order processor is an open-source project, it is undoubtedly a convenient system to inspect and demonstrate those attacks to evaluate the Out-of-Order architecture security [6]. The experiments will be presented in part III of this paper.

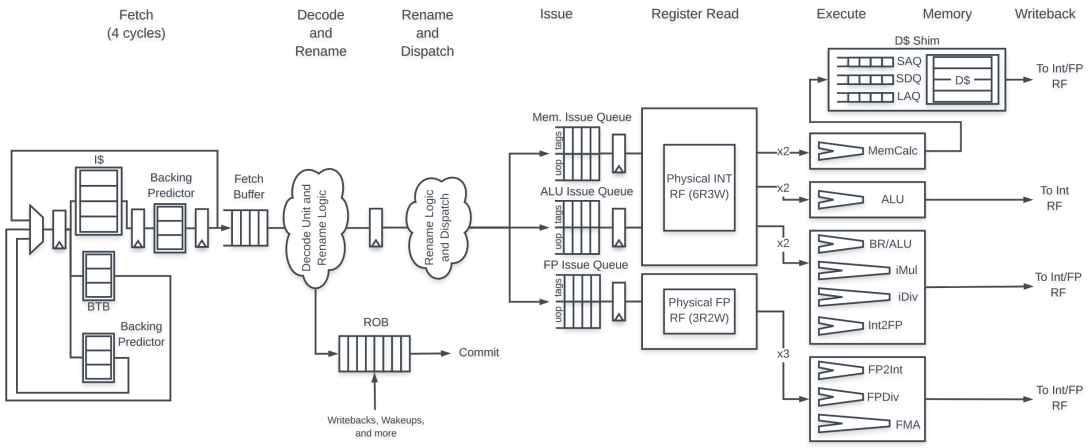

Fig. 1: BOOM pipeline [7]

II. Implementation of BOOM on FPGA and Benchmarks

In our works, we used an FPGA development board called Xilinx SoC ZC706 to implement, debug, assess and benchmark the performance of the BOOM processor. This evaluation board is based on the Zynq®-7000 SoC developed by Xilinx company [8]. It combines both programmable FPGA circuitry and an ARM-based processor. The BOOM processor is implemented on the board and booted with the SW11 SD Boot mode. Firstly, a simple operating system would be booted and run on the ARM processor. For communicating between the host ARM machine and the RISC-V BOOM core, a front-end server library is demanded, which also provides an interface to execute RISC-V programs using HTIF (Host/Target Interface). The program executes through ELF (Executable and Linkable Format) format.

We generated a two-wide single-core BOOM configuration using the rocket-chip generator. The ”two-wide” means that the processor will write up to two instructions into the ”issue queue”. Also, an 64 bits in-order rocket core’s configuration was built and generated using the same method. Both processors are implemented on the same FPGA broad to give them a fair evaluation. These configurations are loaded from the SD card as well as the interface to interact with the RISC-V cores and a root system compressed file which is load into RAM before executing.

The method to measure the processor’s performance is the Coremark [9]. Developed by the Embedded Microprocessor Benchmark Consortium (EEMBC), it is a leading standard benchmark for CPU cores, replacing the Dhrystone. Coremark is frequently used to assess the performance of the central processing unit and embedded micro-controllers. The Coremark’s score describes the number of iterations per second with seeds of 0,0,0x66, and the buffer size of 2000 bytes in total.

We have compiled the Coremark’s source-code using gcc6.1.0 -O2 of the RISC-V gnu toolchain to benchmark and compare the performance of both the Out-of-Order processor, BOOM, and the in-order processor, Rocket Core. As a result shown in Table I, the performance of the BOOM core is half as much again as the in-order process. This outcome could be justified due to the natural character of an Out-of-Order processor. This brings a massive favour on this processor when choosing a suitable architecture to construct a new modern processor.

TABLE I: Coremark Result

	Iterations/s
RV64 Boom two-wide	3,737
RV64 Rocket	2,181

III. Replication of Side Channel Attack via Cache

Despite inheriting the efficiency of an Out-of-Order processor, the BOOM’s micro-architecture contains some fundamentals which provide a suited environment for speculative execution attacks. Firstly, to optimize the computer system, a speculative execution technique is enabled in the processor’s architecture which is an ordinary design paradigm of the modern Out-of-Order processor. During the speculative execution process, the fetch stages issue a predicted instruction guided by a branch predictor unit. This part uses two stages of branch prediction: a next-line predictor, NLP, and a backing predictor, BPD [7]. If a mispredicted branch happens, the register which renames and reorders the structures in the next stages will allow the pipeline to recovery from the misspeculations. Regardless, visible effects still appear in which side-channel attacks could exploit to obtain private information from those misspeculated steps.

There are many types of spectre attacks that exploit different characteristics of the processor: Pattern History Table, Branch Target Buffer, Return Stack Buffer and Store To Load [10]. All of them manipulate the prediction mechanism of the micro-architectural.

In this experiment, we performed an attack on the implemented BOOM configuration on the ZC706 FPGA by focusing on branch misprediction. Suppose the program, which the

```
root@zynq:~# ./fesvr-zynq sd/boom-attacks/bin/spectre.riscv
Bound Check Bypass - Spectre Attack
m[0x0x80002718] | sceret_char(S) | guess_char(hits,score,value) 1.(3, 83, S)
m[0x0x80002719] | sceret_char(e) | guess_char(hits,score,value) 1.(9, 101, e)
m[0x0x8000271a] | sceret_char(c) | guess_char(hits,score,value) 1.(7, 99, c)
m[0x0x8000271b] | sceret_char(r) | guess_char(hits,score,value) 1.(8, 114, r)
m[0x0x8000271c] | sceret_char(e) | guess_char(hits,score,value) 1.(8, 101, e)
m[0x0x8000271d] | sceret_char(t) | guess_char(hits,score,value) 1.(9, 116, t)
m[0x0x8000271e] | sceret_char( ) | guess_char(hits,score,value) 1.(10, 32,  )
m[0x0x8000271f] | sceret_char(K) | guess_char(hits,score,value) 1.(8, 75, K)
m[0x0x80002720] | sceret_char(e) | guess_char(hits,score,value) 1.(8, 101, e)
m[0x0x80002721] | sceret_char(y) | guess_char(hits,score,value) 1.(8, 121, y)
m[0x0x80002722] | sceret_char( ) | guess_char(hits,score,value) 1.(10, 32,  )
m[0x0x80002723] | sceret_char(t) | guess_char(hits,score,value) 1.(9, 116, t)
m[0x0x80002724] | sceret_char(o) | guess_char(hits,score,value) 1.(8, 111, o)
m[0x0x80002725] | sceret_char( ) | guess_char(hits,score,value) 1.(10, 32,  )
m[0x0x80002726] | sceret_char(t) | guess_char(hits,score,value) 1.(6, 116, t)
m[0x0x80002727] | sceret_char(e) | guess_char(hits,score,value) 1.(7, 101, e)
m[0x0x80002728] | sceret_char(s) | guess_char(hits,score,value) 1.(8, 115, s)
m[0x0x80002729] | sceret_char(t) | guess_char(hits,score,value) 1.(8, 116, t)
m[0x0x8000272a] | sceret_char( ) | guess_char(hits,score,value) 1.(10, 32,  )
m[0x0x8000272b] | sceret_char(B) | guess_char(hits,score,value) 1.(7, 66, B)
m[0x0x8000272c] | sceret_char(O) | guess_char(hits,score,value) 1.(7, 79, O)
m[0x0x8000272d] | sceret_char(O) | guess_char(hits,score,value) 1.(8, 79, O)
m[0x0x8000272e] | sceret_char(M) | guess_char(hits,score,value) 1.(7, 77, M)
m[0x0x8000272f] | sceret_char( ) | guess_char(hits,score,value) 1.(10, 32,  )
m[0x0x80002730] | sceret_char(a) | guess_char(hits,score,value) 1.(8, 97, a)
m[0x0x80002731] | sceret_char(t) | guess_char(hits,score,value) 1.(8, 116, t)
```

Fig. 2: Spectre attack log.

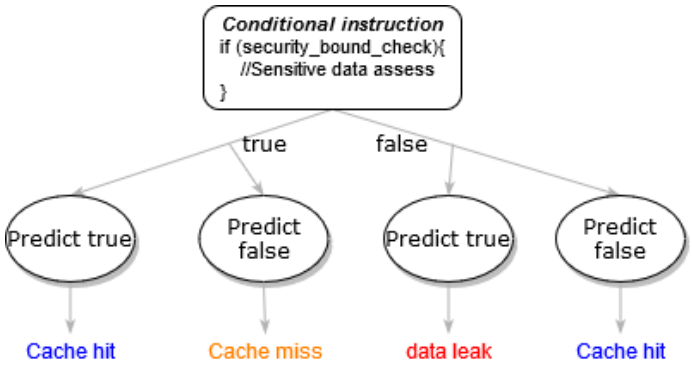

Fig. 3: Bound check bypass attack.

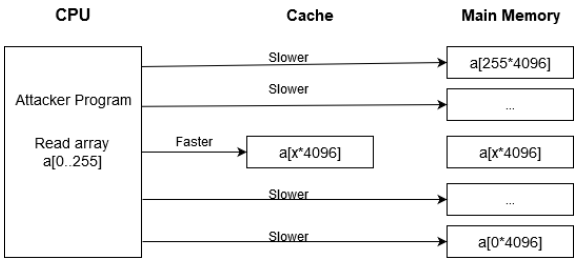

Fig. 4: Effect for side-channel attack.

adversary plan to exploit, includes conditional instruction with a bound check to prevents the processor reads other private information in memory.

```
if (x < size)
{
    key = key + 10;
}
```

The conditional branch is trained to execute an instruction that assesses the crucial data. Typically, the above code example would be executed in sequential order. However, to optimize the performance, the CPU uses the Out-of-Order execution and tries to predict the result of the bound check ($x < size$). When the value of *size* appears, the CPU will compare the predicted result with the actual one. If the result is the same, it would be a cache hit which gains the performance supremely. In contrast, the CPU reverts and the predicted result with its outcome execution would be removed, which is called a cache miss. However, most of the modern CPU, includes BOOM, forgets the effect caught by the caching process in the Out-of-Order execution. That makes it be endangered by the side-channel attack technique.

Return to the attack scenario, a false value will then be given to the program to fail the bound check. However, before that, the speculative execution has already retrieved the sensitive data to the cache as it predicted the bound check would give

a true value. Because the secret data has been revealed, the attacker could scan and time the memory to achieve the value. This method follows the FLUSH+RELOAD attack technique pattern.

The FLUSH+RELOAD technique is performed based on observing the side-effect of the processor's memory when executing programs. There is a huge difference in speed when the data needed by CPU is already fetched in cached (cache hit) with when the CPU has to go to the main memory to achieve the information. First, the adversary flushes shared address in the memory cache, which is the L1 cache in the BOOM process. Then the spy waits for the victim function access the sensitive data. Finally, it reloads to find out which element's loading time is faster than others. Consequently, that is the earlier accessed element. Because most of the security system depends on the conditional instructions to authenticate the permission to access sensitive data, this type of attack could danger many programs containing the private user's information.

An experiment has been implemented, compiled using the RISC-V toolchain and executed on the BOOM core. Firstly, we test the scenario on the software simulations of this processor generated by Verilator. The environment is quickly set up using the Chipyard, which is an agile Chisel-based SoC design framework [11]. After that, the experiment is loaded and run on FPGA board ZC706. The result is displayed in the Fig.2 as the log contains the actual character from the secret string and the guessed character achieved after the attack progress. Perceptibly, the adversary program has successfully managed to obtain the full private string. This proved that this RISC-V Out-of-Order architecture's security would be exposed to spectre attack.

IV. A heterogeneous multi-core processor

As we could see, the above attack exploits the branch prediction mechanism of the Out-of-Order BOOM core. The six-stage pipeline of the Rocket core also contains a front-end with branch prediction, which is served and manageable by a branch target buffer, a branch history table and a return address stack [11]. However, the Rocket Core does not issue data-memory accesses speculatively [12] and Out-of-Order execution. Therefore, the RISC-V In-Order processor would not be affected by above side-channel attack scenario.

Both Rocket and BOOM processor have their advantages and disadvantages. A combination of these two micro-architectures might be an appropriate solution. Fortunately, they were developed to use the same open-source Rocket Chip SoC generator. This generator is written in Chisel that allows to configure and create RTL describing a complete SoC design. Its parameters include: number of cores, instantiation of floating-point units, vector units, cache size, number of TLB entries, the width of off-chip I/O and more. Accordingly, a blended design of a multi-core processor merge BOOM and Rocket is feasible. There would be many modifications need to be processed to avoid conflicts, for example, the hart id of the core. With this heterogeneous multi-core processor, average calculations would be executed by Out-of-Order part to maximize the performance, and the Rocket Core could take care of sensitive tasks.

V. Conclusion

This experiment has successfully implemented and studied the BOOM processor on a physical FPGA. It has also proved that the in-order processor's performance, the Rocket Core, was outplayed by the Berkeley Out-of-Order processor. This result has transnational relation with the previous experiments [2]. However, the BOOM's architecture has made it become a convenient mark for speculative execution attacks. Using the side-channel attack technique that exposes the caching effect when performs the Out-of-Order execution, user's private information could be alarmingly leaked. Also, this work shows the benefits of an open-source RISC-V architecture for designing a custom processor and researching hardware security.

The BOOM processor possesses important qualities for a custom high-speed processor. The problem is its vulnerability with spectre attacks. In contrast, Rocket core's architecture does not speculatively provide data-memory accesses. Therefore, it becomes considerably suitable for securing private and vital information while being processed by user-level programs. A heterogeneous multi-core processor that combines both of these designs could manage and enhance distinct advantages of the Out-of-Order and in-order processors.

Acknowledgement

This paper is based on results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

References

[1] A Waterman, Y Lee, R Avizienis, H Cook, D Patterson, K Asanovic, "The RISC-V Instruction Set," Poster at the Symposium on High Performance Chips (HotChips-25), Stanford, CA (August 2013).
[2] C Celio, K Asanović, D Patterso, "The Berkeley Out-of-Order Machine (BOOM): An Industry-Competitive, Synthesizable, Parameterized RISC-V Processor", Technical Report UCB/EECS-2015–167, EECS Department, University of California, Berkeley (Jun 2015).
[3] Ben Keller, "RISC-V, Spike, and the Rocket Core", CS250 Laboratory 2 (Version 091713)
[4] Lipp, Moritz et al. "Meltdown: Reading Kernel Memory from User Space." USENIX Security Symposium (2018).
[5] Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., Andyarom, Y. "Spectre attacks: Exploiting speculative execution", arXiv:1801.01203 (2018).
[6] González, Abraham Martínez. "Replicating and Mitigating Spectre Attacks on a Open Source RISC-V Microarchitecture." (2019).
[7] C Celio, J Zhao, A, Gonzalez, B Korpan, "RISC-V-BOOM Documentation" (Version 010420)
[8] Xilinx, "Zynq-7000 SoC ZC706 Evaluation Kit - Getting Started Guide" (2018)
[9] Coremark EEMBC Benchmark," https://www.eembc.org/coremark/.
[10] C Canella, J V Bulck, M Schwarz, M Lipp, B v Berg, P Ortner, F Piessens, D Evtyushkin, D Gruss,"A Systematic Evaluation of Transient Execution Attacks and Defenses", arXiv:1811.05441 (2019)
[11] Berkeley Architecture Research, "Chipyard Documentation", (Version 101619)
[12] "Building a more secure world with the risc-v isa." https://riscv.org/2018/01/moresecure-world-risc-v-isa/.