Hardware and Software Co-Verification from Security Perspective

Kejun Chen*†, Qingxu Deng*,Yumin Hou†, Yier Jin†, and Xiaolong Guo‡
*Department of Computer Science and Engineering School, Northeastern University
†Department of Electrical and Computer Engineering, University of Florida
‡Department of Electrical and Computer Engineering, Kansas State University
{kejunchen, hou.yumin}@ufl.edu, dengqx@mail.neu.edu.cn, yier.jin@ece.ufl.edu, guoxiaolong@ksu.edu

***Abstract*—Attacks which combine software vulnerabilities and hardware vulnerabilities are emerging security problems. Although the runtime verification or remote attestation can determine the correctness of a system, existing methods suffer from inflexible security policy setup and high performance overheads. Meanwhile, they rarely focus on addressing the threat in the RISC-V architecture, which provides an open Instruction Set Architecture (ISA) of the processsor. In this paper, we propose a comprehensive software and hardware co-verification method to protect the entire RISC-V system in the runtime. The proposed method adopts the Dynamic Information Flow Tracking (DIFT) framework to implement a new Verifier and Prover security architecture for supporting runtime software and hardware co-verification. We realize a FPGA prototype on the Rocket-Chip, an RISC-V open-source processor core. The framework is implemented as a co-processor which do not change the architecture of main processor core and the new security architecture can be integrated with other RISC-V processors.**

I. Introduction

Software-only attacks, like return oriented programming (ROP), jump oriented programming (JOP) and other runtime attacks are always prevalent. Advanced software based attacks tend to use computer architecture vulnerabilities rather than software-only vulnerabilities. In [1], the authors use speculation vulnerabilities in modern processor core to launch a series of attacks, like branch predictor (BP), branch target buffer (BTB), and return stack buffer (RSB). Besides, the authors in [2] exploit the vulnerability in out-of-order (OOO) processor to leak sensitive information by launching side-channel based attacks.

In the meantime, as an open-source ISA, RISC-V provides more powerful support for low-power and high-performance processor designs. The current RISC-V standard version supports 16-bit, 32-bit, and 64-bit instructions. RISC-V based processor core can support many flexible architecture design and help reduce the hardware overhead. Although many frameworks are developed to resist the above attacks, there are few works proposed for addressing those threats in the RISC-V architecture by now.

Dynamic information flow tracking (DIFT) and remote attestation (RA) are two techniques utilized for protecting the hardware in the runtime effectively. Various DIFT designs are introduced [3]–[7]. Commonly, the software-based DIFT framework causes high performance overhead because of tag storage and complex tag computations. In contrast, hardware-based framework will speed up the process of tag computation. But it cannot provide flexibility as software-based solutions. The authors in [3] present a flexible hardware DIFT framework with programmable interface. Users are able to customize their own security policies by programming the tag operation rules. But it needs to modify the main processor and introduces high hardware overhead. How to make a balance between the functionality of DIFT and performance overhead becomes a difficult problem.

Existing RA schemes [8]–[10] provide a trusted execution environment to ensure the secure execution on the victim device. It is necessary to collect the device's information for detecting the potential vulnerabilities or attacks during the device's execution. Therefore, a Verifier-Prover model is introduced to make a trusted actor (Verifier) measuring the victim devices' (Prover) state at runtime. Specifically, the Verifier requests Prover's runtime state and the Prover returns the attestation report. This Verifier-Prover model can be applied in low-end embedded devices. However, few RAs are proposed to protect RISC-V architecture.

To overcome the above mentioned challenges, this paper presents a runtime verification framework which implements the DIFT in a remote attestation model. In the framework, a co-processor is added to realize the flexible functionality of DIFT. Users can customize security policy to enforce security checking rules on the main RISC-V processor core. Meanwhile, the whole co-processor is designed as the Verifier while the main processor core is treated as the Prover. The Verifier analyzes whether the current state of Prover is legal. The co-processor can be easily attached to the system bus without modifying the whole architecture of main processor core. The main contributions of this paper are as follows:

- We propose a runtime co-verification framework to protect the entire computing system, which runs an untrusted software on a vulnerable RISC-V processor.
- As the key part of the framework, a DIFT security mechanism is implemented as a co-processor and performs security checking. The security checking policy can be customized by users. To the best of our knowledge, it is the first off-chip DIFT architecture developed for protecting a RISC-V system.
- We realize the proposed security architecture as a FPGA prototype. The security policies are validated by defending a buffer overflow attack on a Rocket-chip.

The rest of the paper is organized as follows: In section

2332-5674/19/$31.00 ©2019 IEEE
DOI 10.1109/MTV48867.2019.00018

II, we introduce the threat model and mention previous works on DIFT and runtime verification. In Section III, we provide design details of the proposed new architecture and the security checking rules. Section IV presents demonstrations of the proposed framework by detecting the buffer overflow attack. Final conclusions are drawn in Section V.

II. Background

In this section, the attack model of this paper will be introduced. We also present the background of the RISC-V standard and the Rocket-Chip Platform, a popular implementation of the RISC-V. The existing works of RA and DIFT will also be discussed.

A. Attack Model

In this paper, we assume that an untrusted application/software runs on top of the hardware platform with security vulnerabilities, e.g., modern processors. Hardware vulnerabilities can be utilized by the untrusted application to infer secrets or perform malicious modifications. For instance, if hardware-based protection mechanisms are not implemented to prevent insecure information flows, untrusted software programs may perform any action including reading from all possible sources of labelled or sensitive data, propagating labelled data to all parts of the processor, as well as writing sensitive data through all insensitive inputs/outputs.

B. RISC-V Instruction Set Architecture and Rocket-Chip

RISC-V is an open-source instruction set architecture proposed by the UC Berkeley [11]. The design of the RISC-V instruction set takes the small, fast, low-power reality into consideration. Meanwhile, it does not aim at specific micro-architectures. The RISC-V basic instruction set contains only more than 40 with dozens of other modular extension instructions. The RISC-V ISA provides a flexible modular design, allowing users to flexibly select different modules to combine to satisfy the requirements of their own customized devices.

Rocket-Chip [12] is an open-source System-on-Chip (SoC) generator. It includes a paramaterizable scalar processor with a 5-stage pipeline using the RISC-V instruction set. It leverages the Chisel hardware construction language to support the paramaterizable design. Therefore, the whole SoC can be customized and parameterized easily.

C. Dynamic Information Flow Tracking

DIFT is a technique to detect vulnerabilities or attacks in the computer system by propagating and tracking the tag. A RISC-V based DIFT framework is proposed in [14] which supports the tagged memory and enforces the efficient isolation between data and code. This framework can be deployed on low-end embedded devices. In [15], Raksha et.al. support more flexible operation, by providing different configuration registers to customize the tag propagation rules and checking rules. The whole design is implemented in a processor core. [16] designs a DIFT architecture based on a RISC-V open-source core, PULPino

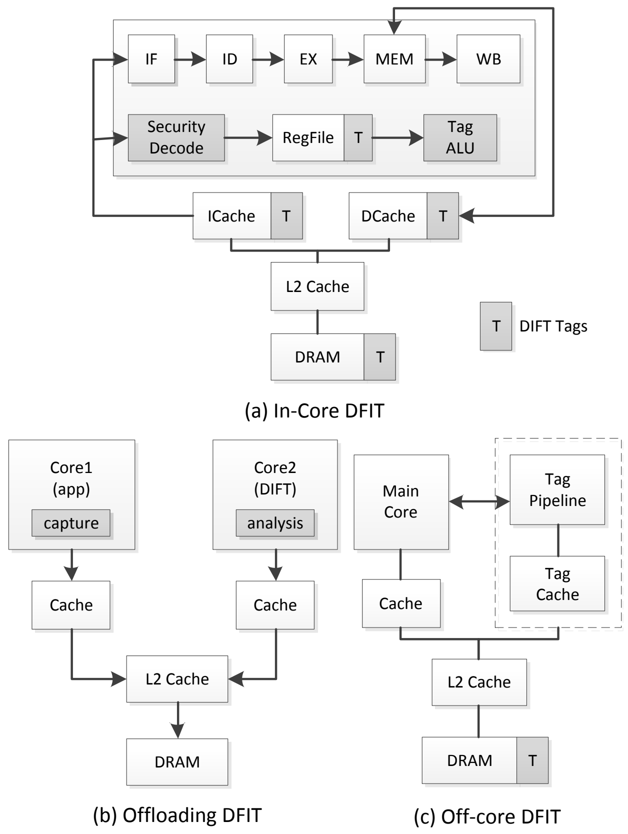

Figure 1: The three types of DIFT structures [5], [13]

[17]. The architecture supports software-programmable policy preventing a series of memory corruption attacks. Authors in [18] present a flexible metadata processing unit which can be used as a DIFT framework to enforce information flow tracking.

In [13], three types of DIFT architectures are investigated – in-core design, offloading design as well as off-chip design, as shown in Figure 1. Although the above DIFT approaches protect the system, they all belong to the in-core design which needs the modification of the whole processor architecture. For in-core design (Figure 1(a)), the whole design of DIFT framework is integrated with the structure of main processor core. The corresponding ISA will also be modified to support special operations, e.g., tag propagation rule configuration. They all bring in the high cost of the deployment.

An alternative approach is to use another core to track the information flow in a multi-core chip [19], which is a kind of a offloading design. However, this method introduces extra inter-communication overhead between cores. For offloading design (Figure 1(b)), the workload of tracking data flow will be transported to another processor core. Therefore, this scheme can only be implemented in a Multi-Processor or Multi-Core system. Moreover, it causes high performance overhead in communication between different processor cores. As a result, the offloading scheme cannot be directly applied in resource restricted devices, e.g., low-end embedded devices.

In addition, [5] proposes a co-processor design to implement DIFT framework. All information flow tracking logic is implemented as an extension of the main processor core, which is a kind of off-chip design. The off-chip design (Figure 1(c)) realizes the whole DIFT framework as a hardware module or hardware IP. Meanwhile, this hardware module can be attached to system bus or specified interface to receive essential information from main processor core. The off-chip design is easy to be implemented and deployed.

D. Runtime Protection

Formal methods have shown their importance in exhaustive hardware security verification [20]–[23], but few of them were designed for securing post-fabrication designs. Remote attestation (RA) model provides a method for a remote host, defined as a Verifier, to authenticate the configurations and states of software and hardware on the local host, defined as Prover. Specifically, the RA provides a runtime detection and protection. A method is proposed in [24] to detect the violation of control flow integrity on local host. Software attacks leading to the deviation of the control-flow, such as code injection and reuse, will be disclosed by the Verifier. [8] protects the execution of program at runtime by checking whether the output is the same to the expected values.

On the hardware side, verifiable ASICs is proposed to verify the correctness of hardware system functionality [25]. In the work, a runtime verification is performed by realizing an interactive encryption protocol between untrusted ICs, the Prover, and a second trusted ICs, the Verifier. It was the first attempt to compute proofs of correct execution through utilizing verifiable computations. However, for security purpose, their correctness checking method would result in high computational cost and overhead. Furthermore, their method was designed for checking specific property rather than the entire set of functional properties. Another solution for hardware runtime formal verification of security properties is presented in [26]. The proposed runtime PCH framework integrates the symbolic execution and the SAT solving. An FPGA based SAT solver is developed to verify the security properties for providing a high-level protection of the hardware system.

Meanwhile, many runtime hardware approaches were developed for information flow security, which could guarantee that all information flows satisfy the given security policies. For instance, GLIFT was proposed in [27] and could dynamically detect malicious logic through tracking the information flow in the hardware at runtime. The security hardware description languages such as Caisson [28], Sapper [29] and SecVerilog [30] enforce security policies by adding logic of information flow control in the hardware. However, these information flow control based techniques can only provide protections against information leakage. In this paper, we apply the DIFT in the RA model to protect both software and hardware from a variety of attacks.

III. Runtime Hardware and Software Co-Verification

In this section, we present the architecture of the proposed runtime solution and then describe the security check rule of the DIFT implemented in the architecture. The proposed solution follows the “Verifier-Prover” architecture. Specifically, an off-chip DIFT co-processor is inserted as the Verifier while the main RISC-V processor is the Prover delivering instructions to the Verifier via a dedicated interface.

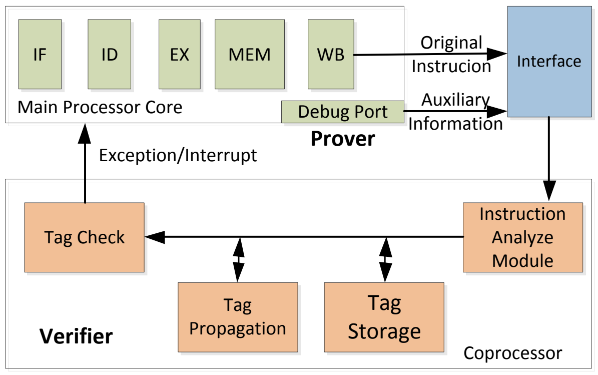

Figure 2: The co-verification framework

A. Off-Chip DIFT Co-Processor

We adopt an off-chip design to provide the runtime verification for the computer system. The entire framework is demonstrated in Figure 2. In the Prover’s side, the main processor core executes the instructions from the software level. Meanwhile, it provides all executed runtime instructions’ information to the Verifier for analyzing whether the current instruction violates the security checking rule. The communication channel between Prover and Verifier can be a system bus or a specified interface, e.g., debug port. Via the channel, the related information, like instructions or processor states, is sent to Verifier.

On the other hand, the Verifier analyzes the information, such as the instruction fetched from the main processor core, and then extracts the essential information. Taking RISC-V ISA as an example, the essential information includes register, opcode and instruction function code. According to these essential information, the register index, memory address can be easily acquired. In addition to instructions, there are other information obtained through the channel, such as the program counter. Based on all the above information from the Prover, the DIFT mechanism in the Verifier propagates the tag and computes it according the specified tag propagation rule. Finally, the tag is checked according to the checking rule and the exception is raised once the checking fails. Although there is latency during the verification, the performance is good enough for the runtime defense of the attack. We will demonstrate it in the experiment part.

B. Tag Propagation and Checks

Based on the above architecture, we implement the DIFT to detect the software-based attack. The DIFT framework includes three parts: i) tag source; ii) tag propagation; and iii) tag checking. The tag is used to store the related attributes of instructions, e.g., the privilege level of instruction and the source location of data (either on-chip or off-chip). The operation of tag system includes tag analysis, tag propagation and tag checking. Along with the execution of instructions in the main processor, the tags' states are updated in the co-processor. As shown in Figure 2, the tag system is implemented in the Verifier. Compared with the in-core design which involves the tag system in the Prover, the proposed off-core DIFT structure does not cause the performance degradation and hardware overhead of the main processor.

Tag Source. In the Verifier side, two Look-Up Tables (LUTs) are maintained to record the register and memory information of the main processor. One is mapped from the registers, and the other is mapped from the memory. An item in the LUT stands for a piece of unique register or memory. For each item in the LUT, there are several bits used as the types of tag. Every type of tag includes two states: tagged or untagged. Each individual bit in the item is utilized to record the state of the specific tag type. Both LUTs are stored in the Tag Storage module. Along with the input of the instruction, the tag states are initialized. Then the tag is read by instruction analyze module and sent to tag propagation module. After each one-step tag propagation, the destination's tag state is determined and then stored in the LUTs. All these operations are finished by hardware, thus no instruction can directly access the storage area.

Tag Propagation. Although the tag is propagated among items in the LUTs, the computation for getting destination's tags is performed according to the propagation rule in the tag propagation module. There are four basic propagation rules shown in Table I. Tags involved in the propagation are performed bit operation depending on the right column of the table. To reduce the cost and overhead, only a specific set of rules, selected by the user, can be applied to compute the tag propagation in the proposed framework. Different propagation rule influences the detection accuracy of the DIFT framework. For instance, the tag propagation using AND-Rule is more convergent than using OR-Rule. The tag propagation module firstly receives the analyzed instruction with the corresponding tags from instruction analyzed module. Then, according to the specified propagation rule, the destination tags' states are inferred based on the source tags' states.

Tag Checking. The security rule checking is performed in the tag check module. The checking rules are a series of user defined rules to check whether the tags' states changes are valid. An interrupt is raised once the violation is detected. For instance, in the return oriented programming (ROP), the return address is modified by attackers. The security policy is that the return address of function cannot be replaced with a malicious one. The check rule is to check whether the tag standing for the program counter is tainted by the tag state value standing for the malicious address. We show the detailed demonstration in the case study section.

User Customization. User can specify different security policies by combining different check rules and propagation rules. This user operation can be completed by configuring the

Propagation Rule	Operation
OR-Rule	Des.tag=Op1.tag $\bigvee Op2.tag$
AND-Rule	Des.tag=Op1.tag $\bigwedge Op2.tag$
XOR-Rule	Des.tag=Op1.tag $\bigoplus Op2.tag$
COPY-Rule	Des.tag=Op.tag

Table I: Common propagation rules in DIFT

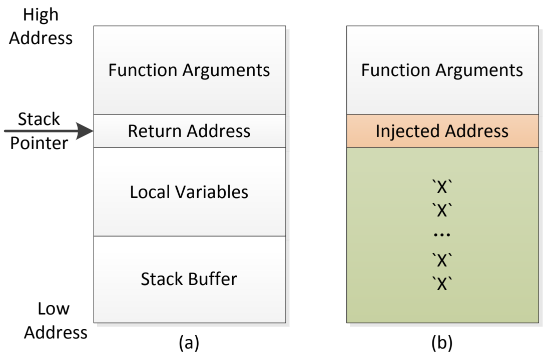

Figure 3: The structure of the software stack

Control Status Register (CSR). The configuration is enforced at boot time and maintained in company with the lifetime of the system. The user customization function can only be used to configure rules rather than the property of tags, e.g., width.

IV. Case Study

In this section, we validate the effectiveness of the proposed DIFT architecture by detecting a buffer overflow attack.

A. Experimental Setup and Buffer Overflow Attacks

In the proposed DIFT RA architecture, the Verifier is implemented as an extended co-processor of Rocket-Chip [12] processor core. We synthesized the Rocket-Chip with the extension on a Xilinx Artix-7 35T FPGA board.

Listing 1 Buffer Overflow Attacks Example

```
void valueCopy(unsigned long *dst,
    unsigned long *src, int length){
    int i;
    for(i = 0; i < length; i++)
      dst[i] = src[i];
}
void vulnerableFunction(unsigned long *src, int length) {
    unsigned long buf[20];
    valueCopy(buf, src, length);
}
void main() {
  unsigned long src[60];
  for(int i = 0; i < 60; i++)
    src[i] = i;
  vulnerableFunction(src, 60);}
```

The basic structure of software attacks is shown in Figure 3. The stack buffer is an area for temporary data storage. The attackers can use the data in buffer to overwrite the return address. Then, the control flow of program will be redirected to malicious programs. We take the following code as an example in List 1 to show the steps of buffer overflow attacks. The function valueCopy will modify the return address of

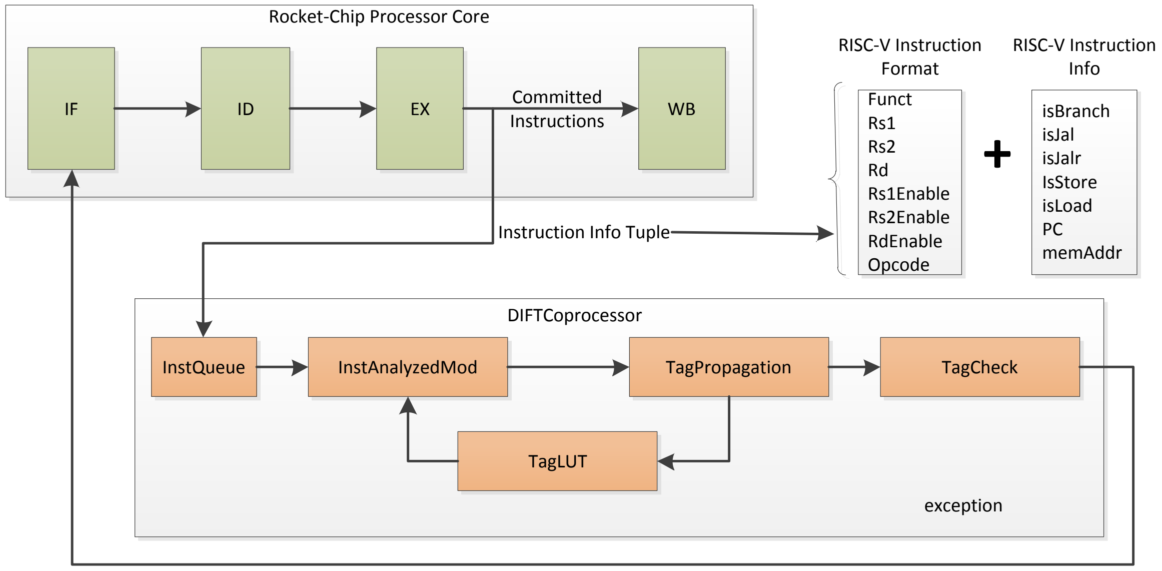

Figure 4: The proposed security structure implemented in the hardware

function vulnerableFunction by overwriting the buffer in it. As a result, the function vulnerableFunction will return to an attacker-controlled program.

B. FPGA Prototype

We implement the Verifier as a co-processor in the Rocket-Chip to verify the instruction executing in the main processor core at runtime. The overview of the prototype is shown in Figure 4. The Rocket-Chip processor core passes the committed instruction and related information to the co-processor. The co-processor uses 1 bit tag to track instruction and verify whether the return address is overwritten by the malicious data. The tags are stored in the TagLUT module and all the LUTs are one-to-one mapped from the registers and memories in the Rocket-Chip core.

From the Rocket-Chip core to the co-processor, the instruction information tuple includes the RISC-V instruction information i.e., function code, register, instruction type, program counter, and memory address. A four-stage pipeline is deployed in the co-processor. The instruction information tuple is delivered and stored into the instruction queue, which is InstQueue module in Figure 4. Then the tuple and the LUTs with the tags, from the TagLUT module, are analyzed in InstAnalyzedMod.

After that, the analyzed outputs from the InstanalyzedMod are utilized to compute the tags' states in the destinations and update the entire tag system in the TagLUT. In this experiment, the propagation rule is set as the OR-Rule enforcing a strict protection on the return address. Data from the outside of the legal address interval is tagged as "1". Once the return address is overwritten by the illegal data, the tag state of the program counter will be updated to "1" accordingly. Therefore, the program counter's tag is monitored during the execution. The exception will be raised if the program counter's tag state is updated as "1".

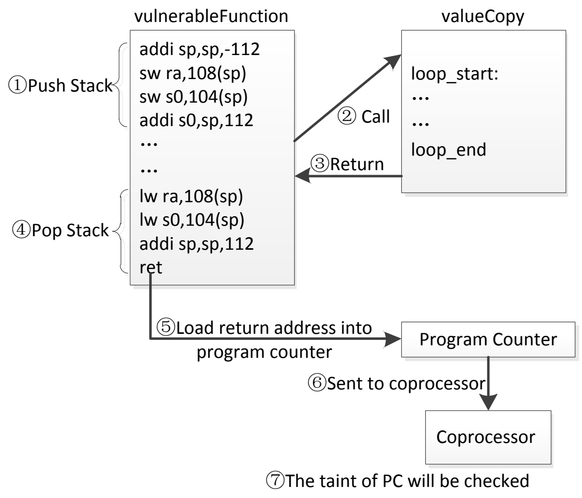

Figure 5: The process of buffer-overflow detection

C. Results and Analysis

From the viewpoint of the tag system, the detailed process of the above buffer-overflow demonstration is illustrated in Figure 5. First, the return address of the vulnerableFunction is stored into the software stack. Then, the function valueCopy is called and the return address is overwritten, accordingly. The function valueCopy then returns to vulnerableFunction. In the fourth step, the return address in software stack is popped and the return address is sent to the program counter. After the return instruction is executed, the related information is sent to the co-processor for further processing. The corresponding tags in the co-processor are tainted by the tag state "1" following the above procedure. Finally, the program counter's tag state is also updated and checked. As a result, the exception notifies the whole processor core to halt the whole system.

Component	BRAM	LUT
Base Rocket-Chip System	35	14843
Co-processor Design	4	290
Co-processor Overhead	11.4%	1.9%

Table II: Complexity of the prototype FGPA implementation of the co-processor design

As the the pipeline in the Rocket-Chip processor core is not modified, there is no latency on instruction execution. The hardware overhead of the prototype FPGA implementation is shown in Table II. The hardware overhead introduced by our framework is the LUT logic and the block RAMs (BRAMs). The hardware overhead of the BRAM and LUTs are 11.4% and 1.9% , respectively.

V. Conclusion and Future Work

In this paper, we present a hardware and software runtime co-verification to protect the threats of software-based attacks from the RISC-V architecture. The DIFT security mechanism is implemented in a RA structure and integrated as a co-processor. Our proposed framework does not modify the architecture of the main processor and address the attack by few hardware overhead. In future, the protection will be extended from the processor to the entire SoC system. The secure communication between peripheral device and third-party IP will be considered. More sophisticated DIFT security policies will be delivered.

Acknowledgment

The work was partially supported by Semiconductor Research Corporation (2018-TS-2860) and National Science Foundation (CNS-1801599).

References

[1] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher *et al.*, "Spectre attacks: Exploiting speculative execution," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 1–19.

[2] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown," *arXiv preprint arXiv:1801.01207*, 2018.

[3] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas, "Secure program execution via dynamic information flow tracking," in *ACM Sigplan Notices*, vol. 39, no. 11. ACM, 2004, pp. 85–96.

[4] L. C. Lam and T.-c. Chiueh, "A general dynamic information flow tracking framework for security applications," in *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*. IEEE, 2006, pp. 463–472.

[5] H. Kannan, M. Dalton, and C. Kozyrakis, "Decoupling dynamic information flow tracking with a dedicated coprocessor," in *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*. IEEE, 2009, pp. 105–114.

[6] R. Whelan, T. Leek, and D. Kaeli, "Architecture-independent dynamic information flow tracking," in *International Conference on Compiler Construction*. Springer, 2013, pp. 144–163.

[7] M. S. Lam, M. Martin, B. Livshits, and J. Whaley, "Securing web applications with static and dynamic information flow tracking," in *Proceedings of the 2008 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*. ACM, 2008, pp. 3–12.

[8] L. Gu, X. Ding, R. H. Deng, B. Xie, and H. Mei, "Remote attestation on program execution," in *Proceedings of the 3rd ACM workshop on Scalable trusted computing*. ACM, 2008, pp. 11–20.

[9] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik, "A minimalist approach to remote attestation," in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2014, pp. 1–6.

[10] C. Kil, E. C. Sezer, A. M. Azab, P. Ning, and X. Zhang, "Remote attestation to dynamic system properties: Towards providing complete system integrity evidence," in *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*. IEEE, 2009, pp. 115–124.

[11] RISC-V: The Free and Open RISC Instruction Set Architecture. https://riscv.org/.

[12] Rocket-Chip SoC Generator. https://github.com/sifive/freedom.

[13] A. Jahanshahi, "A brief review on some architectures providing support for dift," *arXiv preprint arXiv:1911.05664*, 2019.

[14] S. Weiser, M. Werner, F. Brasser, M. Malenko, S. Mangard, and A.-R. Sadeghi, "Timber-v: Tag-isolated memory bringing fine-grained enclaves to risc-v." in *NDSS*, 2019.

[15] M. Dalton, H. Kannan, and C. Kozyrakis, "Raksha: a flexible information flow architecture for software security," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2, pp. 482–493, 2007.

[16] C. Palmiero, G. Di Guglielmo, L. Lavagno, and L. P. Carloni, "Design and implementation of a dynamic information flow tracking architecture to secure a risc-v core for iot applications," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 2018, pp. 1–7.

[17] Open-Source RISC-V Processor Core, PULPino. https://www.pulp-platform.org/.

[18] U. Dhawan, N. Vasilakis, R. Rubin, S. Chiricescu, J. M. Smith, T. F. Knight Jr, B. C. Pierce, and A. DeHon, "Pump: a programmable unit for metadata processing," in *Proceedings of the Third Workshop on Hardware and Architectural Support for Security and Privacy*. ACM, 2014, p. 8.

[19] V. Nagarajan, H.-S. Kim, Y. Wu, and R. Gupta, "Dynamic information flow tracking on multicores," in *Proceedings of the Workshop on Interaction Between Compilers and Computer Architectures*, 2008.

[20] X. Zhang and M. Tehranipoor, "Case study: Detecting hardware trojans in third-party digital ip cores," in *HOST*, 2011, pp. 67–70.

[21] E. Love, Y. Jin, and Y. Makris, "Proof-carrying hardware intellectual property: A pathway to trusted module acquisition," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 1, pp. 25–40, 2012.

[22] Y. Jin, B. Yang, and Y. Makris, "Cycle-accurate information assurance by proof-carrying based signal sensitivity tracing," in *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013, pp. 99–106.

[23] Y. Jin, "Design-for-security vs. design-for-testability: A case study on dft chain in cryptographic circuits," in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2014.

[24] T. Abera, N. Asokan, L. Davi, J.-E. Ekberg, T. Nyman, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "C-flat: control-flow attestation for embedded systems software," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 743–754.

[25] R. S. Wahby, M. Howald, S. Garg, A. Shelat, and M. Walfish, "Verifiable asics," in *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 2016, pp. 759–778.

[26] X. Guo, R. G. Dutta, J. He, and Y. Jin, "Pch framework for ip runtime security verification," *Asian Hardware Oriented Security and Trust (AsianHOST)*, pp. 79–84, 2017.

[27] M. Tiwari, H. M. Wassel, B. Mazloom, S. Mysore, F. T. Chong, and T. Sherwood, "Complete information flow tracking from the gates up," in *ACM Sigplan Notices*, vol. 44, no. 3. ACM, 2009, pp. 109–120.

[28] X. Li, M. Tiwari, J. K. Oberg, V. Kashyap, F. T. Chong, T. Sherwood, and B. Hardekopf, "Caisson: a hardware description language for secure information flow," in *ACM SIGPLAN Notices*, vol. 46, no. 6. ACM, 2011, pp. 109–120.

[29] X. Li, V. Kashyap, J. K. Oberg, M. Tiwari, V. R. Rajarathinam, R. Kastner, T. Sherwood, B. Hardekopf, and F. T. Chong, "Sapper: A language for hardware-level security policy enforcement," in *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1. ACM, 2014, pp. 97–112.

[30] D. Zhang, Y. Wang, G. E. Suh, and A. C. Myers, "A hardware design language for timing-sensitive information-flow security," *ACM SIGPLAN Notices*, vol. 50, no. 4, pp. 503–516, 2015.