Hardware Secure Execution and Simulation Model Correlation using IFT on RISC-V

Geraldine Shirley Nicholas†
Electrical and Computer Engineering
University of North Carolina at Charlotte, Charlotte, USA
gnichola@uncc.edu

Bhavin Thakar
Electrical and Computer Engineering
University of North Carolina at Charlotte, Charlotte, USA
bthakar1@uncc.edu

Fareena Saqib
Electrical and Computer Engineering
University of North Carolina at Charlotte, Charlotte, USA
fsaqib@uncc.edu

ABSTRACT

With Heterogeneous architectures and IoT devices connecting to billions of devices in the network, securing the application and tracking the data flow from different untrusted communication channels during run time and protecting the return address is an essential aspect of system integrity. In this work, we propose a correlated hardware and software-based information flow tracking mechanism to track the data using tagged logic. This scheme leverages the open-source benefits of RISC-V by extending the architecture with security policies providing precise coarse-grain management along with a simulation model with minimal overhead.

KEYWORDS

Information Flow Tracking (IFT), Security, ISA, RISC-V, Toolchain, Software attacks

CCS Concepts

• Security and privacy ~ Security in hardware ~ Hardware security implementation ~ Hardware-based security protocols

ACM Reference format:

Geraldine Shirley Nicholas, Bhavin Thakar and Fareena Saqib. 2021. Hardware Secure Execution and Simulation Model Correlation using IFT on RISC-V, In *Proceedings of 2021 Great Lakes Symposium on VLSI (GLSVLSI '21) June 22–25, 2021, Virtual Event, USA.* ACM, New York, NY, USA. 6 pages. https://doi.org/10.1145/3453688.3461517

1 Introduction

Any hardware design is vulnerable to untrusted entities and malicious data which often exploits the integrity of the system. Unauthorized access to a system and software-based attacks such as code modification and memory corruption attacks through untrusted nodes often degrades the application-level integrity.

GLSVLSI '21, June 22–25, 2021, Virtual Event, USA.

ACM ISBN 978-1-4503-8393-6/21/06.
https://doi.org/10.1145/3453688.3461517

Heterogeneous System on Chips (SoCs) and Internet of Things (IoT) devices connected all over the world communicate with untrusted communication channels that are vulnerable to the system leading to information leakage and code injection. A robust and secure mechanism is needed to protect the data and to monitor the critical information flow in the devices.

Although many techniques were proposed to detect and prevent such software-based security attacks, Information Flow Tracking (IFT) technique has been a promising and effective analysis technique in security applications by determining information leakage and detecting the malicious untrusted data at run time. Based on static verification during the design phase or dynamical checking at the runtime, different IFT approaches have been implemented. The precision of the IFT logic along with the granularity of the building blocks determines the different levels of abstraction. Gate level flow tracking has precise tracking rules for a set of universal gates with shadow logics [1]. Though it provides a formal basis for a system's root of trust with a compositional approach it results in a computationally complex design for the shadow logic functions with a high number of inputs and does not scale with design size.

Register transfer level (RTL) tracking is based on propagation rules with RTL expressions [2]. Though verification is done at the early design stage with flow tracking libraries, RTL-based tracking adds the security labels explicitly which leads to precision degradation. Language-based IFT achieves a dynamic access-control mechanism with communication channels [3]. It focuses on security goals rather than protecting confidentiality by controlling the information flow. This paper is based on Instruction level (ISA) dynamic information flow tracking with coarse granularity labels and propagation policies to track the input data by using a tag bit which is a hardware-based approach by modifying the architecture and defining the security policies to track the spurious data. The most discernible problem with the proprietary architectures is the inflexibility with the access domain and complexity in modifying the architecture. This proposed approach leverages the open-source benefits of RISC-V which makes it flexible and extendable in adding security policies with coarse-grain management.

The hardware-based custom extensions done to the RISC-V ISA is replicated with the simulation model correlating the feasibility and functional verification with minimal overhead for RISC-V security. This approach exploits the open-source

software toolchain [4] components to build a minimal working model with custom extensions. In this paper, a hardware design with a software simulation model emphasizes the correlation using information flow tracking approach on the RISC-V platform.

Our contributions in this work are following:

1) Design of hardware-based Information Flow Tracking framework with tagged mechanism.
2) Design of simulation model of RISC-V with secure IFT extensions.
3) Toolchain extensions to support the RISC-V hardware security commands.
4) Verification of the simulation model to capture the stack attacks using return address.

2 Background

2.1 DIFT

Dynamic Information Flow Tracking includes both hardware and software-based implementations depending on the architecture of the system and focuses on the control as well as non-control-data attacks. Most of the implementations can be used to detect buffer overflow attacks, format string attacks along with various memory corruption attacks. The DIFT mechanism flow in order to monitor and track the untrusted data starts by allocating a tag to the malicious channels and marks it as spurious. During program execution, the tag propagation unit keeps track of the information flow generated by the spurious data. Finally, a tag checking unit detects the unsafe data by matching it with the security policies implemented for each untrusted channel and raises a security exception.

Many software-based approaches such as randomization [5] masking [6], access control [7] have been implemented but they lack isolation, and data shadowing requires additional overhead. Some hardware-based approaches include pointer taintedness [8], hardware-assisted data flow isolation [9], SIFT [10], SHIFT [11], virtualization [12] etc. All these approaches lack flexibility to certain extend and cause unavoidable overhead to the system. Fault-based attacks that gains control of the target execution flow by accessing the return address can be secured by IFT approaches which are specifically designed to protect the return address [13].

Simulation-based models provide a proof of concept in determining the feasibility of the design by verification and validation [14]. Thus, this approach is based on hardware design implementation along with simulation-based verification providing better flexibility and precision logic.

2.2 RISC-V

Security has been a major concern of RISC-V and developing security applications with the architecture's flexibility has led to various advancements and programs [15]. The RISC-V ISA security committee has proposed an abstraction augmented aISA that extends a bridge between hardware and software beyond the traditional ISA [16] for better control and optimization. The software toolchain for RISC-V supports GCC and Clang/LLVM compilers along with various benchmark environments [17]. Many simulators for RISC-V have been developed and the most common simulator which is used as a reference model for RISC-V ISA is the Spike simulator [18]. Combining the hardware and software implementations provides more coarse-grained data flow tracking with flexibility and minimal overhead. Thus, this paper implements a novel hardware and software simulation-based IFT by leveraging the RISC-V platform and providing a secure extension to the processor ISA. The ISA is modified, and the toolchain is updated for run time security.

3 Design of Hardware and Simulation Model for RISC-V Secure Extensions

The proposed scheme consists of the Hardware IFT mechanism for data tracking and software model with toolchain support of verification along with a simulation model to track the return addresses to prevent memory corruption and software attacks. To the authors knowledge this is a novel effort to integrate the toolchain support for the RISC-V security extensions.

3.1 IFT in RISC-V

The IFT technique focuses on preventing memory corruption and protects the return address from software attacks. The control data flow integrity with tagged bits ensures that the return address matches the corresponding address after the context switching which prevents an adversary from hijacking the return addresses. The tag-based analysis is flexible in tracking the record of the data with minimal overhead if a single bit is used as a tag. Compiler-based modification is done to add the security policies and to assist with the additional new instructions added to the architecture.

Stack and data protection by tracking and detecting the tagged data eliminates software attacks which focuses on the return address. In the tagged mechanism, labels are associated with the address of the data that are received from the untrusted source. With minimal hardware overhead, the tag mechanism is implemented by using a 1-bit tag to the data address. When a program is executed during run time the tag mechanism assigns the tag bits to the spurious data through the tag propagation module. The tag propagation module assigns tags by using the new custom instructions implemented in the ISA architecture to store and check the tag bits. The tag bits are stored in the tag cache which reduced the dedicated memory assigned for storing the tag bits.

The RISC-V Rocket core is modified to incorporate the security features both in the ISA level and on the toolchain to detect and eliminate the buffer overflow and string format attacks. At the ISA level, the tag module consists of all the tag management units and the translated compiler modifications are developed in RISC-V GNU. Design verification and validation is performed on

the RISC-V Spike Simulator. This model focusses on the stack protection and the return address stored on the stack. The security policy functions are used for checking the tag bits from the return address and the untrusted data source. An exception is raised when the framework detects a mismatch in the tag bit of the return and data address indicating a software attack has occurred.

3.1.1 Proposed ISA Level Extension in the Hardware model

The RISC-V core has a fixed base integer ISA with two primary variants: RV32I and RV64I, respectively [19]. The base integer has fixed-length 32-bit instructions with variable-length encoding and customizable accelerator extensions. This leverages the ISA to extend with more optional instruction-set extensions. The Load/Store architecture is dedicated to copying the data to the memory. Loads are encoded in the I-type format and stores are S-type format. Thus, the RISC-V core is modified by adding new Load/Store instructions which are used to provide security checks for the 1-bit tag in matching the return addresses. Based on the load and store encoding specified in the RISC-V core the new instructions are added:

- In the load instruction encoding: LDTCHECK
- In the store instruction encoding: SDTCHECK

Figure 1 shows the Load/Store Instruction format where the red highlighted fields (funct3 and opcode) are modified for the new instructions to specify the operations to be performed. When the data from an untrusted source is received the SDTCHECK will set the tag bit to 1 and LDTCHECK is used for loading the data word and checking if the tag bit is 0 or 1.

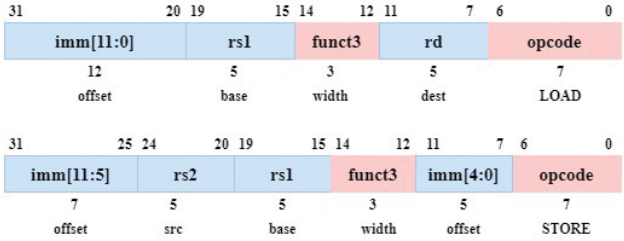

Figure 1: Load/Store Instruction format of RISC-V Architecture

The CSR address convention which provides accessibility for error checking by using the large CSR space is used to check the read/write compatibility of the custom instructions with the tag bit where the loaded tag bit and the expected tag bit in the instruction match, if not it raises an exception. A separate Tag cache module consists of all the tag management mechanisms with tag initialization, tag propagation, and tag checking modules. This module reduces the overhead of physically adding a one-bit tag to the memory. When a one-bit tag is added to the instruction this module fetches the tag and store it in the tag cache with its own tag cache mapping which reduces a significant amount of overhead in the architecture. The tag cache acts as a cache simulator for the tags and this is used to validate the return addresses and the memory access of the untrusted data. Figure 2 shows the E31 core RISC-V architecture with the added IFT Tag module.

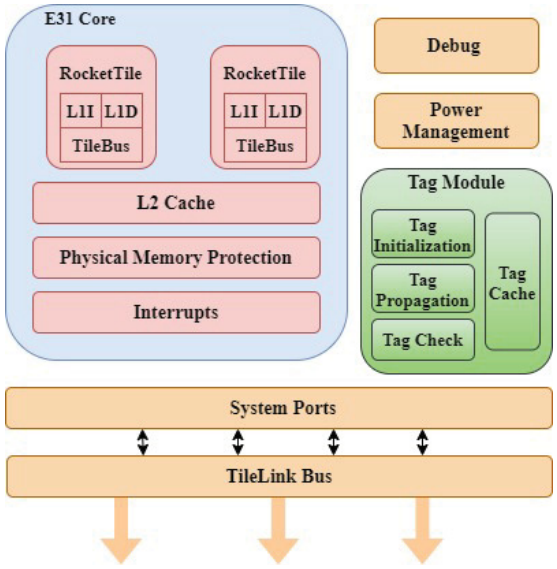

Figure 2: E31 Core RISC-V Architecture with Tag Module

The tag for each memory data is directly mapped and stored in the tag cache by virtual address spacing and lookup tables are used for mapping. During program execution, the tag initialization module assigns the tag bits to the data for the new instructions added and the tag propagation module tracks the flow of the data and the return addresses of each data from the new instructions. Finally, the tag check module is used to compare the tag bits of the new instructions and if there is a return address miss-match it raises a security exception. This detects the return address modification attacks when an adversary tries to modify the return address by function call and context switching.

When sensitive data is sent from an untrusted source the tag module initials the tag mechanism using the new instructions with load and store request similar to lowRISC [20]. The L1 data cache holds the data with the tag value for both read and write request and the tag module checks for a mismatch condition based on the tag bits.

3.1.2 Integration of the Features in Toolchain in the Simulation model

The RISC-V toolchain is modified for the newly added load/store instructions and the security policies for memory protection have been implemented. This proposed scheme focuses on tag security policy that addresses the load address, store address, and the return address. During program execution, security policies are applied to protect the data and the stack with

minimal overhead. The following section discusses the security policies to capture return address attacks.

3.1.3 Security Policies

Overwriting the return address by redirecting the execution using buffer overflow attacks and procedure calls with format string attacks corrupt the memory and lead to malicious code injections. In the RISC-V growing convention, the return address is x1(ra), stack pointer is x2(sp) and function argument/return value is x10-11. To protect the memory and the return address the data is tracked by tag bits.

i. Return Address attacks with procedure calls

An example of a buffer overflow attack, which modifies the return address is demonstrated in Figure 3 and the security policy to protect the return address is implemented using the Tag mechanism.

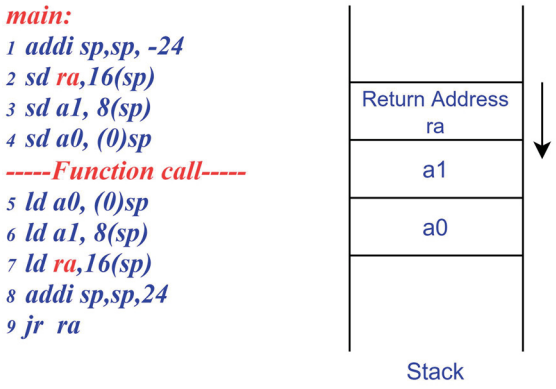

Figure 3: Return Address attacks with procedure calls

In the above code snippet, the stack is adjusted to make room for 3 items. The values in registers a1 and a0 are saved for future use and the return address(ra) is also pushed onto the stack and there is a procedure call that is carried out after which the registers and return address are restored. This simple implementation is done using the normal load/store operations where an adversary can modify and overwrite the return address during context switching which may not be noticed. One might think that the program is executing as intended but, in the background, this may lead to the execution of hidden functionalities and data leakage. In order to protect the return address from such attacks, the security policies implemented in this scheme use the new LDTCHECK and SDTCHECK instructions and assigns a tag bit to it. This helps in validating the return address after a function call. If the stored and loaded return address tag bit is not the same and a mismatch occurs it raises a memory exception.

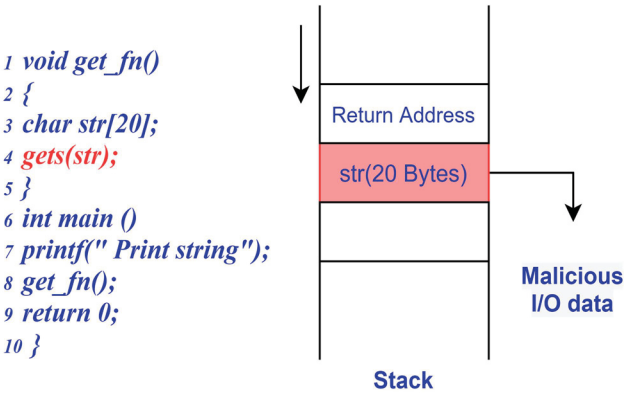

Figure 4: Buffer Return address attacks with I/O data

ii. Return address attacks with I/O data

Figure 4 demonstrates the return address affected by the input data from an untrusted communication channel and the security policy implemented to protect it.

From the above code snippet, the main function calls the *get_fn()* and the return address of the main is saved on the stack. The local variables are created for the *get_fn* and the specified space is allocated to the buffer. The gets(str) function which is highlighted gets an I/O string data that is received from a communication channel. If an adversary takes control of the I/O channels providing data that exceeds the limit of the buffer capacity, it results in a buffer overflow and overwrites the return address. To protect the return address from such attacks the security policy implemented marks the input string which is received by the *gets* function as malicious and assigns a tag bit to it. The return address of the main function is also tagged, and the malicious data propagation is tracked. When returning to the main function the security tag is matched with the return address. If a mismatch occurs, then a security exception is raised indicating a malicious I/O data attack.

4 Experimental Evaluation of the Simulation model

The IFT framework enables the software simulation model and introduces a novel toolchain modification to correlate the hardware design for verification security extensions in RISC-V. The architecture specific extensions are translated to assembler specific simulation model. The RISC-V toolchain is updated to support the IFT framework by adding two instructions to the gnu assembler and the spike simulator for software simulation. The modifications done on the ISA architecture is replicated in the Spike simulator verifying the minimal overhead and feasibility in both designs.

In this work we demonstrate the security capabilities on return address attacks where an adversary can control the targets

execution flow by corrupting the return address in the stack and executing an arbitrary code on the target. The security policies are described to check the flow of the tag bit in the tag cache and traversing through the newly added instructions. Figure 5 illustrates the security policy pseudocode for the check_tag and store_tag functions. The check_tag function executes the tag condition for the return address. If there is a mismatch it raises an exception without returning to the main address. The store_tag function checks for the untrusted channels and the return address before procedure calls and assigns a tag bit to it and stores the tag bit on the tag cache.

```
Function :check_tag
1 Offset decoding
2 Masking Address Decoding
3 Fetch tag value from tag cache for the masked address
4 condition( tag_bit = 0)
5       Function  executes
6 else
7       Raise an Exception
8 end


Function :store_tag
1 Offset decoding
2 Masking Address Decoding
3 Fetch tag value from tag cache for the masked address
4 condition check on the untrusted source
5       Assign tag bit =1
6 else
7       Assign tag bit =0

8 Store tag bit on tag cache
9 Store the address masked
```

Figure 5: Pseudocode for Check_tag and Store_tag functions

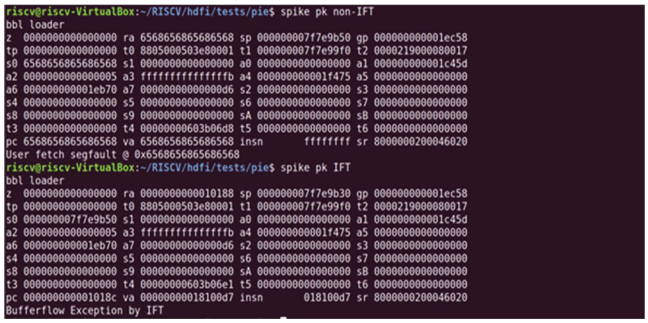

Figure 6: IFT Framework raising an exception for buffer overflow attack.

The IFT framework is tested on the Spike simulator by implementing a buffer overflow attack program. The new instructions are used with the stack operations where the return address is saved. Tag bits are assigned to the return address and the data to be stored on the stack. When the buffer overflow attack occurs the return address is checked for the tag value and if it's the same it executes the program. If there is a mismatch an exception is raised, and the program execution is stopped by eliminating buffer overflow attack. Figure 6 shows the window where, buffer overflow attack is executed, and the program returns to the main function when normal LOAD/STORE instructions are used. The newly added SDTCHECK is used to assign a tag bit for the return address and the LDTCHECK is used to the check the tag corresponding to the return address in the tag cache. For IFT the program raises an exception and stops without returning to the main function using the new instructions when there is a tag mismatch thus eliminating buffer overflow attacks. This framework protects the stack by using the customized instructions where the new return address compromised by an adversary is not loaded on the stack.

This feature can be further extended to support the main memory and detecting other software attacks by incorporating new security policies.

5 Security Analysis

The RISC-V model is vulnerable to security leaks such as buffer overflow, program counter attacks, fault attacks etc. The attacker leverages the bit flips or modification to the return address (RA) of the stack and leading to compromised devices. The hardware integration of the security features takes much longer for verification and security analysis.

The toolchain extension support helps in creating new software features by adding, modifying instructions and registers developed for our custom ISA inside the toolchain. It provides the flexibility to add new security policies and develop test codes and software programs to carry out security analysis for the custom ISA. The proposed simulation model supports the design and verification of security extensions to the RISC-V processor. The simulation model has extended toolchain that supports new functions and instructions such as IFT enabled execution and encryption support.

5 Conclusion

In this paper, we designed a hardware-based Information Flow Tracking framework with Tagged mechanism by assigning a 1-bit tag to the spurious data address and return address and translated the hardware architecture-specific extensions to compiler-specific simulation model. This is a novel contribution to integrate the hardware security to support the architectural integration for simulation model. The results of the implemented simulation model show that the framework tracks the tagged address and eliminates the buffer overflow attacks and the results are corelated to the hardware design. This implementation has minimal design overhead with better precision logic and higher performance in terms of verifying the security extensions.

ACKNOWLEDGMENTS

This research work has been sponsored by the National Science Foundation under grant No: 1814420, 1819687 and 1819694.

REFERENCES

[1] Mohit Tiwari, Hassan M.G. Wassel, Bita Mazloom, Shashidhar Mysore, Frederic T. Chong, and Timothy Sherwood, Complete information flow tracking from the gates up. *SIGARCH Computer. Architecture. News* 37, 1 (March 2009), 109–120. DOI: https://doi.org/10.1145/2528521.1508258

[2] A. Ardeshiricham, W. Hu, J. Marxen and R. Kastner, "Register transfer level information flow tracking for provably secure hardware design," *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, Lausanne, Switzerland, 2017, pp. 1691-1696, doi: 10.23919/DATE.2017.7927266.

[3] Sabelfeld, Andrei & Myers, Andrew. Language-Based Information-Flow Security. *IEEE Journal on Selected Areas in Communications*. (2013). 21. 10.1109/JSAC.2002.806121.

[4] RISC-V GNU Complier Toolchain [Online]. Available: https://github.com/riscv/riscv-gnu-toolchain

[5] V. Kuznetsov, L. Szekeres, M. Payer, G. Candea, R. Sekar, and D. Song, "Code-pointer Integrity," *in Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[6] J. Criswell, N. Dautenhahn, and V. Adve, "KCoFI: Complete control-flow integrity for commodity operating system kernels," in IEEE Symposium on Security and Privacy (Oakland), 2014.

[7] U. Erlingsson, M. Abadi, M. Vrable, M. Budiu, and G. C. Necula, "XFI: Software guards for system address spaces," *in Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[8] S. Chen, J. Xu, N. Nakka, Z. Kalbarczyk and R. K. Iyer, "Defeating memory corruption attacks via pointer taintedness detection," *2005 International Conference on Dependable Systems and Networks (DSN'05)*, Yokohama, Japan, 2005, pp. 378-387, doi: 10.1109/DSN.2005.36.

[9] C. Song *et al.*, "HDFI: Hardware-Assisted Data-Flow Isolation," *2016 IEEE Symposium on Security and Privacy (SP)*, San Jose, CA, USA, 2016, pp. 1-17, doi: 10.1109/SP.2016.9.

[10] Meltem Ozsoy, Dmitry Ponomarev, Nael Abu-Ghazaleh, and Tameesh Suri. 2011. SIFT: a low-overhead dynamic information flow tracking architecture for SMT processors. In Proceedings of the 8th ACM International Conference on Computing Frontiers (CF '11). Association for Computing Machinery, New York, NY, USA, Article 37, 1–11. DOI: https://doi.org/10.1145/2016604.2016650.

[11] H. Chen, X. Wu, L. Yuan, B. Zang, P. Yew and F. T. Chong, "From Speculation to Security: Practical and Efficient Information Flow Tracking Using Speculative Hardware," *2008 International Symposium on Computer Architecture*, Beijing, China, 2008, pp. 401-412, doi: 10.1109/ISCA.2008.18.

[12] A. Seshadri, M. Luk, N. Qu, and A. Perrig, "SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes," *in ACM Symposium on Operating Systems Principles (SOSP)*, 2007.

[13] N. Timmers, A. Spruyt and M. Witteman, "Controlling PC on ARM Using Fault Injection," *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, Santa Barbara, CA, USA, 2016, pp. 25-35, doi: 10.1109/FDTC.2016.18.

[14] Stephen McCamant and Michael D. Ernst. 2007. A simulation-based proof technique for dynamic information flow. In Proceedings of the 2007 workshop on Programming languages and analysis for security (PLAS '07). Association for Computing Machinery, New York, NY, USA, 41–46. DOI: https://doi.org/10.1145/1255329.1255336

[15] Arjun Menon, Subadra Murugan, Chester Rebeiro, Neel Gala, and Kamakoti Veezhinathan. 2017. Shakti-T: A RISC-V Processor with Light Weight Security Extensions. *In Proceedings of the Hardware and Architectural Support for Security and Privacy (HASP '17)*. Association for Computing Machinery, New York, NY, USA, Article 2, 1–8. DOI: https://doi.org/10.1145/3092627.3092629

[16] Q. Ge, Y. Yarom, and G. Heiser. No security without time protection: we need a new hardware-software contract. *In Asia-Pacific Workshop on Systems (APSys)*, 2018.

[17] M. Poorhosseini, W. Nebel and K. Grüttner, "A Compiler Comparison in the RISC-V Ecosystem," *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, Barcelona, Spain, 2020, pp. 1-6, doi: 10.1109/COINS49042.2020.9191411.

[18] RISC-V foundation. 2021. RISC-V Spike. https://github.com/riscv/riscv-isa-sim

[19] Andrew Waterman, Yunsup Lee, David Patterson, Krste Asanović, "The RISC-V Instruction Set Manual, Volume I: User-Level ISA Version 2.1", Technical Report UCB/EECS-2016-118, EECS Department, University of California, Berkeley, May 31, 2016.

[20] Alex Bradbury, Gavin Ferris, and Robert Mullins. Tagged memory and minion cores in the lowRISC SoC. Technical report, Computer Laboratory, University of Cambridge, Dec 2014.