A portable approach for SoC-based Dynamic Information Flow Tracking implementations

Muhammad Abdul Wahab[α], Pascal Cotret[α], Mounir Nasr Allah[β], Guillaume Hiet[β]
Vianney Lapôtre[γ], Guy Gogniat[γ]
[α] IETR / SCEE research group, firstname.lastname@centralesupelec.fr
[β] INRIA / CIDRE research group, firstname.lastname@centralesupelec.fr
[γ] Lab-STICC / University of South Brittany, firstname.lastname@univ-ubs.fr

Abstract

*This work introduces an efficient approach for DIFT (*Dynamic Information Flow Tracking*) implementations on reconfigurable chips. Existing solutions are either hardly portable or bring unsatisfactory time overheads. This work presents an innovative implementation for DIFT on reconfigurable SoCs such as Xilinx Zynq devices. Even though the feasibility of this approach is currently being studied, the first results are promising.*

1 Introduction

Security threats still remain a major concern in high technology systems. Regarding software security breaches, recent efforts such as DIFT have been proposed. DIFT aims to track the application control flow by adding metadata (also known as *tags*) to information containers (e.g. registers, memory addresses), propagating and checking it at runtime. These approaches have been successfully used against a wide range of attacks including buffer overflow, SQL injections and so on.

Nevertheless, existing approaches cannot be implemented in modern SoCs due to their rigidity and strong time overhead. The purpose of this work is to find a more efficient method to implement DIFT features in embedded systems without compromising their security level. The chosen approach, including a dedicated hardware DIFT coprocessor, is discussed in this paper.

Section 2 presents related works on SoC-based DIFT solutions. Then, Section 3 explains the overall architecture of the approach proposed in this work. Section 4 introduces the DIFT coprocessor; especially, the interface between the PS (*Processing System*) and the coprocessor. Finally, Section 5 sums up the contributions of this work and looks at perspectives.

2 Related work

First and foremost, DIFT was implemented in software as in [7] which presents a flexible solution. However, the performance overhead is too high (from 300% up to 3700% as noted in [6]). Several hardware architectures were proposed to speed up DIFT processing time: [2, 8, 9] provide lower performance penalties at the expense of flexibility.

In [6], Kannan et al. proposed to decouple tags computation from the main application instructions towards a dedicated hardware coprocessor allowing applications on the CPU to run faster with multiple concurrent active policies. From that time, other solutions were proposed to add features or improve performances shown in [6]. For instance, Deng et al. ([3, 4]) proposed to use dynamic tainting to implement DIFT and other similar techniques such as UMC (*Uninitialized Memory Check*) or BC (*Boundary Check*).

In [5], Heo et al. proposed system-level approach to implement DIFT and other related techniques. Information required for tags computation by the coprocessor are added to the application source code through binary instrumentation. This information is executed at runtime: as a result, it sends data from the CPU to a FIFO queue read by the coprocessor. This approach, even though more realistic and generic, presents some drawbacks:

1. Information leakage at the interface between the CPU and the coprocessor (transmission of `load/store` memory addresses).
2. Code injection attacks may not be detected because the injected code is not instrumented (no information flow control will be done on this code).
3. It requires binary instrumentation to export memory addresses to the coprocessor (added instructions will be architecture-dependent).

3 Global architecture

Previous works were implemented using softcores (the CPU is implemented on FPGA logic): as a consequence, information required for tags computation was easily extracted by accessing internal signals. However, it is impossible with hardcores (where the CPU is an ASIC). This work proposes to use existing debug components in ARM CPUs to partially recover information required to decouple regular computation from tags computation. The remaining information is obtained through static analysis. The architecture proposed in this work is shown in Figure 1.

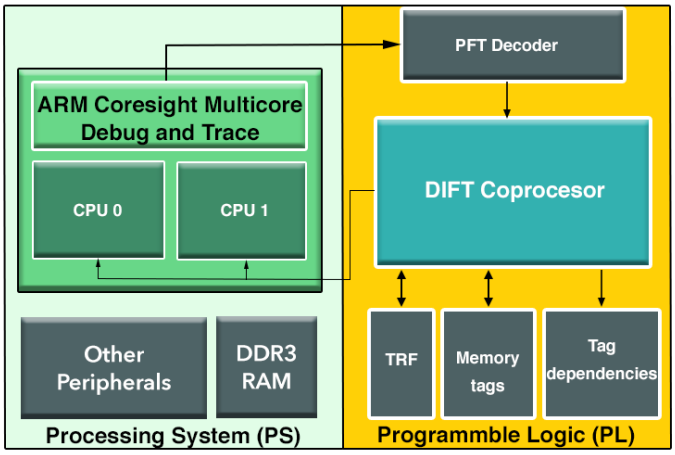

Figure 1. Proposed Architecture for DIFT

In this work, Zynq SoCs from Xilinx were used. However, it could be used with any architecture combining an ARM CPU with a FPGA. ARM debug components (called *CoreSight*) allow to generate and recover traces of applications ran by the CPU: traces contain information on instructions committed on the CPU. However, it uses a special protocol also known as PFT (*Program Flow Trace*): PFT outputs must be decoded to obtain human-

readable information on committed instructions.
The PFT decoder analyzes traces and sends them to the coprocessor implemented in the FPGA logic. The PFT decoder takes only 0.48% of FPGA logic. Moreover, during compilation phase of source code with LLVM, static analysis is done. PFT data is used alongside static analysis results that are loaded by the OS to Tag dependencies IP in the PL (*Programmable Logic*) when the program is launched. Tags are stored in TRF and in Memory tags (see Figure 1). The DIFT coprocessor checks tags according to user-defined security policies to verify if the CPU handles data in an unauthorized way.

4 DIFT Coprocessor

Decoupling DIFT operations from instruction decoding is possible by synchronizing both cores at system calls. The DIFT coprocessor requires at least three information from the CPU core obtained through CoreSight components and static analysis: PC (*program counter*), memory addresses (for *load/store* instructions) and instruction encoding.

4.1 DIFT Coprocessor interface

ARM CoreSight components allow to debug the code efficiently with negligible time overhead. Information obtained from CoreSight components of ARM Cortex-A9 (CPU included Zynq SoCs) are related to all the instructions modifying the PC register.

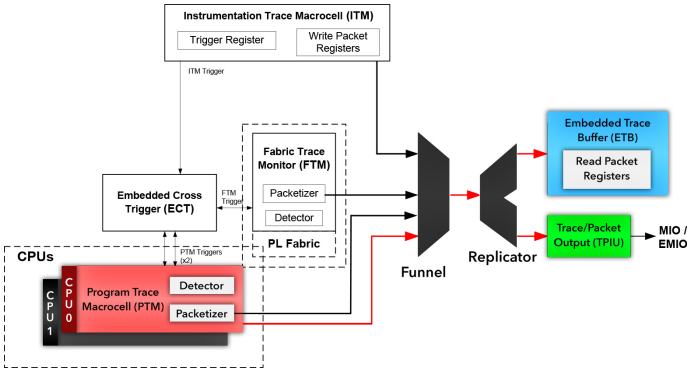

Figure 2. CoreSight Components[1]

Figure 2 shows the CoreSight components involved in the trace generation and export to the PL (FPGA area). PTM (*Program Trace Macrocell*) generates a trace for each committed instruction modifying the PC value. For instance, considering the code in Figure 3, PTM will generate a trace for instructions on lines 4 or 5 depending on the condition on line 3. The trace is transmitted through the funnel and the replicator and pushed in trace sinks (ETB and TPIU). ETB (*Embedded Trace Buffer*) allows to store traces in an on-chip RAM while TPIU (*Trace Port Interface Unit*) can send it to the programmable logic.

4.2 Static Analysis

Figure 3 shows the code and the result obtained through static analysis for this code. Static analysis allows to obtain tag dependencies shown in the control flow graph. The directives inside curly brackets indicate how tags should be propagated. Consider the node 2 in figure 3, the variable x should be tainted by the tag of "random" function output and y should be tainted by the tag of "input" function output.
The main core (ARMv7 architecture) commits instruction and waits, if necessary, on system calls until the DIFT coprocessor completes tags checking. Meanwhile the DIFT coprocessor reads tag dependencies, propagates and checks tags accordingly for all

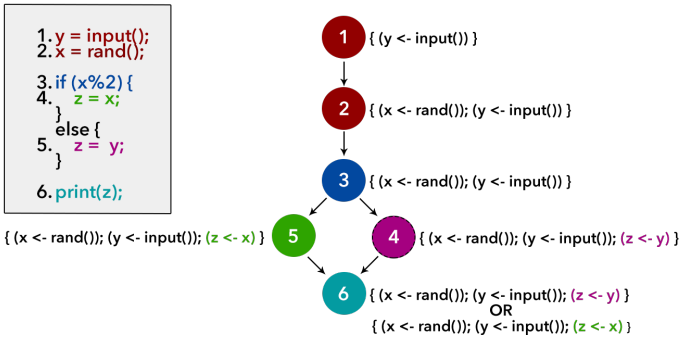

Figure 3. Example Code and CFG

the instructions. If a tag check fails, an exception is raised to alert the CPU.

5 Conclusion

The approach proposed in this work shows huge potential as it allows to efficiently implement DIFT on SoCs. This approach should not be limited to ARM-based SoCs: Intel also offers trace components, for debug purposes, allowing to retrieve information on committed instructions. Any SoC with hard cores including debug components can be used to implement DIFT with our approach. A prototype is under development to study feasibility of DIFT on Zynq SoC using our approach. Then, we plan to look at frequency issues between the CPU core and the DIFT coprocessor which is another reason why existing architectures are not commonly used.

References

[1] Zynq technical reference manual. `www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf`.

[2] M. Dalton, H. Kannan, and C. Kozyrakis. Raksha: A flexible information flow architecture for software security. *SIGARCH Comput. Archit. News*, 35(2):482–493, June 2007.

[3] D. Y. Deng, D. Lo, G. Malysa, S. Schneider, and G. E. Suh. Flexible and efficient instruction-grained run-time monitoring using on-chip reconfigurable fabric. In *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 137–148. IEEE Computer Society, 2010.

[4] D. Y. Deng and G. E. Suh. High-performance parallel accelerator for flexible and efficient run-time monitoring. In *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*, pages 1–12. IEEE, 2012.

[5] I. Heo, M. Kim, Y. Lee, C. Choi, J. Lee, B. B. Kang, and Y. Paek. Implementing an application-specific instruction-set processor for system-level dynamic program analysis engines. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 20(4):53, 2015.

[6] H. Kannan, M. Dalton, and C. Kozyrakis. Decoupling dynamic information flow tracking with a dedicated coprocessor. In *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, pages 105–114. IEEE, 2009.

[7] J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. 2005.

[8] G. E. Suh, J. W. Lee, D. Zhang, and S. Devadas. Secure program execution via dynamic information flow tracking. In *Acm Sigplan Notices*, volume 39, pages 85–96. ACM, 2004.

[9] G. Venkataramani, I. Doudalis, Y. Solihin, and M. Prvulovic. Flexitaint: A programmable accelerator for dynamic taint propagation. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 173–184. IEEE, 2008.