Towards a hardware-assisted information flow tracking ecosystem for ARM processors

Muhammad Abdul Wahab, Pascal Cotret, Mounir Nasr Allah, Guillaume Hiet, Vianney Lapotre, Guy Gogniat

Towards a hardware-assisted information flow tracking ecosystem for ARM processors

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 details a hardware-assisted approach for information flow tracking implemented on reconfigurable chips. Current solutions are either time-consuming or hardly portable (modifications of both sofware/hardware layers). This work takes benefits from debug components included in ARMv7 processors to retrieve details on instructions committed by the CPU. First results in terms of silicon area and time overheads are also given.**

I. Introduction

Nowadays, high-technology systems are highly threatened by security issues. In the context of software security, original solutions such as DIFT (*Dynamic Information Flow Tracking*) have been proposed since the 2000s. DIFT aims to ensure the application control flow by adding metadata (also known as *tags*) to information containers (e.g. registers, memory addresses). These tags are checked at runtime. DIFT already demonstrated a detection of a wide range of attacks such as SQL injections and buffer overflow.

However, existing solutions are not widely used in modern SoCs due to hardware and software dependencies. This work provides a clever DIFT implementation for recent SoCs without compromising their security level. This manuscript also describes the internal structure of a new hardware DIFT coprocessor and its implementation results.

Section II presents the most relevant related works. Then, Section III describes the main objectives of this work. Section IV presents the internal mechanisms and implementation results. Finally, Section V gives some conclusions and future perspectives.

II. Related works

First and foremost, DIFT implementations were primarily performed in software (without any hardware extensions) as done by Newsome et al. [1]. However, time overheads were too high (from 300% up to 3700%). In order to decrease processing times, several hardware extensions were proposed providing lower penalties at the expense of flexibility ([2], [3], [4]).

Kannan et al. [5] suggested to separate tags computation from the main application flow: a dedicated coprocessor handles tags, allowing the CPU to run faster. Furthermore, it allows to run simultaneously multiple DIFT checking rules. More recently, other solutions aimed to add features and improve performances shown in [5]. For instance, Deng et al. [6], [7] proposed a solution to implement DIFT and other similar runtime monitoring techniques such as UMC (*Uninitialized Memory Check*) or BC (*Boundary Check*).

Heo et al. [8] proposed a system-level approach to implement DIFT and other related techniques. Information required by the coprocessor for tags computation is added to the application source code through binary instrumentation. This information is executed at runtime: 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; 2) code injection attacks may not be detected as the injected code is not instrumented; 3) added instructions through binary instrumentation are architecture-dependent.

Table I is a qualitative comparison of some previous works. [5], [6] implemented DIFT using a softcore processor. In both cases, there are modifications of the CPU itself in order to export information. In this work, the main constraint is that the CPU is an ASIC: however, it will be easier to implement on several SoC based on the same architecture.

TABLE I
Brief comparison of previous works

Approaches	**Kannan [5]**	**Deng [6]**	**Heo [8]**
Hardcore portability	No	No	Yes
Time Overhead	+	++	+
Surface Overhead	+	-	-
Main CPU	Softcore	Softcore	Softcore

III. Objectives

Due to inflexibility and time overheads, DIFT is hardly adopted in modern SoCs. The main goal of this work is to provide a flexible approach for hardware-assisted DIFT based on a standard OS and a heterogeneous architecture such as Xilinx Zynq or Altera DE1-SoC. This work promotes DIFT by proposing a solution with several features:

- **Targeting unmodified processors**. Previous works used a softcore LEON3. Zynq devices contain an ARM processor which cannot be modified.

- **Scalability**. At first, this work focuses on single-core CPUs. An extension to multicore architectures is planned in the future.
- **Efficiency and flexibility**. It must be a low-area and fast solution: the processor must not wait for the coprocessor to complete DIFT tasks (at least, it may halt for the shortest possible time).
- **Secure tags computation**. It is assumed that tags and DIFT outputs must not be revealed to an unknown authority.

IV. Current status and preliminary results

The overall architecture used in this work is shown in Figure 1. Information required by the coprocessor for tags computation is partially recovered using existing debug components available in ARM processors (also known as Coresight components). Remaining information is obtained through software analysis.

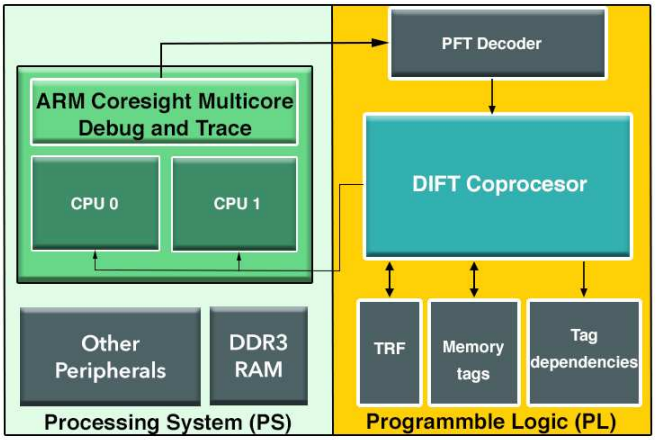

Fig. 1. Overview of the hardware-assisted DIFT architecture implemented on a Zynq device

A. Global approach

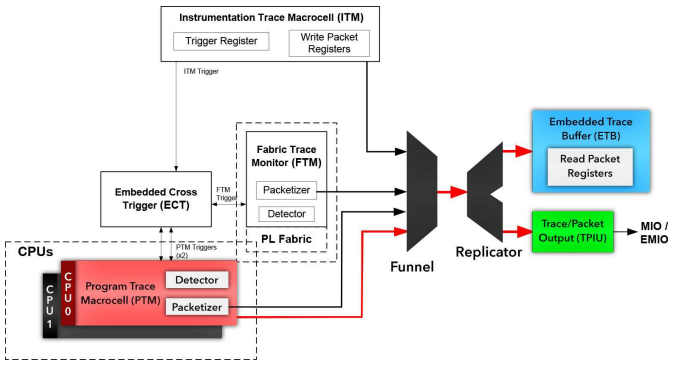

Fig. 2. Coresight Components on Zedboard [9]

Coresight components (Figure 2) can be used to debug (or trace) in an efficient manner multicore processors. A PTM (*Program Trace Macrocell*) is assigned to each CPU core: PTMs generate traces (e.g. partial inputs for DIFT computations). Traces only provide runtime information on instructions modifying the program counter (e.g. branches). Traces are transmitted through the funnel and replicator and then pushed in trace sinks (ETB and TPIU). ETB (*Embedded Trace Buffer*) is able to store traces in an 4KB on-chip RAM while TPIU (*Trace Port Interface Unit*) can send it to the programmable logic through the EMIO (Extended Multiplexed I/O) pins.

On the PL side, traces are decoded by the PFT decoder (*Program Flow Trace*, see Figure 1) and given in a format readable by the DIFT coprocessor. Tag dependencies block contains information obtained through software analysis and rules to handle tags. DIFT coprocessor reads traces given by the PFT decoder and finds which information containers must be propagated. Then, it looks for related tags in TRF (*Tag Register File*) or MR (*Memory tags*): TRF contains tags of each CPU register while MR contains tags for memory locations. The granularity of tags is a user-defined parameter. Finally, the DIFT coprocessor looks for security policy violations and eventually raises an exception.

B. Results

1) Traces generation: The approach described in this work is at least compatible with SoCs combining an ARM Cortex-A9 processor with a FPGA: Xilinx ZedBoard is the experiment platform in this work. All synthesis were done in Vivado 2014.4. Xilinx Standalone OS was first used to develop Coresight components drivers in order to understand the features offered by such modules and to verify trace contents.

Coresight drivers for standard Linux are currently being studied and compiled in a Yocto recipe. Traces have been successfully recovered in ETB; however, parasite traces are generated due to context switches.

2) Implementation results: For MiBench programs, the overhead introduced by Coresight components is negligible. As tracing components are in hardware and separated from CPU core, almost no overhead is observed. However, the worst case scenario is not evaluated yet and further testing with other benchmarks needs to be done before pronouncing on the efficiency of Coresight components.

Area results of TRF and PFT Decoder IPs are shown in Table II. Percentages are shown relatively to a Microblaze softcore with minimum area configuration (without caches nor BRAMs).

TABLE II
IP size for Zedboard (Zynq Z7020)

IP Name	Slice LUTs	Slice Registers	Slice
Microblaze	824	530	300
PFT Decoder	308 (37%)	222 (42%)	110(37%)
TRF	49 (6%)	64 (12%)	13 (4%)

V. Conclusion and future work

A first prototype is currently being developped to demonstrate the feasibility of the approach proposed in this work. Next steps are to build a full-featured system including a secure DIFT coprocessor. Then, DIFT on both Cortex-A9 cores will be implemented by duplicating DIFT coprocessor and other IPs. Dynamic partial reconfiguration will be studied to address energy consumption issues. The proposed approach is not specific to ARM hardcores and may well be adapted to Intel cores using Intel Processor Trace components.

References

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

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

[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, pp. 85–96, ACM, 2004.

[4] 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*, pp. 173–184, IEEE, 2008.

[5] 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*, pp. 105–114, IEEE, 2009.

[6] 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*, pp. 137–148, IEEE Computer Society, 2010.

[7] 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*, pp. 1–12, IEEE, 2012.

[8] 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)*, vol. 20, no. 4, p. 53, 2015.

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