Monitoring information flows in heterogeneous SoCs with a dedicated coprocessor

Muhammad Abdul Wahab [1], Mounir Nasr Allah [2], Pascal Cotret [4]
Guillaume Hiet [2], Vianney Lapôtre [3], Guy Gogniat [3]
[1] SCEE/IETR/CentraleSupélec - firstname.lastname@centralesupelec.fr
[2] CIDRE/INRIA/CentraleSupélec - firstname.lastname@centralesupelec.fr
[3] Lab-STICC - Université de Bretagne-Sud - firstname.lastname@univ-ubs.fr
[4] Independent researcher - pascal.cotret@gmail.com

Abstract

*Security is a major issue nowadays for the embedded systems community. Untrustworthy authorities may use a wide range of attacks in order to retrieve critical information. This paper introduces ARMHEx, a practical solution targeting DIFT (*Dynamic Information Flow Tracking*) on ARM-based SoCs (e.g. Xilinx Zynq). ARMHEx takes profit of ARM CoreSight debug components and static analysis to drastically reduce instrumentation time overhead (up to 90% compared to existing works).*

1 Introduction

During the last decade, several software security vulnerabilities have been discovered. Access control or cryptography can be used to limit access to confidential data or to enforce integrity. However, such techniques do not provide any guarantees once access is granted or data decrypted. Monitoring applications at runtime to check their behavior is a complementary solution. Among the different existing approaches, IFT (*Information Flow Tracking*) is an appealing solution that consists in tracking the dissemination of data inside the system.

This work is based on an hybrid approach combining SIFT (*Static Information Flow Tracking*) and DIFT (*Dynamic Information Flow Tracking*) [7]: both dynamic and hybrid IFTs will be cited as DIFT in this work. DIFT consists of performing three operations:

1. **Tag initialization**: it consists in attaching tags to information containers (e.g. file, variable, memory word, etc). Those tags correspond to the security level or the type of data they contain.
2. **Tag propagation**: tags need to be propagated from source operands to destination operands to track information flows resulting from the execution of each CPU instruction.
3. **Tag Check**: tags are checked with a security policy, at runtime and on a regular basis, to ensure that critical information is not handled by untrusted functions or entities.

2 Related works

In order to overcome high time overheads of software solutions for DIFT (at least 300%), hardware mechanisms were implemented. We can distinguish four main approaches:

1. **Filtering hardware accelerator**. Instead of computing tags for each CPU instruction (as done in other approaches), this approach proposes to filter monitored events (e.g. system calls) before computing tags to lower DIFT time overhead.
2. **In-core** ([2]). This approach relies on a deeply revised processor pipeline. Each stage of the pipeline is duplicated with a hardware module in order to propagate tags all along the program execution.
3. **Offloading**. In this case, DIFT operations are computed by a second general purpose processor. The required information for DIFT (i.e. PC register value, instruction encoding and `load/store` memory addresses) is sent by the processor running the application.
4. **Off-core** ([5, 4]). This approach seems similar to the offloading one. However, DIFT is performed on a dedicated unit instead of a general purpose processor. ARMHEx is based on this approach but differs in its implementation.

This work extends ideas presented in [1] and proposes a proof-of-concept prototype and its implementation on Zynq SoC (Zedboard) is detailed. It is shown that the area and power overhead of proposed implementation is better than existing approaches.

3 ARMHEx approach

A DIFT implementation is efficient when required information is obtained in the shortest possible time. ARMHEx coprocessor requires at least three pieces of information to compute tags propagation:

1. PC register value.
2. Instruction encoding.
3. `load/store` memory addresses.

PC register value and some memory addresses are partially recovered using CoreSight components. Missing information about memory addresses and instruction encoding is obtained through static analysis and instrumentation.

CoreSight components (Figure 1) are a set of IP blocks providing hardware-assisted software tracing. These components are used for debug and profiling purposes. For instance, they can be used to find software bugs and errors or even for CPU profiling (number of cache misses/hits and so on).

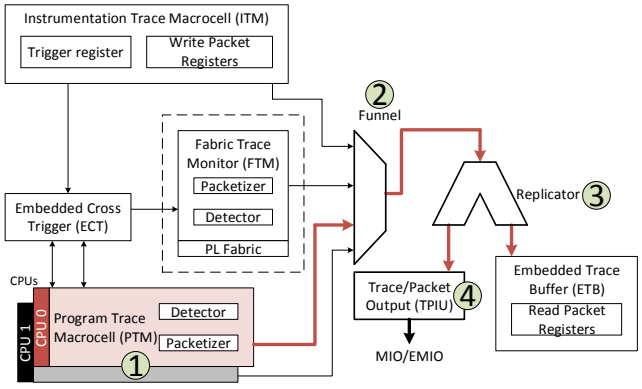

Figure 1. CoreSight components in Xilinx Zynq

ARMHEx uses these components to retrieve information on some instructions committed by the CPU at runtime. In Figure 2, the PFT decoder (1) is a state machine that decodes trace packets received from CoreSight components. As the trace is sent at 250 MHz by the TPIU, it needs to be decoded at the same frequency to avoid unnecessary storage overhead. The PTM sends different types of packets to analyze the code being executed on the CPU. Each type of packet has its own packet FSM (*Finite State Machine*) and a global state

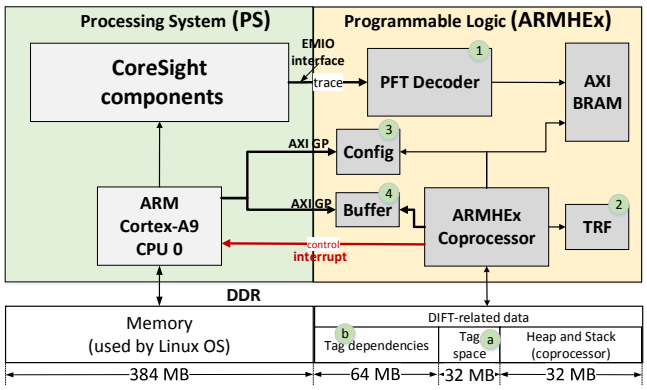

Figure 2. Internal architecture of an ARMHEx system

machine controls packet FSMs. Finally, decoded traces are stored in AXI Block RAM.

The TRF (*Tag Register File* (2)) is a register file that stores tags for each of 16 ARM CPU registers and 32 NEON registers. The Config IP (3) is an AXI slave IP containing a set of registers that provides a communication channel between the CPU and the ARMHEx coprocessor: it is used to configure tag propagation rules, send the initial value of SP and for debug purposes. Buffer (4) is a FIFO (AXI Slave write-only interface and a custom interface for read channel) that contains instrumented memory addresses.

4 Implementation results

Implementations were done with Vivado 2016.4 tools on a Xilinx Zedboard including a Z-7020 SoC (dual-core Cortex-A9 running at 667MHz and an Artix-7 FPGA). The ARMHEx coprocessor is implemented in a Microblaze softcore for this proof-of-concept.

4.1 Instrumentation overhead

The instrumentation time overhead is proportional to the number of instrumented instructions. The average time overhead for strategy #1 is 24.6% while it reaches 53.7% for related work instrumentation strategy. The average time overhead for strategy #2 is 5.37% which is better than the overhead of 60% reported by Heo et al. [4].

4.2 Area

Area results are shown in Table 1. Most of the FPGA area is filled by the AXI interconnect (5.87%), Config IP (5.20%) and the Microblaze softcore (4.62%). Other IPs occupy less than 1% of the FPGA area in terms of slices. In this work, ARMHEx targets a single Cortex-A9 core. Implementation results show that a Cortex-A9 dual-core, such as the one included in the Zynq Z-7020, could be easily protected. In the current configuration, the ARMHEx infrastructure could cover up to 5 Cortex-A9 cores simultaneously.

4.3 Comparison with previous works

Table 2 shows a performance comparison of ARMHEx with previous off-core approaches. Unlike previous works, ARMHEx is based on an ARM hardcore processor: it opens interesting perspectives as this work is easily portable to existing embedded systems. Approaches proposed by Heo [4] and Lee [6] are not portable on Zynq SoC due to CoreSight PTM component. Furthermore, the time cost for communication between a CPU and the coprocessor is 5.4% in this work compared to 60% in [4]. In terms of area, ARMHEx has the best coprocessor/processor ratio.

Table 1. Area results of ARMHEx on Xilinx Zynq Z-7020

IP Name	Slice LUTs	Slice Registers	Slice (in %)	BRAM Tile
Microblaze	1578	1407	614 (4.62)	6
MDM	102	110	40 (0.30)	0
Local memory	14	4	11 (0.08)	32
PFT Decoder	105	211	60 (0.45)	0
AXI TRF	53	105	24 (0.18)	1
Config	914	2141	692 (5.20)	0
AXI Interconnect	1788	2436	781 (5.87)	0
BRAM	2	0	1 (0.01)	2
BRAM Controller	157	168	59 (0.44)	0
Miscellaneous	641	586	171 (1.29)	0
Total Design	5354 (10.06%)	7168 (6.74%)	2453 (18.44%)	41 (29.29%)
Total Available	**53200**	**106400**	**13300**	**140**

Table 2. Performance comparison with related work

Approaches	Kannan [5]	Deng [3]	Heo [4]	ARMHEx
Hardcore portability	No	No	Yes	Yes
Main CPU	Softcore	Softcore	Softcore	Hardcore
Communication overhead	N/A	N/A	60%	5.4%
Area overhead	6.4%	14.8%	14.47%	0.47%
Area (Gate Counts)	N/A	N/A	256177	128496
Power overhead	N/A	6.3%	24%	16%
Max frequency	N/A	256 MHz	N/A	250 MHz

5 Conclusion and perspectives

This work is the first one to implement DIFT on ARM hardcore processors. Even though DIFT implementations on softcores exist, they are not all portable to hardcore CPUs. It is shown that by using our approach, only 6% of instructions need to be instrumented in an application compared to 60% instrumented instructions in related works. Implementation results show interesting perspectives for ARMHEx in terms of multicore runtime security. ARMHEx can be implemented in parallel as it has a moderate impact in terms of area (less than 20% of FPGA area is currently used).

References

[1] M. Abdul Wahab, P. Cotret, M. Nasr Allah, G. Hiet, V. Lapotre, and G. Gogniat. Towards a hardware-assisted information flow tracking ecosystem for ARM processors. In *FPL 2016*, Lausanne, Switzerland, Aug. 2016.

[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*, MICRO '43, 2010.

[4] 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 Trans. Des. Autom. Electron. Syst.*, 20(4):53:1–53:32, Sept. 2015.

[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*, June 2009.

[6] J. Lee, I. Heo, Y. Lee, and Y. Paek. Efficient security monitoring with the core debug interface in an embedded processor. *ACM Trans. Des. Autom. Electron. Syst.*, 22(1):8:1–8:29, May 2016.

[7] S. Moore and S. Chong. Static analysis for efficient hybrid information-flow control. In *2011 IEEE 24th Computer Security Foundations Symposium*, pages 146–160, June 2011.