Unveiling the Invisible Threads: Dynamic Information Flow Tracking and the Intriguing World of Fault Injection Attacks

JAIF 2023 - Gardanne

William PENSEC, Vianney LAPÔTRE, Guy GOGNIAT

Université Bretagne Sud
Lab-STICC

September 28, 2023

Context

- Rapid expansion of connected objects.
- Increased attack surface.
- Objects with physical proximity and network connectivity.
- Software and physical threats.

Issue

- How can we maintain maximum protection against software attacks in the presence of physical attacks?

Outline

Information Flow Tracking : what is it ?

IFT overview

- 2 categories: static or **dynamic**
- Different types: software, **hardware**, hybrids [1]
- Protection against software attacks (e.g.: *buffer overflow*, *format string*, *SQL injections*, ...) [2], [3]

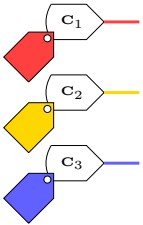
DIFT principle

- We attach **labels** called tags to **containers** and specify an information flow **policy**, i.e. relations between tags
- At runtime, we **propagate** tags to reflect information flows that occur and **detect** any **policy violation**

DIFT : how does it work ?

Three steps

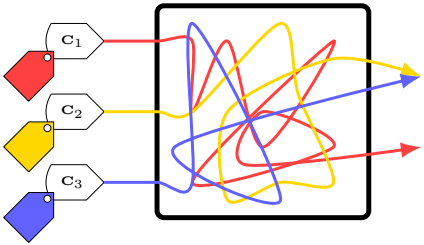
- Tag initialization



DIFT : how does it work ?

Three steps

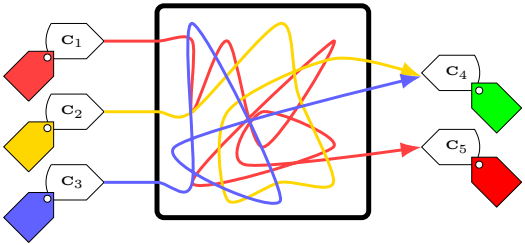
- Tag initialization
- Tag propagation



DIFT : how does it work ?

Three steps

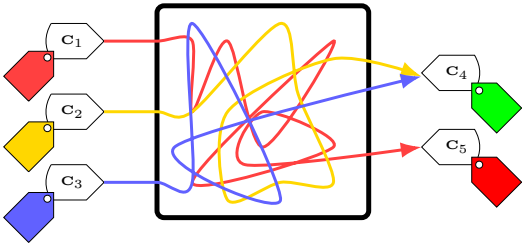
- Tag initialization
- Tag propagation
- Tag check



DIFT : how does it work ?

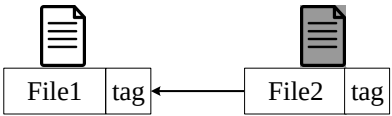
Three steps

- Tag initialization
- Tag propagation
- Tag check



Levels of IFT

- Application level



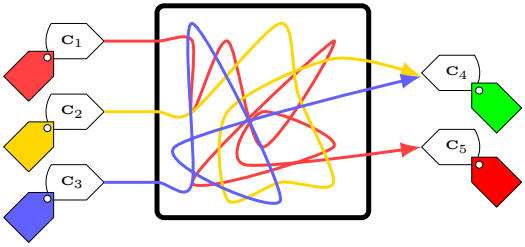
DIFT : how does it work ?

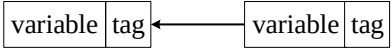
Three steps

- Tag initialization
- Tag propagation
- Tag check

Levels of IFT

- Application level
- OS level





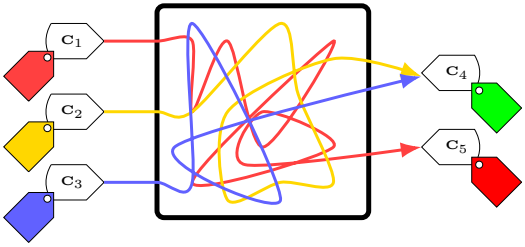
DIFT : how does it work ?

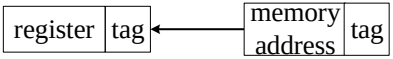
Three steps

- Tag initialization
- Tag propagation
- Tag check

Levels of IFT

- Application level
- OS level
- Architecture level





D-RI5CY

- Design [4] made by researchers at Columbia University (USA) in partnership with the University of Turin (Italy),
- Based on the 32-bit RISC-V processor: RI5CY (PULP platform),
- Flexible security policy that can be modified while an application is running.

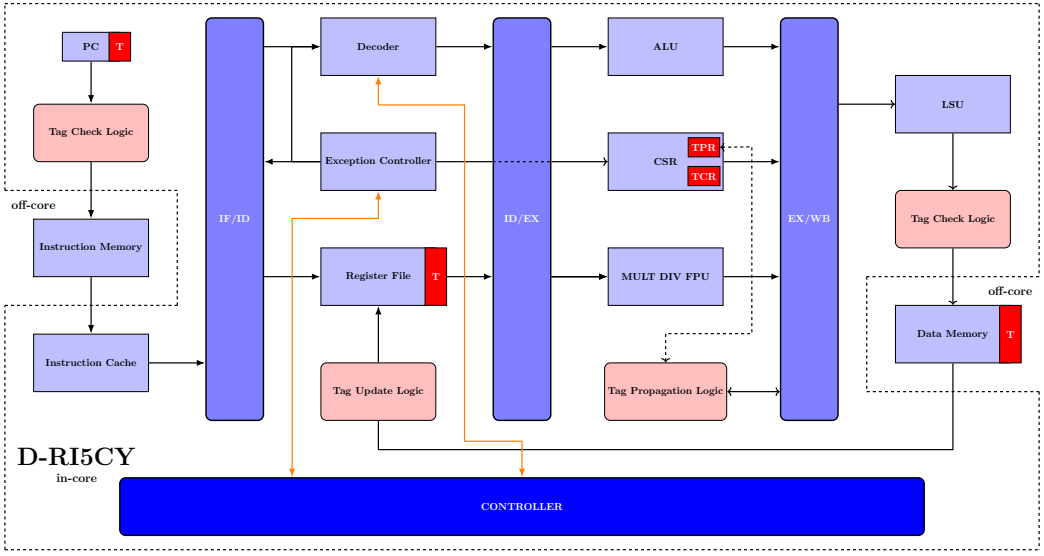


Figure 1: Architecture of the D-RI5CY. DIFT components in red and pink

Motivations

- Many studies have shown the vulnerabilities of critical systems against FIA :
  - Glitch injections : power supply to control the program counter [5],
  - EM Fault Injection (EMFI) : to recover an AES key by targeting the cache hierarchy and the MMU as shown in [6],
  - SCA/FIA : [7] have shown that you can combine side-channel attacks (SCA) and FIAs to bypass the PMP mechanism in a RISC-V processor

In this work

- We propose to study the use of physical attacks to defeat the DIFT mechanism implemented in the D-RI5CY processor to succeed software attacks.

Threat model

We consider an attacker able to

- perform physical attacks to defeat the DIFT mechanism and realize a software attack,
- inject faults in registers associated to the DIFT-related components:
  - set to 0,
  - set to 1,
  - bit-flips.

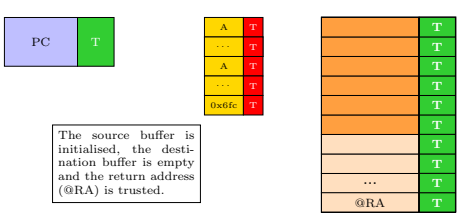
Outline

Vulnerability Assessment

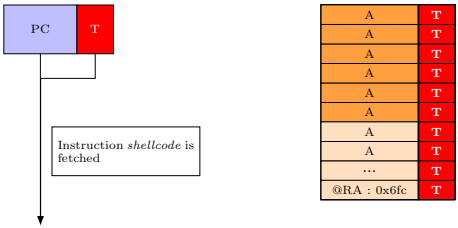
- Analysis of 3 cases: buffer overflow, format string, compare/compute.
- Analysis of tag propagation temporally and logically.
- Presentation of only 1 case in this presentation

Case: Buffer overflow

- The attacker exploits a buffer overflow to access the return address register (*ra*).



(a) Malicious buffer and *ra* trusted



(b) Overflow and overwriting of *ra* and its tag

- As the data in the source buffer is manipulated by the user, it is marked as *untrusted*.
- Thanks to DIFT, the tags associated with the source buffer data overwrite the *ra* register tag.
- When the function returns, the corrupted register *ra* is loaded into *PC* using a jalr instruction.

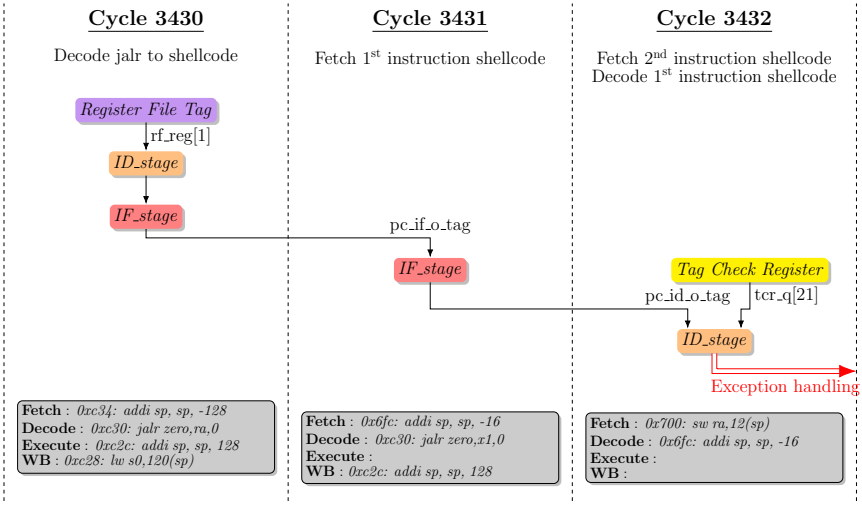


Figure 3: Temporal analysis of the tags propagation in *Buffer Overflow* attack

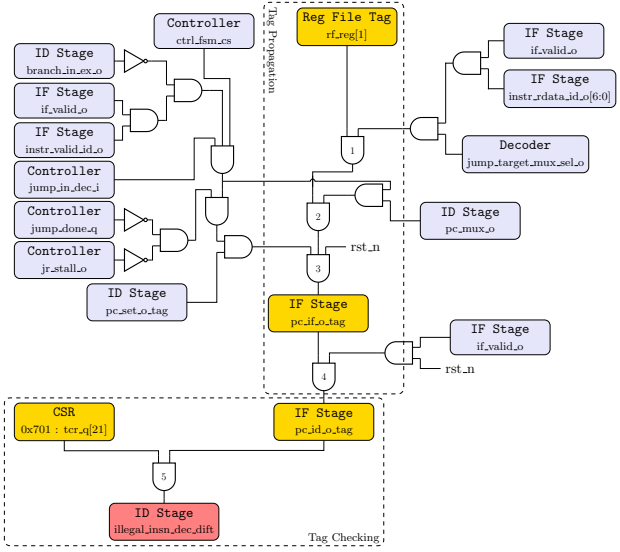


Figure 4: Logical analysis of the tags propagation in a *Buffer Overflow* attack

Outline

Experimental Setup - Simulation fault injections campaign

- Logical fault injection simulation is used for preliminary evaluations
  - faults are injected in the HDL code at cycle accurate and bit accurate level
  - a set of 55 DIFT-related registers are targeted
  - attack windows are determined based on the previous study
  - results are classed in four groups
    - crash: reference cycle count exceeded,
    - Nothing Significant To Report (NSTR)
    - delay: illegal instruction is delayed
    - success: DIFT has been bypassed
- Simulations with QuestaSim 10.6e.

Main results : 3 cases

Table 1: End of simulation status

	Crash	NSTR	Delay	Success	Total
Buffer overflow	0	1362	20	22 (1.57%)	1404
Format string	0	1743	77	52 (2.78%)	1872
Compare/Compute	0	905	12	19 (2.03%)	936

Buffer overflow

Table 2: Buffer overflow : Register sensitivity as determined by fault model and simulation time

	Cycle 3428			Cycle 3429			Cycle 3430			Cycle 3431			Cycle 3432		
	set0	set1	bitflip	set0	set1	bitflip	set0	set1	bitflip	set0	set1	bitflip	set0	set1	bitflip
pc_if_o_tag										✓		✓			
rf_reg[1]							✓		✓						
tcr_q	✓			✓			✓			✓			✓		
tcr_q[21]			✓			✓			✓			✓			✓
tpr_q	✓	✓		✓	✓										
tpr_q[12]			✓			✓									
tpr_q[15]			✓			✓									

Discussion

- 4212 simulations have been performed,
- 2.21% of the fault injections lead to successful attacks (93 successes),
- 34.41% are due to set to 0 fault type,
- 11.83% are due to set to 1 fault type,
- 53.76% are due to a bitflip,
- 2.59% of the simulated injections delay the DIFT exception
- 13 sensitive registers identified for all cases combined

Outline

Conclusion

- We have shown that the D-RI5CY DIFT mechanism is vulnerable to FIAs
- We identified 13 DIFT-related sensitive registers
- 93 simulated fault injections over 4212 have lead to a successful attack (2.21%)
- 2.59% of the simulated injections delay the DIFT exception

Perspectives

- In the future we will
  - implement and evaluate countermeasures to face fault injection attacks (simple parity, Hamming Code $\Rightarrow$ work in progress),
  - extend our study to take into account a more complex threat model (multi-faults models $\rightarrow$ open to discussion),
  - strengthen the analysis through actual fault injection campaign targeting a FPGA implementation.

Thank you for your attention.

If you have any questions, feel free to ask them now.

References

References

[1] K. Chen, X. Guo, Q. Deng, and Y. Jin, "Dynamic Information Flow Tracking: Taxonomy, Challenges, and Opportunities," *Micromachines*, 2021. DOI: 10.3390/mi12080898.

[2] C. Brant, P. Shrestha, B. Mixon-Baca, *et al.*, "Challenges and Opportunities for Practical and Effective Dynamic Information Flow Tracking," *ACM Computing Surveys (CSUR)*, 2021. DOI: 10.1145/3483790.

[3] W. Hu, A. Ardeshiricham, and R. Kastner, "Hardware Information Flow Tracking," *ACM Computing Surveys*, 2021. DOI: 10.1145/3447867.

[4] 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)*, 2018, pp. 1–7. DOI: 10.1109/HPEC.2018.8547578.

[5] N. Timmers, A. Spruyt, and M. Witteman, "Controlling PC on ARM Using Fault Injection," in *Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2016. DOI: 10.1109/FDTC.2016.18.

[6] T. Trouchkine, S. K. K. Bukasa, M. Escouteloup, R. Lashermes, and G. Bouffard, "Electromagnetic Fault Injection Against a Complex CPU, toward new Micro-architectural Fault Models," *Journal of Cryptographic Engineering*, 2021. DOI: 10.1007/s13389-021-00259-6.

References

[7] S. Nashimoto, D. Suzuki, R. Ueno, and N. Homma, "Bypassing Isolated Execution on RISC-V using Side-Channel-Assisted Fault-Injection and Its Countermeasure," *IACR Transactions on Cryptographic Hardware and Embedded Systems (TCHES)*, 2021. DOI: 10.46586/tches.v2022.i1.28-68.

Backup

Tag Propagation Register

Table 3: Tag Propagation Register configuration

	Load/Store Enable	Load/Store Mode	Logical Mode	Comparison Mode	Shift Mode	Jump Mode	Branch Mode	Arith Mode
Bit index	17 16 15	13 12	11 10	9 8	7 6	5 4	3 2	1 0
Policy V1	0 0 1	1 0	1 0	0 0	1 0	1 0	0 0	1 0
Policy V2	1 1 1	1 0	1 0	1 0	1 0	1 0	1 0	1 0

- A Mode field for each class of instructions which specifies how to propagate the tags of the input operands to the output operand tag.
  - the output tag keeps its old value (00);
  - the output tag is set to one, if both the input tags are set to one (01);
  - the output tag is set to one, if at least one input tag is set to one (10);
  - the output tag is set to zero (11).
- The three bits in the L/S enable field allow the policy to enable the source, source-address, and destination-address tags, respectively

Tag Check Register

Table 4: Tag Check Register configuration

	Execute Check	Load/Store Check	Logical Check	Comparison Check	Shift Check	Jump Check	Branch Check	Arith Check
Bit index	21	20 19 18 17	16 15 14	13 12 11	10 9 8	7 6 5	4 3	2 1 0
Policy V1	1	1 0 1 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0	0 0 0
Policy V2	0	0 0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0	0 1 1

- The tag-check rules restrict the operations that may be performed on tagged data. If the check bit for an operand tag is set to one and the corresponding tag is equal to one, an exception is raised.
  - For all the classes except Load/Store, there are three tags to consider: first input, second input, and output tags
  - For the Load/Store class there are four tags to take into account: source-address, source, destination-address, and destination tags
  - the additional Execute Check field is associated with the program counter and specifies whether to raise a security exception when the program-counter tag is set to one