

Enhanced Processor Defence Against Physical and Software Threats by Securing DIFT Against Fault Injection Attacks

PhD Dissertation Defense

William PENSEC

Université Bretagne Sud, UMR 6285, Lab-STICC, Lorient, France

December 19, 2024

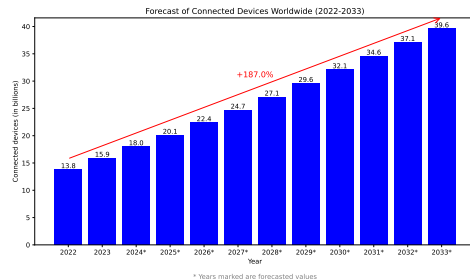
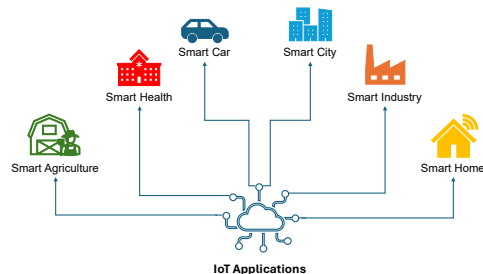


- 1 Introduction
- 2 D-RI5CY – Vulnerability Assessment
- 3 Fault Injection Simulation for Security Assessment
- 4 Proposed protections against FIAs
- 5 Experimental results
- 6 Conclusion and Perspectives

- 1 Introduction
 - Context
 - Motivations
 - Software threats: Information Flow Tracking
 - Hardware threats: Physical Attacks
 - Issue
 - Objectives

Internet of Things (IoT)

- Wide range of application
- Fast growing market with exponential usage
- Rely on sensors depending on their use
- Collect and share data
- Manipulation of critical data
- Increasingly vulnerable to multiple threats



Threats

- Software threats: malwares, memory overflow attacks, SQL injection, etc
- Network threats: DDoS, Man-In-The-Middle, jamming, etc
- Hardware threats: physical attacks such as reverse engineering, Side-Channel Attacks (SCA), Fault Injection Attacks (FIA)

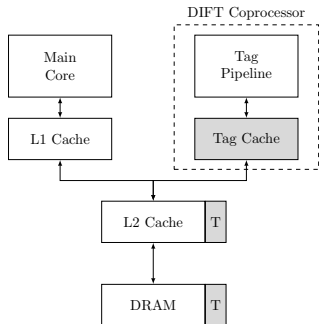
Threats

- **Software threats:** malwares, memory overflow attacks, SQL injection, etc
- Network threats: DDoS, man-in-the-middle, jamming, etc
- **Hardware threats:** physical attacks such as reverse engineering, Side-Channel Attacks (SCA), Fault Injection Attacks (FIA)

- Security mechanism
- Protection against software attacks (e.g.: *buffer overflow*, *format string*, *SQL injections*, ...) [1, 2]
- Static or Dynamic
- Software, Hardware or Hybrid
- Hardware DIFT: off-core, off-loading core, in-core

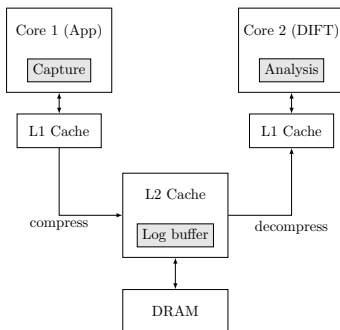
Software threats: Information Flow Tracking

- Security mechanism
- Protection against software attacks (e.g.: *buffer overflow*, *format string*, *SQL injections*, ...) [1, 2]
- Static or Dynamic
- Software, Hardware or Hybrid
- Hardware DIFT: **off-core**, off-loading core, in-core



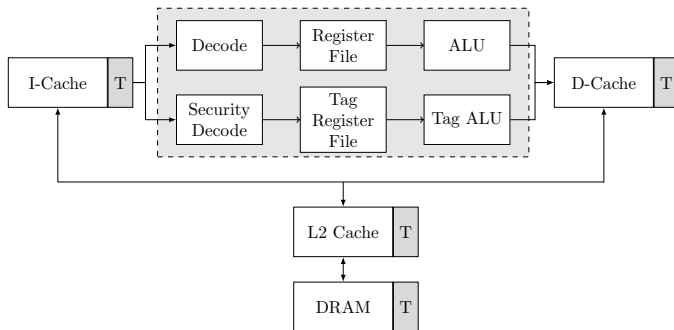
Software threats: Information Flow Tracking

- Security mechanism
- Protection against software attacks (e.g.: *buffer overflow*, *format string*, *SQL injections*, ...) [1, 2]
- Static or Dynamic
- Software, Hardware or Hybrid
- Hardware DIFT: off-core, **off-loading core**, in-core



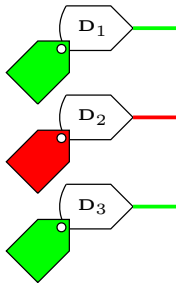
Software threats: Information Flow Tracking

- Security mechanism
- Protection against software attacks (e.g.: *buffer overflow*, *format string*, *SQL injections*, ...) [1, 2]
- Static or Dynamic
- Software, Hardware or Hybrid
- Hardware DIFT: off-core, off-loading core, **in-core**



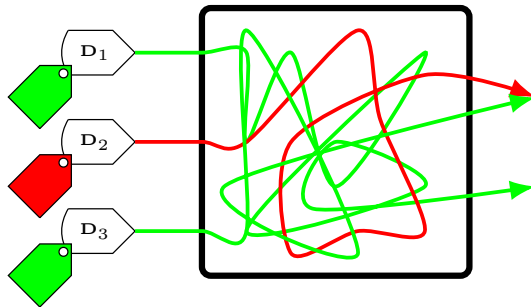
Three steps

- Tag initialisation
- Tag propagation
- Tag check



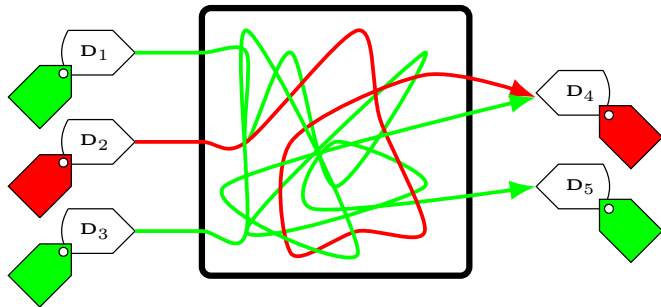
Three steps

- Tag initialisation
- Tag propagation
- Tag check



Three steps

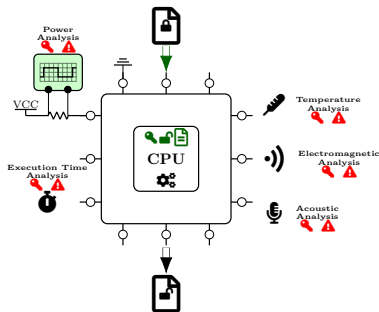
- Tag initialisation
- Tag propagation
- Tag check



- Reverse Engineering: process of information retrieval from a product by analysing and understanding the design, functionality, and operation of existing hardware
- Side-Channel Attacks: exploit information leakages on the circuit behaviour
- Fault Injection Attacks: involve deliberately introducing one or more fault(s) into the system to observe its behaviour and identify potential vulnerabilities.

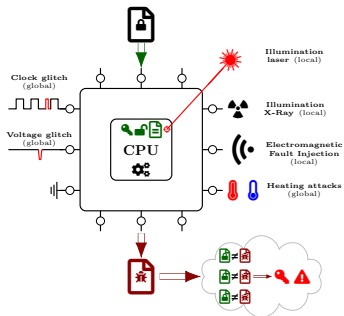
Hardware threats: Physical Attacks

- Reverse Engineering: process of information retrieval from a product by analysing and understanding the design, functionality, and operation of existing hardware
- Side-Channel Attacks: exploit information leakages on the circuit behaviour
- Fault Injection Attacks: involve deliberately introducing one or more fault(s) into the system to observe its behaviour and identify potential vulnerabilities.



Hardware threats: Physical Attacks

- Reverse Engineering: process of information retrieval from a product by analysing and understanding the design, functionality, and operation of existing hardware
- Side-Channel Attacks: exploit information leakages on the circuit behaviour
- Fault Injection Attacks: involve deliberately introducing one or more fault(s) into the system to observe its behaviour and identify potential vulnerabilities.



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

Contributions

- ▶ Provide a robust security mechanism against software and hardware threats.
- ▶ Take into account Fault Injection Attacks
- ▶ Propose lightweight countermeasures against FIA
- ▶ Take into account constraints, such as area and performance overhead

- 2 D-RI5CY – Vulnerability Assessment
 - D-RI5CY - origins and architecture
 - Vulnerability assessment
 - Use case : presentation
 - Experimental Setup

- Design [3] made by researchers at Columbia University (USA) with Politecnico di Torino (Italy)
- Based on the 32-bit RISC-V processor: RI5CY (Pulp Platform)
- Open source¹
- 1-bit tag datapath
- Flexible security policy that can be modified at runtime



¹<https://github.com/sld-columbia/riscv-dift>

D-RI5CY - architecture

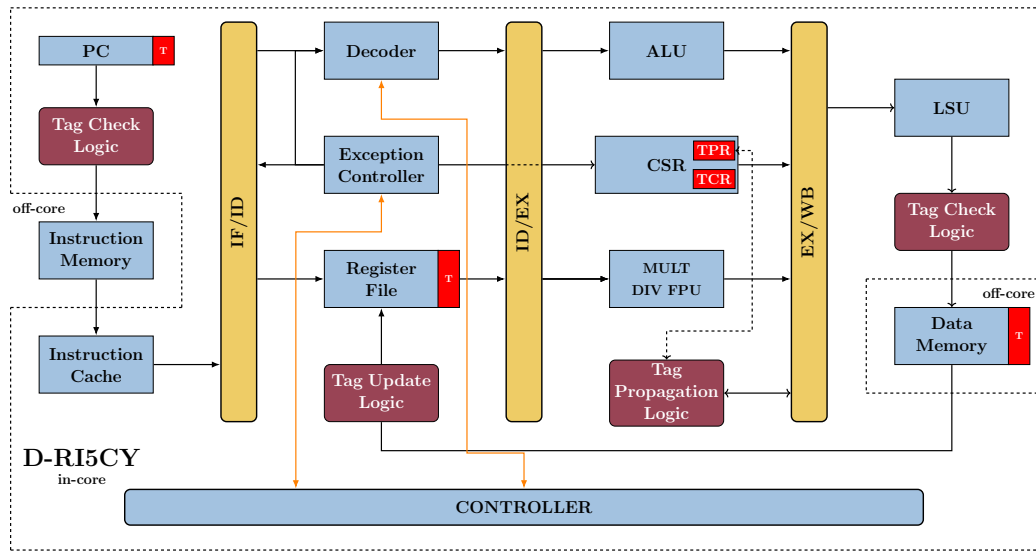


Figure 1: Architecture of the D-RI5CY.

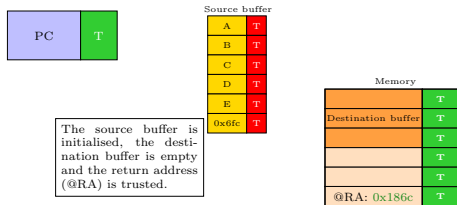
Threat model

We consider an attacker able to:

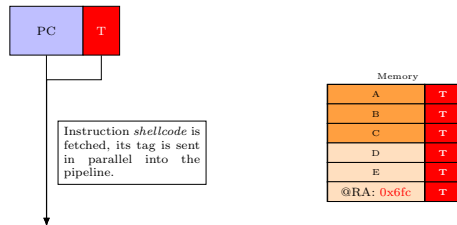
- perform a physical attack to defeat the DIFT mechanism and realise a software attack,
- inject faults in DIFT-related registers:
 - bit set,
 - bit reset,
 - bit-flip.

Case 1: 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 *jlr* instruction.

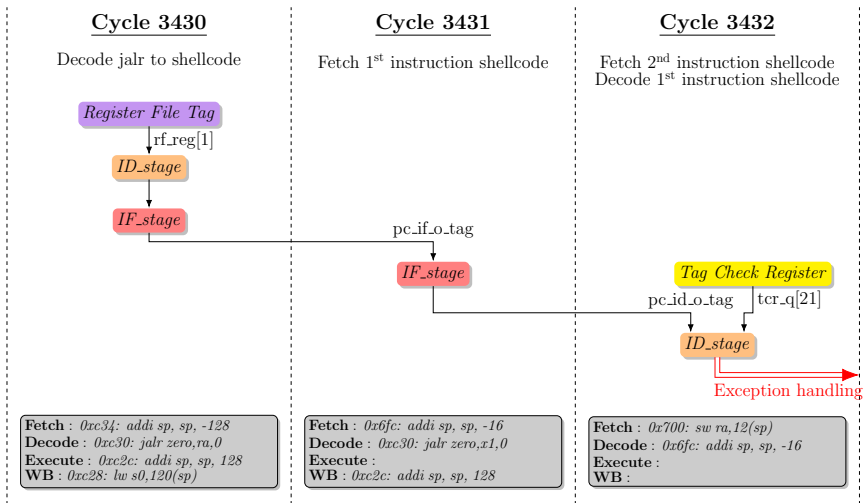


Figure 3: Temporal analysis of the tags propagation in *Buffer Overflow* attack

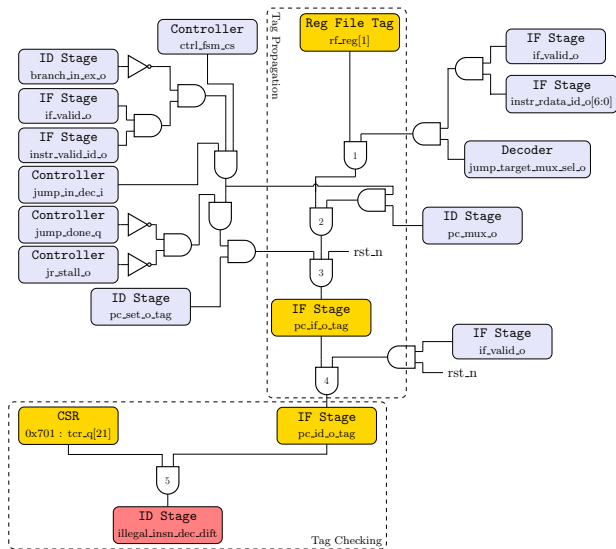


Figure 4: Logical analysis of the tags propagation in a *Buffer Overflow* attack

- 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
 - a reference simulation is done without fault
 - results are classed in four groups
 - crash: reference cycle count exceeded,
 - silent: current faulted simulation is the same as the reference simulation
 - delay: illegal instruction is delayed
 - success: DIFT has been bypassed
- Simulations with QuestaSim 10.6e.
- FISSA is used in order to create our injection campaigns

Table 1: End of simulation status

	Crash	NSTR	Delay	Success	Total
Buffer overflow	0	1380	20	24 (1.69%)	1422
WU-FTPd	0	1767	77	52 (2.74%)	1896
Compare/Compute	0	917	12	19 (2.00%)	948

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										✓		✓			
memory_set_o_tag		✓	✓												
rf_reg[1]							✓		✓						
tcr_q	✓			✓			✓			✓			✓		
tcr_q[21]			✓			✓			✓			✓			✓
tpr_q	✓	✓		✓	✓										
tpr_q[12]			✓			✓									
tpr_q[15]			✓			✓									

- ▶ 4266 simulations have been performed,
- ▶ 95 successes (2.23%),
- ▶ We have shown that the D-RI5CY DIFT is vulnerable to FIA
- ▶ Propagation of faults is facilitated by paths fully made of *AND* gates

-
-
- 3 Fault Injection Simulation for Security Assessment

- quelques mots sur l'outil
- parler à la première du singulier
- images d'Architecture
- lien github
- citation vers articles
- aller vite vers résultat (pas d'exemples, trop long ...)

- 4 Proposed protections against FIAs
 - Simple Parity
 - Hamming Code
 - Hamming Code - SECDED

Protections

- We propose 3 lightweight countermeasures using parity codes
 - Simple parity
 - Hamming Code
 - Hamming Code with an additional bit (SECDED)

Detection of single-bit errors - Simple Parity

- Focusing into lightweight protections for small systems
- Often used for error detection.
- Add an extra bit for parity computation.
- Can only detect one error without correction.

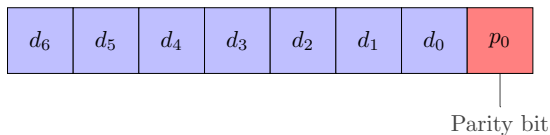


Figure 5: Simple parity codeword

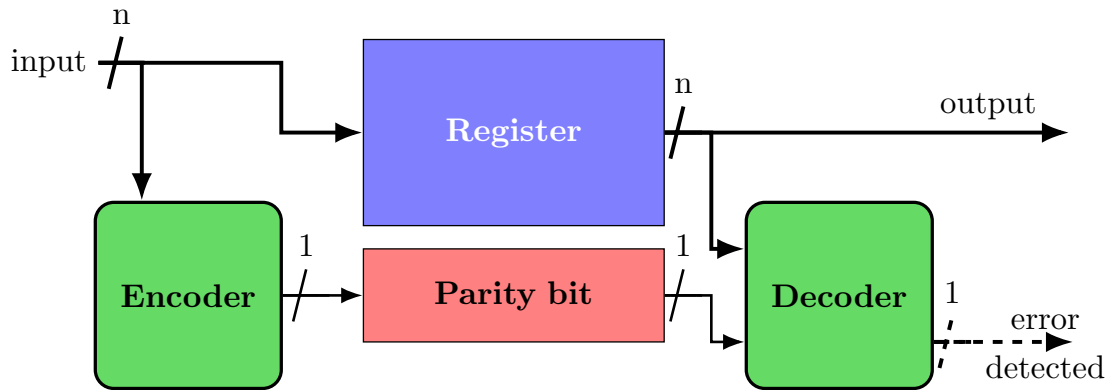


Figure 6: Simple Parity implementation

Detection and correction of single-bit errors - Hamming Code

- Linear error-correcting codes invented by Richard W. Hamming [4].
- Mostly used in digital communication and data storage systems.
- Detect and correct single-bit error.
- Redundancy bits are placed in power of 2 positions.
- The number of redundancy bits depends on the number of bits to be protected ($2^r \geq m + r + 1$)

$$r_0 = d_0 \oplus d_1 \oplus d_3 \oplus d_4 \oplus d_6$$

$$r_1 = d_0 \oplus d_2 \oplus d_3 \oplus d_5 \oplus d_6$$

$$r_2 = d_1 \oplus d_2 \oplus d_3$$

$$r_3 = d_4 \oplus d_5 \oplus d_6$$

(1)

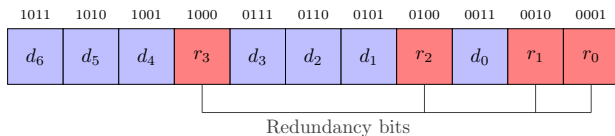


Figure 7: Hamming codeword

Implementation - Hamming Code

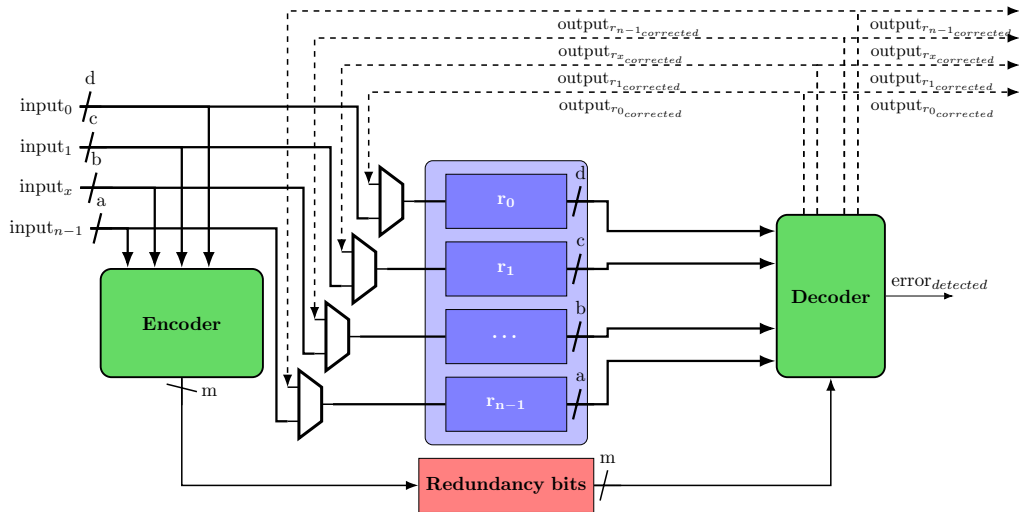


Figure 8: Hamming Code implementation for independent registers

Implementation - Hamming Code

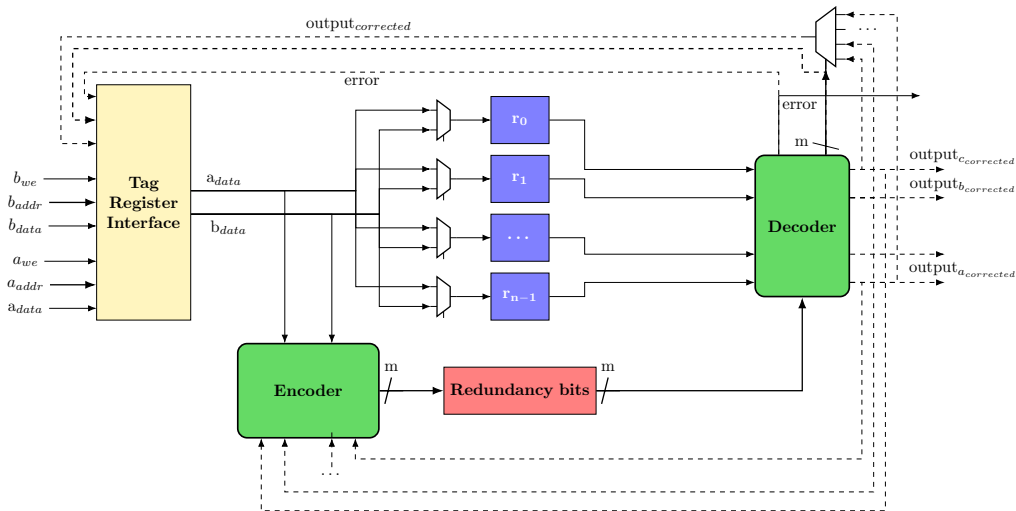


Figure 9: Hamming Code implementation for Register File

Detection of two-bit errors and correction of single-bit errors - SECDED

- Based on Hamming Code.
- Detect two-bit error and correct single-bit error.
- The number of redundancy bits depends on the number of bits to be protected ($2^r \geq m + r + 1$).
- An additional bit is placed at index 0, it is called: general parity bit

$$r_0 = d_0 \oplus d_1 \oplus d_3 \oplus d_4 \oplus d_6$$

$$r_1 = d_0 \oplus d_2 \oplus d_3 \oplus d_5 \oplus d_6$$

$$r_2 = d_1 \oplus d_2 \oplus d_3$$

$$r_3 = d_4 \oplus d_5 \oplus d_6$$

$$gp_0 = \bigoplus_{i=0}^6 d_i \oplus \bigoplus_{j=0}^3 r_j$$

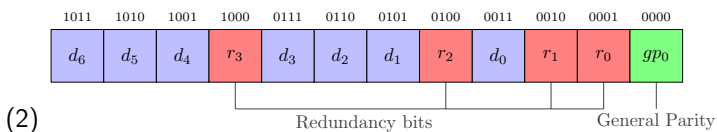


Figure 10: SECDED codeword

Implementation - SECDED

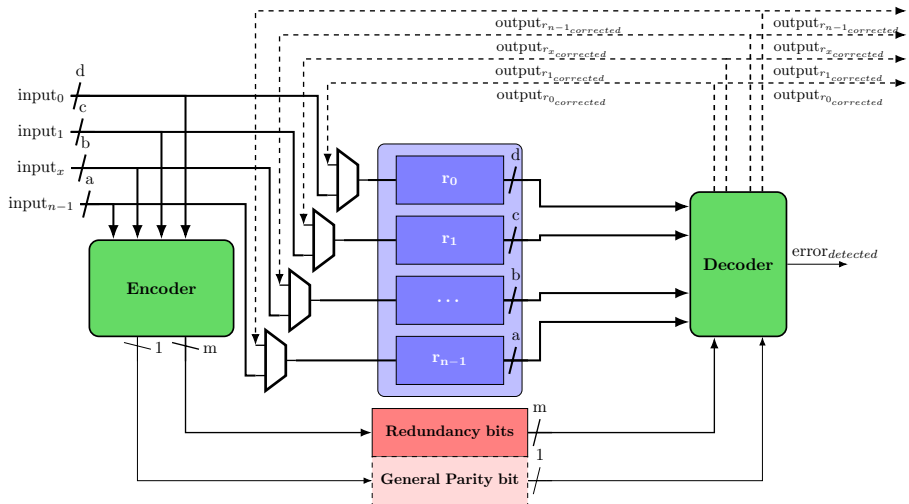


Figure 11: SECDED implementation for independent registers

Implementation - SECDED

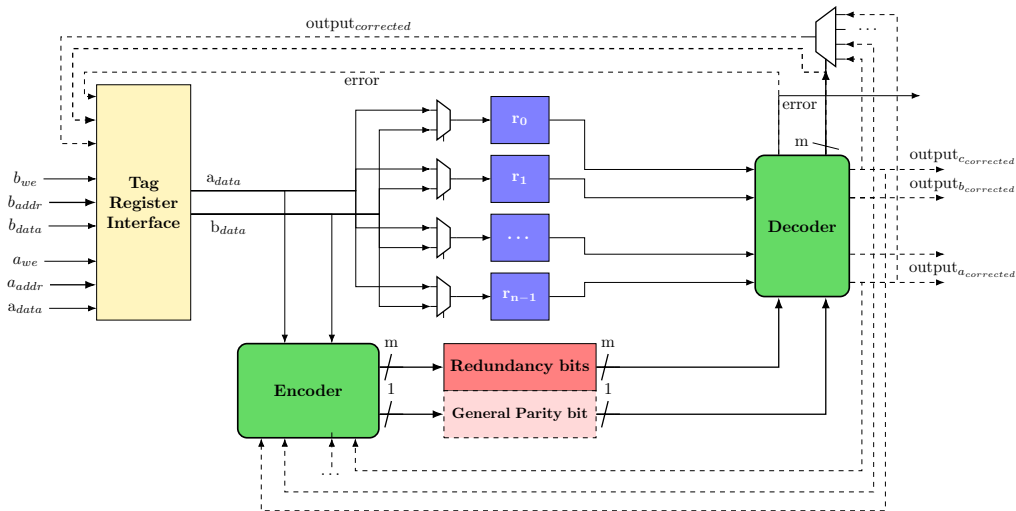


Figure 12: SECDED implementation for Register File

Threat model

- DIFT-related registers + protection-related registers
- Single bit-flip in one register
- Single bit-flip in two registers at two distinct clock cycles
- Single bit-flip in two registers at a given clock cycle
- Multi-bit faults in one register at a given clock cycle
- Multi-bit faults in two registers at a given clock cycle

Table 3: Grouping composition of implemented strategies

	Grouping composition	Number of groups	Number of registers
Baseline	–	–	55
Strategy 1	Minimisation of groups	5	65
Strategy 2	Protection per pipeline stage	7	69
Strategy 3	Protection per register	24	103
Strategy 4	Protection per register with slicing of CSR	38	131
Strategy 5	Coupling sliced registers	39	133

Table 4: Summary of DIFT-related protected registers – taking SECDED

	Number of protected bits	Number of redundancy bits	Number of parity bits	Number of bits
Strategy 1	107	25	5	157
Strategy 2	107	30	7	164
Strategy 3	107	64	24	215
Strategy 4	103	101	38	266
Strategy 5	102	114	39	280

- 5 Experimental results

Table 5: FPGA implementation results — Vivado 2023.2

Protection	Number of LUTs	Number of FFs	Maximum frequency
Baseline	6,597 (-4.54%)	2,211 (-5.31%)	49.1 MHz (3%)
D-RI5CY	6,911 (0%)	2,335 (0%)	47.6 MHz (0%)
Simple parity	7,011 (1.45%)	2,337 (0.09%)	47.6 MHz (0%)
Hamming Code	7,283 (5.38%)	2,361 (1.11%)	47.4 MHz (-0.36%)
SECDED	7,428 (7.48%)	2,366 (1.33%)	47.2 MHz (-0.95%)

- 6 Conclusion and Perspectives
 - Conclusion
 - Perspectives

Publications

International peer-reviewed conferences

- ① **William Pensec**, Vianney Lapôte, and Guy Gogniat. 2023. Another Break in the Wall: Harnessing Fault Injection Attacks to Penetrate Software Fortresses. In Proceedings of the First International Workshop on Security and Privacy of Sensing Systems (SensorsS&P), 2023. [5]
- ② **William Pensec**, Francesco Regazzoni, Vianney Lapôte, and Guy Gogniat. Defending the Citadel: Fault Injection Attacks Against Dynamic Information Flow Tracking and Related Countermeasures. 2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2024, pp. 180-185. [6]
- ③ **William Pensec**, Vianney Lapôte, and Guy Gogniat. Scripting the Unpredictable: Automate Fault Injection in RTL Simulation for Vulnerability Assessment. 2024 27th Euromicro Conference on Digital System Design (DSD), 2024. [7]

Enhanced Processor Defence Against Physical and Software Threats by Securing DIFT Against Fault Injection Attacks

PhD Dissertation Defense

William PENSEC

Thank you for your attention.



References

- [1] Christopher Brant et al. “Challenges and Opportunities for Practical and Effective Dynamic Information Flow Tracking”. In: *ACM Computing Surveys* 55.1 (Nov. 2021). ISSN: 0360-0300. DOI: [10.1145/3483790](https://doi.org/10.1145/3483790).
- [2] Wei Hu, Armaiti Ardesiricham, and Ryan Kastner. “Hardware Information Flow Tracking”. In: *ACM Computing Surveys* (2021). DOI: [10.1145/3447867](https://doi.org/10.1145/3447867).
- [3] Christian Palmiero et al. “Design and Implementation of a Dynamic Information Flow Tracking Architecture to Secure a RISC-V Core for IoT Applications”. In: *High Performance Extreme Computing*. 2018. DOI: [10.1109/HPEC.2018.8547578](https://doi.org/10.1109/HPEC.2018.8547578).
- [4] R. W. Hamming. “Error detecting and error correcting codes”. In: *The Bell System Technical Journal* (1950). DOI: [10.1002/j.1538-7305.1950.tb00463.x](https://doi.org/10.1002/j.1538-7305.1950.tb00463.x).
- [5] William Pensec, Vianney Lapôte, and Guy Gogniat. “Another Break in the Wall: Harnessing Fault Injection Attacks to Penetrate Software Fortresses”. In: *Proceedings of the First International Workshop on Security and Privacy of Sensing Systems*. SensorsS&P. Istanbul, Turkiye: Association for Computing Machinery, 2023, pp. 8–14. DOI: [10.1145/3628356.3630116](https://doi.org/10.1145/3628356.3630116).
- [6] William PENSEC et al. “Defending the Citadel: Fault Injection Attacks Against Dynamic Information Flow Tracking and Related Countermeasures”. In: *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. Knoxville, United States, July 2024, pp. 180–185. DOI: [10.1109/ISVLSI61997.2024.00042](https://doi.org/10.1109/ISVLSI61997.2024.00042).
- [7] William Pensec, Vianney Lapôte, and Guy Gogniat. “Scripting the Unpredictable: Automate Fault Injection in RTL Simulation for Vulnerability Assessment”. In: *2024 27th Euromicro Conference on Digital System Design (DSD)*. Paris, France, Aug. 2024, pp. 369–376. DOI: [10.1109/DSD64264.2024.00056](https://doi.org/10.1109/DSD64264.2024.00056).

- [8] Transforma Insights; Exploding Topics. *Number of Internet of Things (IoT) connections worldwide from 2022 to 2023, with forecasts from 2024 to 2033*. Online. Accessed 13 August 2024. 2024. URL: <https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/>.

Backup

Table 6: 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

Table 7: 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

- The vulnerability is the use of an unchecked user input as the format string parameter in functions that perform formatting, e.g. `printf()`
- An attacker can use the format tokens, to write into arbitrary locations of memory, e.g. the return address of the function.

```
void echo(){  
    int a;  
    register int i asm("x8");  
    a = i;  
    printf("%224u%n%35u%n%253u%n%n", 1, (int*) (a-4), 1, (int*) (a-3), 1, (int*) (a-2), (int*) (a-1));  
}
```

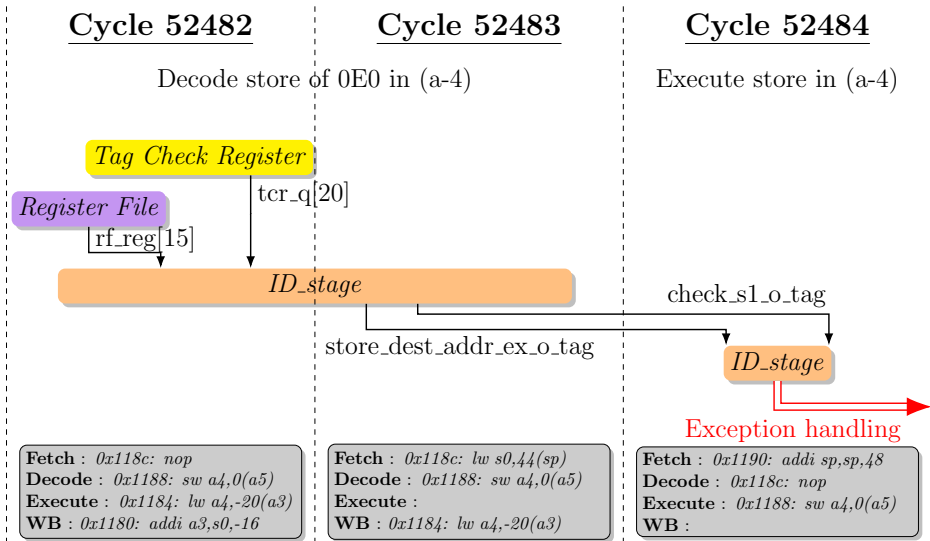


Figure 13: Temporal analysis of the tags propagation in a *format string* attack

Case 3: Compare/Compute

- No software vulnerability
- Used to cover the DIFT surface

```
int main(){
    int a, b = 5, c;
    register int reg asm("x9");
    a = reg;
    asm volatile("csrw 0x700, tprValue");
    asm volatile("csrw 0x701, tcrValue");
    asm volatile("p.spsw x0, 0(\\%0);" :: "r" (&a));
    c = (a > b) ? (a-b) : (a+b);
    //42c:    ble a4,a5,448
    //430:    addi a5,s0,-16
    //434:    lw a4,-12(a5)
    //438:    addi a3,s0,-16
    //43c:    lw a5,-4(a3)
    //440:    sub a5,a4,a5
    //444:    j 45c
    //448:    addi a5,s0,-16
    //44c:    lw a4,-12(a5)
    //450:    addi a3,s0,-16
    //454:    lw a5,-4(a3)
    //458:    add a5,a4,a5
    //45c:    sw a5,-24(s0)
    return EXIT_SUCCESS;
}
```

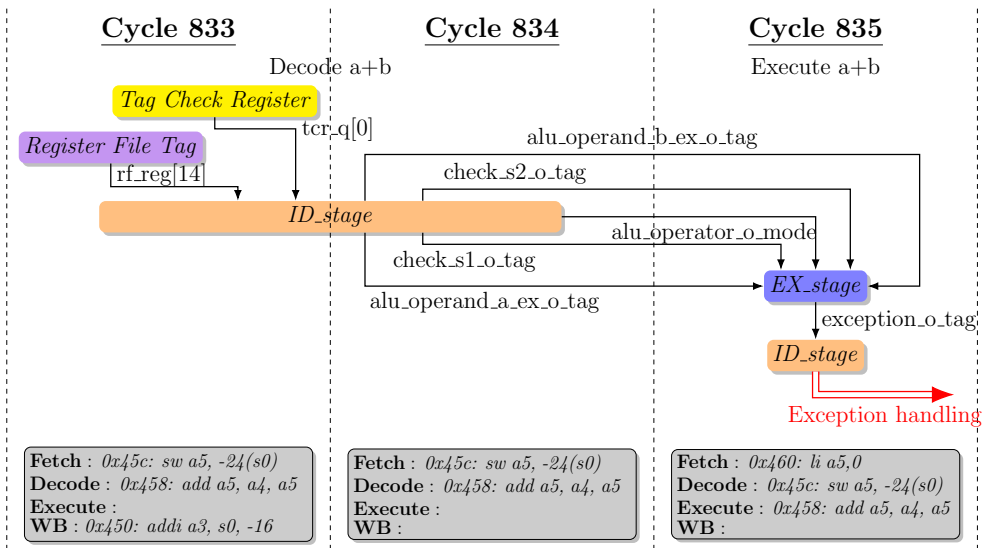


Figure 15: Temporal analysis of the tags propagation in a *format string* attack

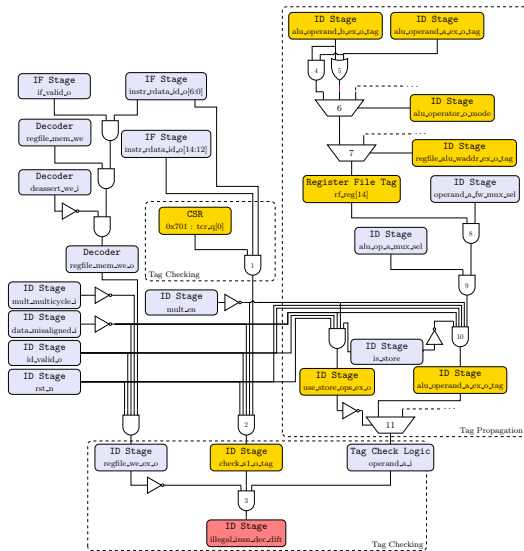


Figure 16: Logical analysis of the tags propagation in a *format string* attack