When in-core Dynamic Information Flow Tracking faces fault injection attacks

Vianney Lapôtre, William Pensec, Guy Gogniat

CryptArchi 2023

June 13, 2023

Table of Contents

Introduction

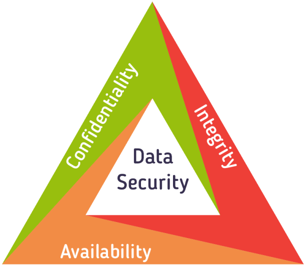
- ▶ Introduction
- ▶ D-RI5CY processor
- ▶ Fault Injection Attacks against D-RI5CY
- ▶ Conclusions

Data security: principles

Introduction

Principles

- **Confidentiality**
- **Integrity**
- Availability



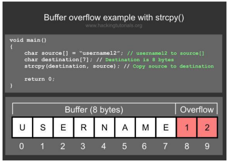
Security Policy

- Which security property is expected on each information container (file, variable, register, etc.) ?
- What operations are allowed on each container ?

Threat model

Introduction

- Software attacks: buffer overflow, ROP…

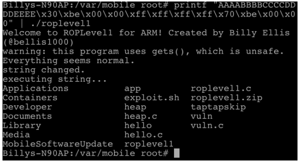


Buffer overflow example with strcpy()

www.hackingtutorials.org

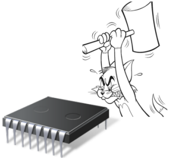
```
void main()
{
    char source[] = "username12"; // username12 to source[]
    char destination[7]; // Destination is 8 bytes
    strcpy(destination, source); // Copy source to destination

    return 0;
}
```



```
Billys-N90AP:/var/mobile root# printf "AAAABBBBCCCCDD
DDEEEE\x30\xbe\x00\x00\xff\xff\xff\xff\x70\xbe\x00\x0
0" | ./roplevel1
Welcome to ROPLevel1 for ARM! Created by Billy Ellis
(@bellis1000)
warning: this program uses gets(), which is unsafe.
Everything seems normal.
string changed.
executing string...
Applications          app          roplevel1.c
Containers            exploit.sh   roplevel1.zip
Developer             heap         taptapskip
Documents             heap.c       vuln
Library               hello        vuln.c
Media                 hello.c
MobileSoftwareUpdate  roplevel1
Billys-N90AP:/var/mobile root# █
```

- Fault injection attacks



- Side-channel attacks not taken into account

Software security: Existing solutions

Introduction

Security mechanisms

Detect, prevent or recover from a security attack

Preventive mechanisms

Enforce the security policy:

- Cryptographic mechanisms
- Isolation (e.g., Trustzone, SAM L11)
- Formal proof, etc.

Reactive mechanisms

Monitor the system and detect any security policy violation to recover

- Intrusion detection systems (e.g., Snort, OSSEC)
  - **Dynamic Information flow tracking (DIFT)**

Dynamic Information Flow Tracking (DIFT)

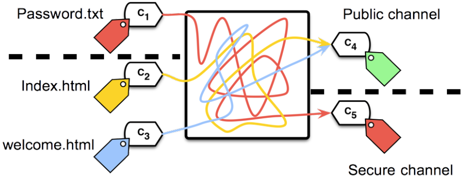
Introduction

Motivation

DIFT for security purposes : Integrity and Confidentiality

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**



Hardware-based DIFT (fine-grained)

Introduction

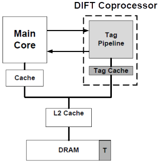


Figure: Dedicated DIFT co-processor [3, 1]

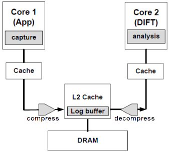


Figure: Dedicated CPU for DIFT [4]

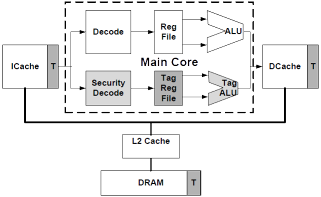


Figure: In-core DIFT [2, 5]

Table of Contents

D-RI5CY processor

▶ Introduction

▶ D-RI5CY processor

▶ Fault Injection Attacks against D-RI5CY

▶ Conclusions

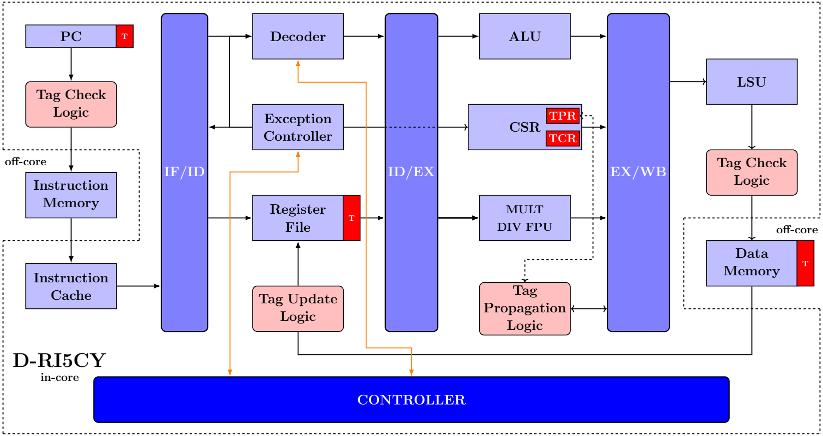
Overview

D-RI5CY processor

- Fork of the RI5CY processor [6]
  - 4-stage in-order 32-bit RISC-V optimized for low-power embedded systems and IoT application
  - Fully supports the base integer instruction set (RV32I), compressed instructions (RV32C) and the multiplication instruction set extension (RV32M) of the RISC-V ISA. In addition, it implements a set of custom extensions (RV32XPulp)
- *The D-RI5CY must be able to detect and stop various known memory-corruption attacks; the protection must be flexible and extendable through software programmable security policies to target future kinds of attacks; finally, the protection must provide a transparent and fine-grain management of security with no latency and small storage overhead*

Block diagram

D-RI5CY processor



- In red and pink the DIFT components

Tag initialization

D-RI5CY processor

- To initialize the security tags of user-supplied inputs to one, four new instructions have been implemented
  - **p.set rd** sets to one the security tag of the destination register `rd`;
  - **p.spsb x0, offset(rt)** sets to one the security tag of the memory byte at the address `rt + offset`;
  - **p.spsh x0, offset(rt)** sets to one the security tags of the memory half-word at the address `rt + offset`;
  - **p.spsw x0, offset(rt)** sets to one the security tags of the memory word at the address `rt + offset`.

Tag Propagation

D-RI5CY processor

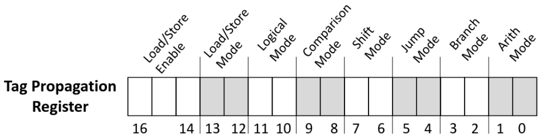


Figure: D-RI5CY Tag Propagation Register [5]

- 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 Checking

D-RI5CY processor

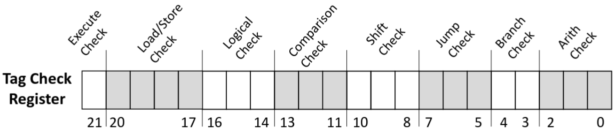


Figure: D-RI5CY Tag Check Register [5]

- 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

Table of Contents

Fault Injection Attacks against D-RI5CY

- ▶ Introduction
- ▶ D-RI5CY processor
- ▶ **Fault Injection Attacks against D-RI5CY**
- ▶ Conclusions

Motivation and fault model

Fault Injection Attacks against D-RI5CY

- Identify vulnerabilities of the D-RI5CY DIFT mechanism when considering FIA and propose contermeasures
- We consider an attacker able to
  - combine software and physical attacks to defeat the DIFT mechanism
  - inject faults in registers associated to the DIFT-related components
    - set to 0, set to 1, or a bit-flip at a random position of the targeted register
- In this presentation, we consider 2 use cases: buffer overflow and Format string attacks

Buffer overflow attack

Fault Injection Attacks against D-RI5CY

- The attacker exploits a buffer overflow to reach the return address (ra) register

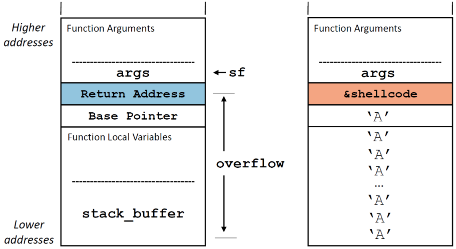
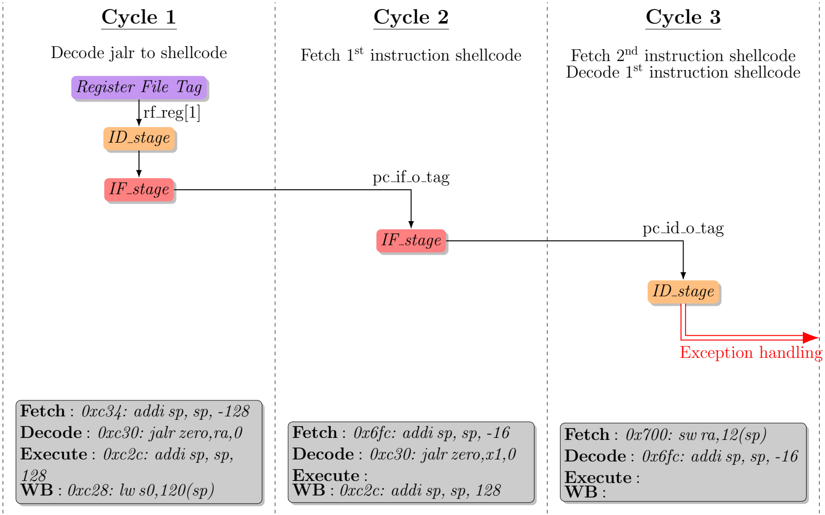


illustration from [5]

- Due to the DIFT mechanism, the tag associated with the buffer data overwrites the `ra` register tag.
- Since the buffer data is manipulated by the user, it is tagged as *not trusted*.
- When returning from the called function, the corrupted `ra` register is loaded into `PC` via a `jalr` instruction.

Tag propagation in a buffer overflow attack

Fault Injection Attacks against D-RI5CY



Cycle 1

Decode jalr to shellcode

Fetch : *0xc34: addi sp, sp, -128*
Decode : *0xc30: jalr zero,ra,0*
Execute : *0xc2c: addi sp, sp, 128*
WB : *0xc28: lw s0,120(sp)*

Cycle 2

Fetch 1st instruction shellcode

Fetch : *0x6fc: addi sp, sp, -16*
Decode : *0xc30: jalr zero,x1,0*
Execute :
WB : *0xc2c: addi sp, sp, 128*

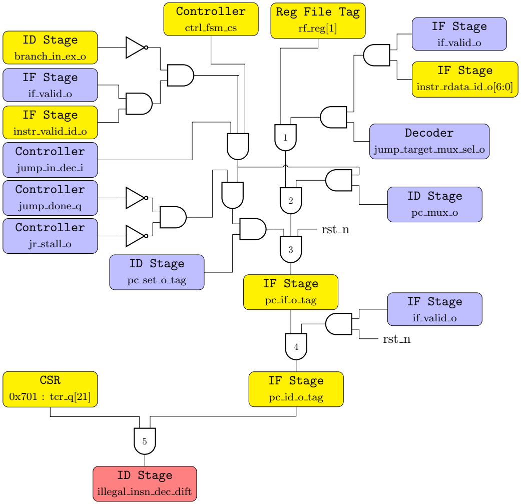
Cycle 3

Fetch 2nd instruction shellcode
Decode 1st instruction shellcode

Fetch : *0x700: sw ra,12(sp)*
Decode : *0x6fc: addi sp, sp, -16*
Execute :
WB :

Tag propagation in a buffer overflow attack - logic view

Fault Injection Attacks against D-RI5CY



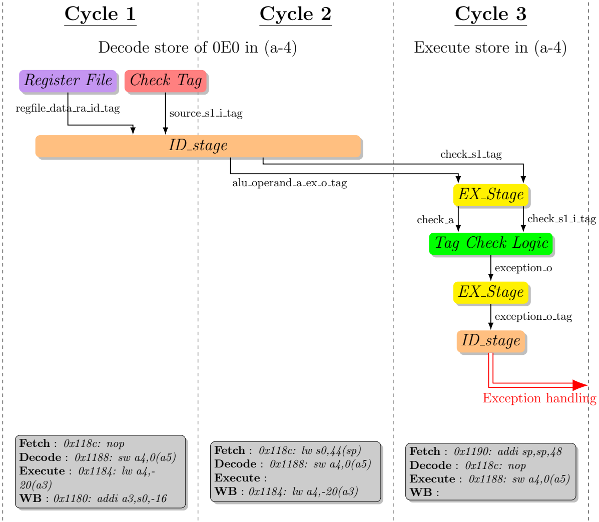
Format string attack

Fault Injection Attacks against D-RI5CY

- 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.
- We consider the example of Format string attack available at https://github.com/sld-columbia/riscv-dift/tree/master/pulpino_apps_dift/wu-ftpd

Tag propagation in a Format string attack

Fault Injection Attacks against D-RI5CY



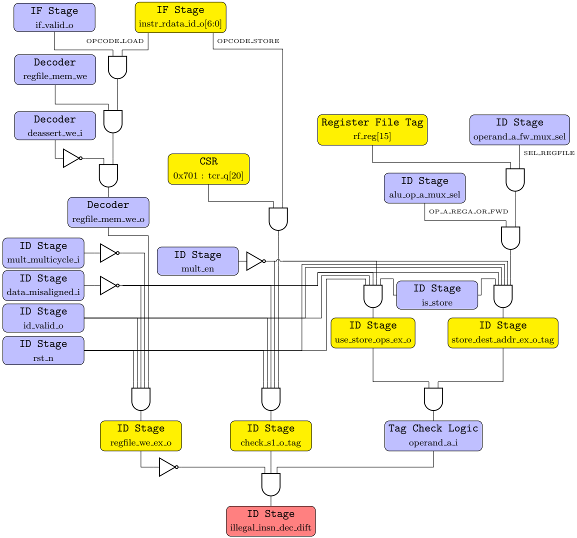
Fetch : *0x118c: nop*
Decode : *0x1188: sw a4,0(a5)*
Execute : *0x1184: lw a4,-20(a3)*
WB : *0x1180: addi a3,s0,-16*

Fetch : *0x118c: lw s0,44(sp)*
Decode : *0x1188: sw a4,0(a5)*
Execute :
WB : *0x1184: lw a4,-20(a3)*

Fetch : *0x1190: addi sp,sp,48*
Decode : *0x118c: nop*
Execute : *0x1188: sw a4,0(a5)*
WB :

Tag propagation in a Format string attack - logic view

Fault Injection Attacks against D-RI5CY



fault simulation campaign

Fault Injection Attacks against D-RI5CY

- 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 54 DIFT-related registers are targeted
  - a set of attack windows are determined based on the previous study
  - set to 0, set to 1, or a bit-flip at a random position are considered
  - 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

fault simulation campaign - main results

Fault Injection Attacks against D-RI5CY

Table: Fault simulations end status

	Crash	NSTR	Delay	Success	Total
Buffer overflow	0	940	17	15	972
Format string	0	1036	69	29	1134

fault simulation campaign - Buffer overflow

Fault Injection Attacks against D-RI5CY

Table: Buffer overflow: success per register, fault type and simulation time

	137140 ns		137180 ns			137220 ns		137260 ns		137300 ns
	set to 0	set to 1	set to 0	set to 1	bitflip	set to 0	bitflip	set to 0	bitflip	set to 0
pc_if_o_tag						✓		✓	✓	
rf_reg[1]						✓	✓			
tcr_q	✓		✓		✓	✓		✓		✓
tpr_q	✓	✓	✓	✓						

fault simulation campaign - format string

Fault Injection Attacks against D-RI5CY

Table: Format string attack: success per register, fault type and simulation time

	2099140 ns			2099180 ns		2099220 ns		2099260 ns		2099300 ns		2099340 ns		2099380 ns	
	set to 0	set to 1	bitflip	set to 0	set to 1	set to 0	set to 1	set to 0	set to 1	set to 0	bitflip	set to 0	bitflip	set to 0	bitflip
alu_operand_b_ex_o_tag	✓		✓												
alu_operator_o_mode	✓	✓	✓												
check_s1_o_tag												✓		✓	✓
store_dest_addr_ex_o_tag												✓		✓	✓
use_store_ops_ex_o												✓		✓	✓
rf_reg[15]										✓	✓	✓	✓		
tcr_q	✓			✓		✓		✓		✓		✓			
tpr_q		✓	✓		✓		✓		✓						

Table of Contents

Conclusions

- ▶ Introduction
- ▶ D-RI5CY processor
- ▶ Fault Injection Attacks against D-RI5CY
- ▶ Conclusions

- We have shown that the D-RI5CY DIFT mechanism is vulnerable to FIAs
- We identified 12 DIFT-related sensitive registers
- 72 simulated fault injections over 3726 have lead to a successful attack (1.93%)

- In future works we will
  - Strengthen the proposed analysis through actual fault injection campaign targeting a FPGA implementation
  - Propose a robust in-core DIFT mechanism against FIAs

When in-core Dynamic Information Flow Tracking faces fault injection attacks

Many thanks to William Pensec for his work
Thank you for listening!
Any questions?

References

Conclusions

[1] Abdul Wahab, M., Cotret, P., Nasr Allah, M., Hiet, G., Lapotre, V., and Gogniat, G.
Towards a hardware-assisted information flow tracking ecosystem for ARM processors.
In *26th International Conference on Field-Programmable Logic and Applications (FPL 2016)* (Lausanne, Switzerland, Aug. 2016).

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

[3] Kannan, H., Dalton, M., and Kozyrakis, C.
Decoupling dynamic information flow tracking with a dedicated coprocessor.
In *Dependable Systems & Networks, 2009.* (2009), IEEE, pp. 105–114.

References

Conclusions

[4] Nagarajan, V., Kim, H.-S., Wu, Y., and Gupta, R.
Dynamic information tracking on multicores.
In *INTERACT* (Feb 2008).

[5] Palmiero, C., Di Guglielmo, G., Lavagno, L., and Carloni, L. P.
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).

[6] Traber, A., Gautsch, M., and Davide, S. P.
RI5CY: User Manual Revision 1.7.
ETH Zurich, University of Bologna, 2017.