ACM CCS '24
Accepted

Libra

Dream of Secure Balanced Execution on High-End Processors? - Let's Make it Real!

Shonan Meeting – Microarchitectural attacks and defenses

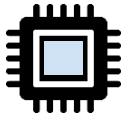
Hans Winderix, Marton Bognar, Lesly-Ann Daniel, Frank Piessens

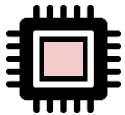
fwo

KU LEUVEN DistriNet

The control-flow leakage (CFL) problem

```
beq s1 a0 Target
    add a1 a2 a3
    j End
Target:
    add a2 a3 a4
End:
```



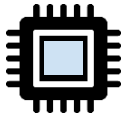


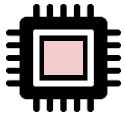
Executions produce *observations*

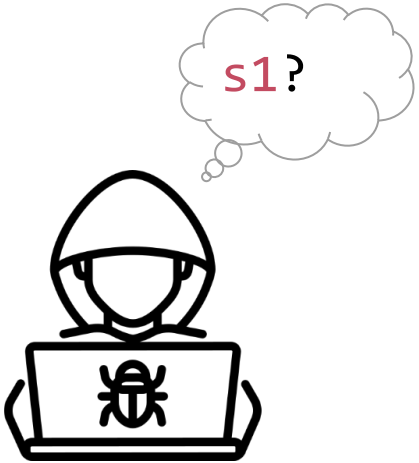
- End-to-end timing
- Microarchitectural resource usage
  - cache usage
  - port contention
  - etc.

The control-flow leakage (CFL) problem

```
beq s1 a0 Target
    add a1 a2 a3
    j End
Target:
    add a2 a3 a4
End:
```

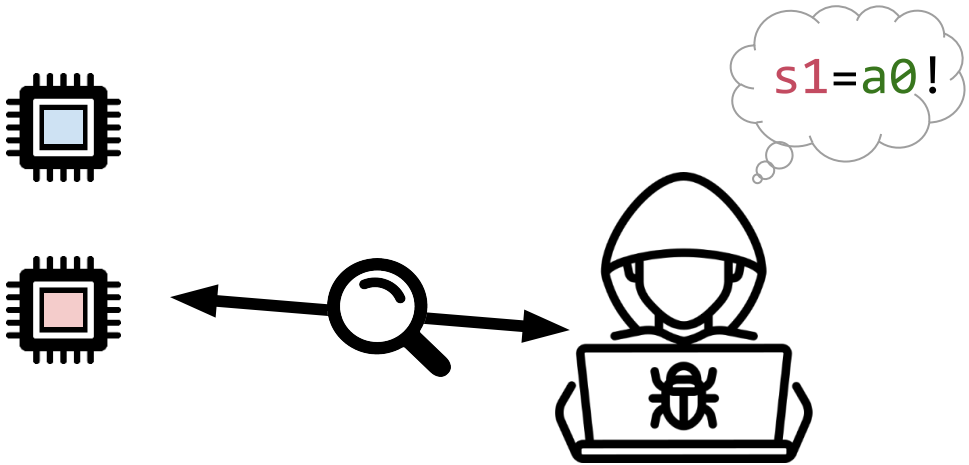






The control-flow leakage (CFL) problem

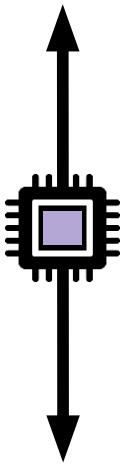
```
beq s1 a0 Target
    add a1 a2 a3
    j End
Target:
    add a2 a3 a4
End:
```



State of the art software countermeasures

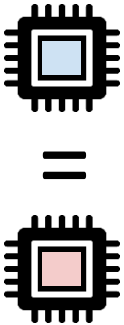
Linearization (Molnar [1])

```
sub t0 s1 a0
setqz t0 t0
addi t0 t0 -1 ;true mask
not t1 t0     ;false mask
and t2 a1 t0
add a1 a2 a3
and a1 a1 t1
or  a1 a1 t2
and t2 a2 t1
add a2 a3 a4
and a2 a3 t0
or  a2 a2 t2
```



Balancing

```
beq s1 a0 Target
    add a1 a2 a3
    j End
Target:
    add a2 a3 a4
    j End
End:
```

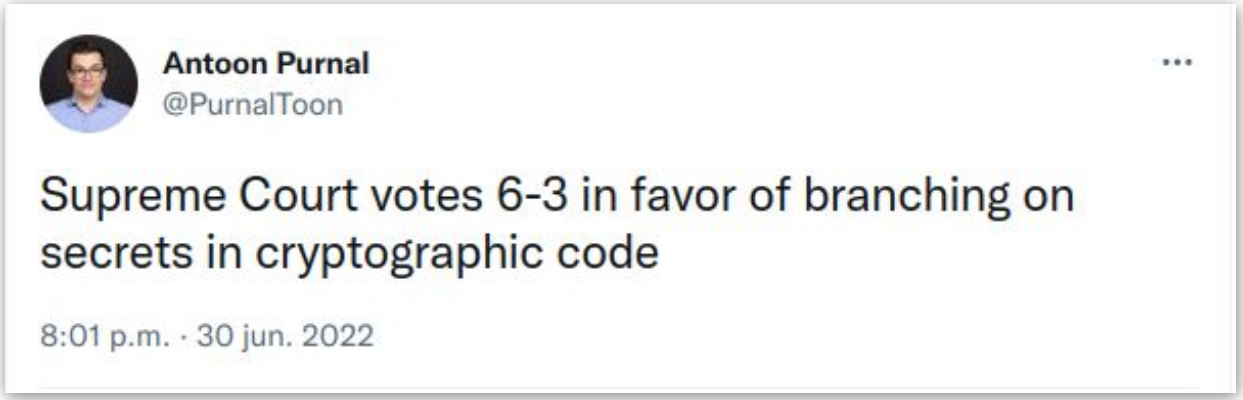


[1] Molnar et al., The program counter security model: Automatic detection and removal of control-flow side channel attacks (ICISC 2005)

Branch balancing, are you kidding me?

“What about branch predictors or instruction caches?”
– Any side-channel expert

“We all know it’s insecure on high-end processors!”
– Any reasonable cryptographer

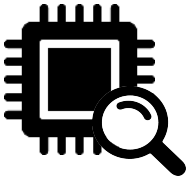


Branch balancing, are you kidding me?

“But actually why not?”

– Hopeful dreamer

Research questions

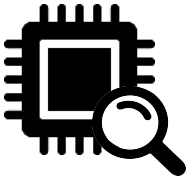


What **microarchitectural features** leak control-flow?

How to **securely balance branches** on high-end CPUs?

Can it improve **performance** over linearization?

Contributions



What **microarchitectural features** leak control-flow?

➔ **Characterization** of HW sources of **control-flow leakage**

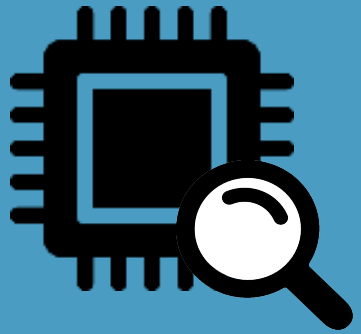
How to **securely balance branches** on high-end CPUs?

➔ **Libra**: Architectural support for balanced execution

Can it improve **performance** over linearization?

➔ HW **implementation** & evaluation (19.3% less overhead)

Characterization HW sources of control-flow leakage



Literature review

65 attack papers

29 optimizations

Balanceable leakage

Independent of pc

- instruction latency
- data cache
- data TLB
- loads/store buffer dep.
- data dependencies
- …

➔ can be handled in SW :-)

➔ but not in a principled way :-(

Unbalanceable leakage

Dependent of pc

- instruction cache
- instruction TLB
- instruction prefetcher
- branch predictors
- µ-op caches
- …

➔ cannot be handled in SW :-(

We need a new HW/SW co-design for balancing

Security-oriented HW/SW co-design

Full side-channel security

Principled: Leakage contract for balanced execution

Can be leveraged by SW to write secure code

Efficient: do not disable HW optimizations

Common case: keep all optimizations enabled

Secret-dependent region: keep as many optimizations as possible

Libra

SW handles **balanceable** leakage (principled)

HW support to address **unbalanceable** leakage

Libra Leakage Contracts

Augment ISA with **2-D leakage contract**

1. **Leakage classes**
   - same class = same observation (e.g. **add** x1 x2 ≈ **sub** x1 x2)
   - *dummy (no-op)* instruction for each class (e.g., **mv** x1 x1)

2. **Safe/unsafe instructions**
   - ***Safe*** *instr*: timing does not depend on operand – **add** x1 x2
   - ***Unsafe*** *instr*: timing depends on operand – **load** x1 (x2)

Libra Leakage Contracts

Used by **software** to balance secret-dependent regions

```
beq s1 a0 Target
    addi a1 a1 1
    load a2 (a3)
    j End
Target:

End:
```

Libra Leakage Contracts

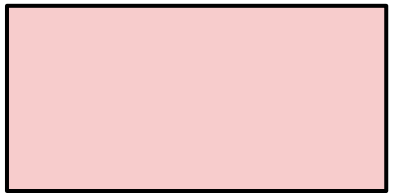
Used by **software** to balance secret-dependent regions

```
beq s1 a0 Target
    addi a1 a1 1
    load a2 (a3)
    j End
Target:

End:
```

```
addi a1 a1 1
load a2 (a3)
j End
```

~
~
~



Balance instruction by instruction

Libra Leakage Contracts

Used by **software** to balance secret-dependent regions

```
beq s1 a0 Target
    addi a1 a1 1
    load a2 (a3)
    j End
Target:

End:
```

`addi a1 a1 1`	~	`addi a1 a1 0`
`load a2 (a3)`	~	
`j End`	~	

Instructions from same class

Libra Leakage Contracts

Used by **software** to balance secret-dependent regions

```
beq s1 a0 Target
    addi a1 a1 1
    load a2 (a3)
    j End
Target:


End:
```

`addi a1 a1 1`	~	`addi a1 a1 0`
`load a2 (a3)`	~	`load x0 (a3)`
`j End`	~	

Balance operands of unsafe instr.

Libra Leakage Contracts

Used by **software** to balance secret-dependent regions

```
beq s1 a0 Target
    addi a1 a1 1
    load a2 (a3)
    j End
Target:
    addi a1 a1 0
    load x0 (a3)
    j End
End:
```

Software balanced w.r.t. weak observer

… But still insecure w.r.t. strong observer

Libra Folding Transformation

Key Idea: *interleave* instructions of secret-dependent regions

```
beq s1 a0 Target
    addi a1 a1 1
    load a2 (a3)
    j End
Target:
    addi a1 a1 0
    load x0 (a3)
    j End
End:
```

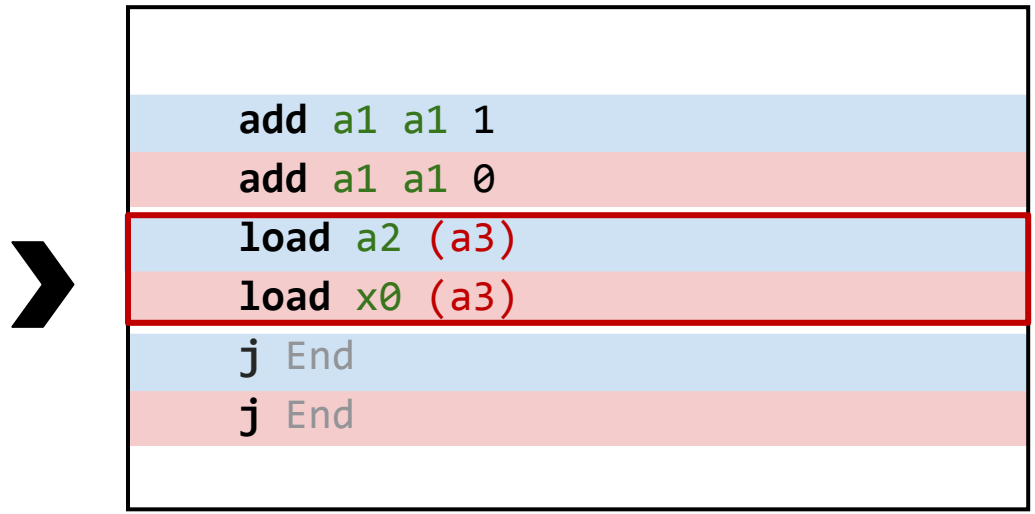
`addi a1 a1 1`	~	`addi a1 a1 0`
`load a2 (a3)`	~	`load x0 (a3)`
`j End`	~	`j End`

slice

Libra Folding Transformation

Key Idea: *interleave* instructions of secret-dependent regions

```
beq s1 a0 Target
    addi a1 a1 1
    load a2 (a3)
    j End
Target:
    addi a1 a1 0
    load x0 (a3)
    j End
End:
```



Fold memory layout of secret regions

➔ instr. in slice are contiguous

Libra Folding Transformation

Key Idea: *interleave* instructions of secret-dependent regions

```
beq s1 a0 Target
    addi a1 a1 1
    load a2 (a3)
    j End
Target:
    addi a1 a1 0
    load x0 (a3)
    j End
End:
```

```
lo.beq s1 a0 offT:1 offF:0 #bb:2
    add a1 a1 1                    ;pc+2
    add a1 a1 0                    ;pc+2
    load a2 (a3)                   ;pc+2
    load x0 (a3)                   ;pc+2
    lo.beq x0 offT:0 offF:0 #bb:1
    lo.beq x0 offT:0 offF:0 #bb:1
```

slice

Level-offset branch informs CPU:

➔ how to navigate folded region

➔ enter secret region so adapt behavior

Libra Folding Transformation

Key Idea: *interleave* instructions of secret-dependent regions

```
beq s1 a0 Target
    addi a1 a1 1
    load a2 (a3)
    j End
Target:
    addi a1 a1 0
    load x0 (a3)
    j End
End:
```

```
lo.beq s1 a0 offT:1 offF:0 #bb:2
    add a1 a1 1                 ;pc+2
    add a1 a1 0                 ;pc+2
    load a2 (a3)                ;pc+2
    load x0 (a3)                ;pc+2
    lo.beq x0 offT:0 offF:0 #bb:1
    lo.beq x0 offT:0 offF:0 #bb:1
```

slice

pc leaks at *slice granularity*

How to satisfy slice-granular leakage  ?

pc-dependent buffering

E.g. I-cache, I-prefetcher

Slice-granular fetch

pc-dependent mapping

E.g. pc-dep prefetcher

Slice-based mapping

Unbalanceable leakage

Instr-specific opt.

E.g. µ-op cache

In secret region, disable or do operation on whole slice

Inhibit dummy creation

E.g. *silent-store* opt.

In secret regions: disable opt. or blacklist stores

More in the paper

Advanced features

- Nested secret-dependent regions
- Function calls
  - **lo.call** + fold with dummy
  - Save/Restore Libra context

Formalization

- ISA semantics
- Folding transformation
  - Proof of correctness
  - Proof of security

Evaluation

Q1. Feasibility

Q2. Security

Q3. Performance

Q4. HW overhead

PoC: 32-bit RISC-V implem. on Proteus

Level-offset branch. Repurpose 2 prefix bits in RISC-V branches encoding

Sources of unbalanceable leakage.

- branch target predictor → disable in folded regions
- instruction caches
- instruction prefetcher

} ~~Libra-~~aware fetch unit

HW cost: No impact on CP

LUT: +11% (16.5k -> 18.4k), Reg: +9.5% (13.6k -> 14.9k)

Q1. Feasibility ✅

Q4. HW-Overhead ✅

Evaluation

Benchmark

11 programs from [1]:

- baseline
- balanced
- linearized
- folded

RTL-level noninterference testing

Run programs with ≠ secret & monitor:

- branch predictor state
- addresses in D/I-caches
- instruction prefetcher
- execution-unit occupancy

Q2. Security ✅

[1] H. Winderix, J. T. Mühlberg, and F. Piessens, "Compiler-assisted hardening of embedded software against interrupt latency side-channel attacks," in EuroS&P, 2021.

Performance

Binary size overhead

	Bal.	Linear.	Folded
Min	+0%	+8%	-6%
Max	+41%	+2%	+16%
Mean	+9%	+20%	**+3%**

Execution time overhead

	Bal.	Linear.	Folded
Min	+0%	+8%	-2%
Max	+282%	+225%	+227%
Mean	+42%	+56%	**+45%**

Overhead relative to linearization: **-19.3%**

Q3. Performance ✅

We need a new HW/SW co-design for balancing

Security-oriented HW/SW co-design

Full side-channel security

Principled: Leakage contract for balanced execution

Can be leveraged by SW to write secure code

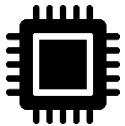
Efficient: do not disable HW optimizations

Common case: keep all optimizations enabled

Secret-dependent region: keep as many optimizations as possible

Summary

Balances secret regions
Applies **folding**



Slice-granular leakage

HW optimizations **enabled**

A new era for balancing?

Well, there are still challenges!

- Verif/synthesis for balancing contracts
- Balancing transformation
- Evaluation on larger benchmarks
- Feasibility with more complex optimizations?

Exploring HW-SW Co-Designs

Let’s take a dive

Proteus: An Extensible RISC-V Core for Hardware Extensions

(RISC-V Summit ’23)

Marton Bognar, Job Noorman, Frank Piessens

A modular textbook processor to study HW extensions

- **In/Out-of order** pipelines
- **Optimizations**: branch predictors, cache, prefetchers, …
- **Configurable**: #exec units, ROB size, …
- **Extensible**: plugin system
- **SpinalHDL** → verilog → FPGA / simulator
- **Validate**: HW fuzzing (in progress)

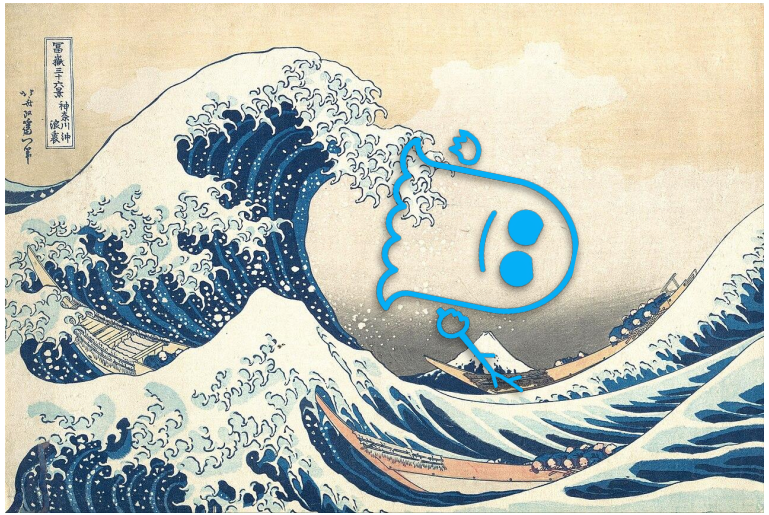
ProSpeCT: Provably Secure Speculation for the Constant-Time Policy

(USENIX'23)

Lesly-Ann Daniel, Marton Bognar, Job Noorman, Sébastien Bardin, Tamara Rezk, Frank Piessens

SW: annotate secrets
HW: no speculation on secrets

- **CT code secure against Spectre**
- Without sacrificing speculation
- Design proven secure w.r.t. contract
- Holds for many variants of Spectre



Architectural Mimicry: Innovative Instructions to Efficiently Address Control-Flow Leakage in Data-Oblivious Programs

(SP'24)

Hans Winderix, Marton Bognar, Job Noorman, Lesly-Ann Daniel, Frank Piessens

HW: Mimic execution (imitate µarch behavior)
SW: ISA extension to control it

- Principled control-flow hardening
- Support for linearization / balancing
- **Accelerate linearized code**

Challenges?

Ocean is
Large & Murky

Trade-offs to explore?

- Security guarantees
- HW/SW changes
- Performance
- Power

Evaluate & compare?

- **Performance** on large code
  - Need compiler support
  - Compilers are not *really* modular
  - Unified benchmark?
- **Hardware** costs / feasibility
  - How to evaluate robustly?
  - Generalization ≠ (μ)arch?

Libra: Architectural Support For Principled, Secure And Efficient Balanced Execution On High-End Processors

Hans Winderix
frank.piessens@kuleuven.be
DistriNet, KU Leuven
Leuven, Belgium

Marton Bognar
frank.piessens@kuleuven.be
DistriNet, KU Leuven
Leuven, Belgium

Lesly-Ann Daniel
frank.piessens@kuleuven.be
DistriNet, KU Leuven
Leuven, Belgium

Frank Piessens
frank.piessens@kuleuven.be
DistriNet, KU Leuven
Leuven, Belgium

Soon to appear :)

https://github.com/proteus-core

Credit

Image by djvstock on Freepik