# **Timesecbench**: a work in progress benchmark suite to assess microarchitectural timing leakages

Mathieu Escouteloup

Ronan Lashermes

Inria

# Contents





Inria

### Scenario 1

You are a hardware designer who wants to quickly look for timing leakage in your microarchitecture implementation.

### Scenario 2

You are a security evaluator who wants to verify a hardware design's robustness.

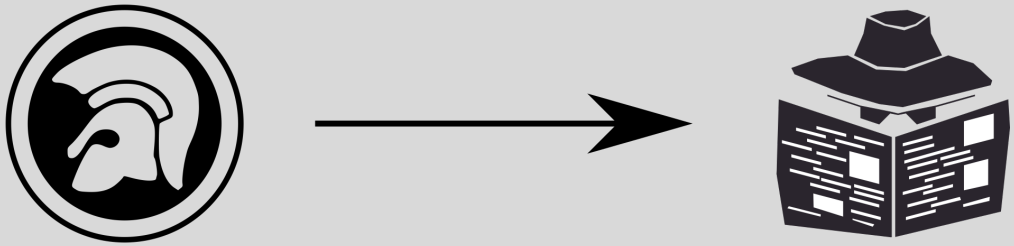
### Vulnerability $\neq$ Exploitability

We look for the possibility of a leakage from the hardware. It doesn't mean that this leakage can be realistically exploited.



Inria

## A communication channel between two security domains




## Threat model for microarchitectural leakage

The attacker controls both trojan and spy but can only encode information in the microarchitectural state.

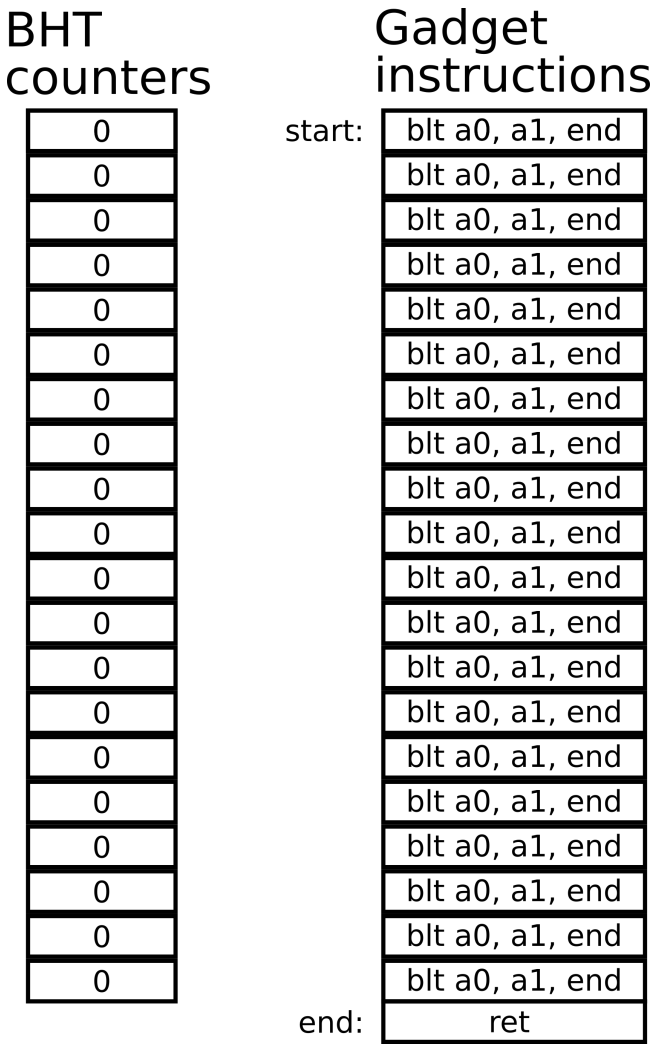
```
trojan(i);
architecture_clear(); //registers, memory, CSRs
domain_switch_to(spy_domain);
spy(o);
```

$$\exists \text{ side channel} \implies \exists \text{ covert channel.}$$



Inria

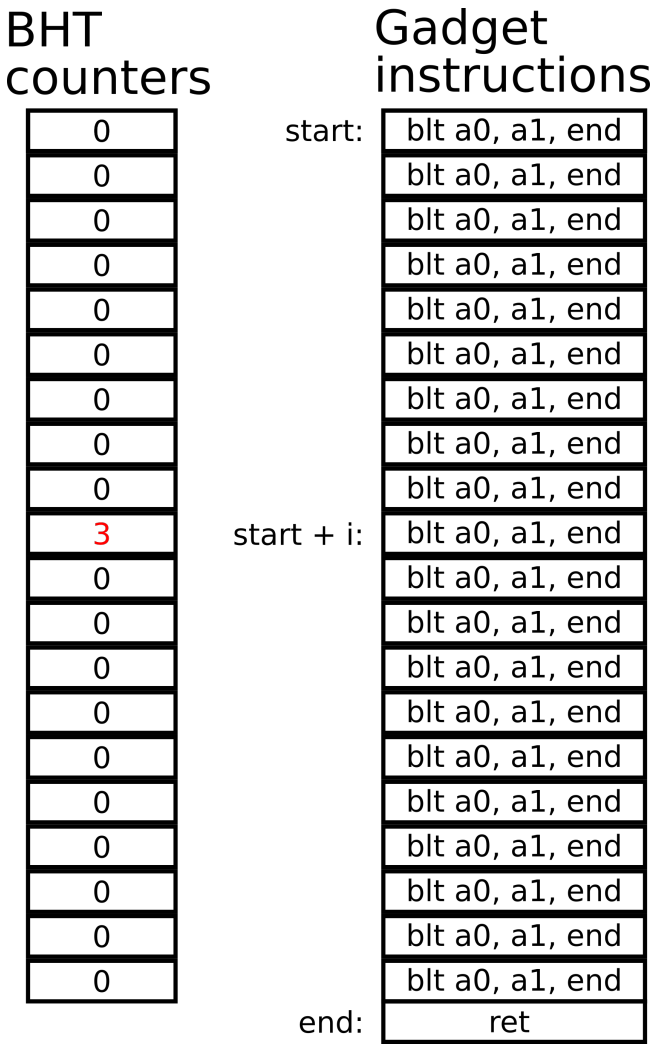
# Example: Branch History Table (BHT)




## Clearing the counters

Repeat `(*start)(2,1);`

# Example: Branch History Table (BHT)



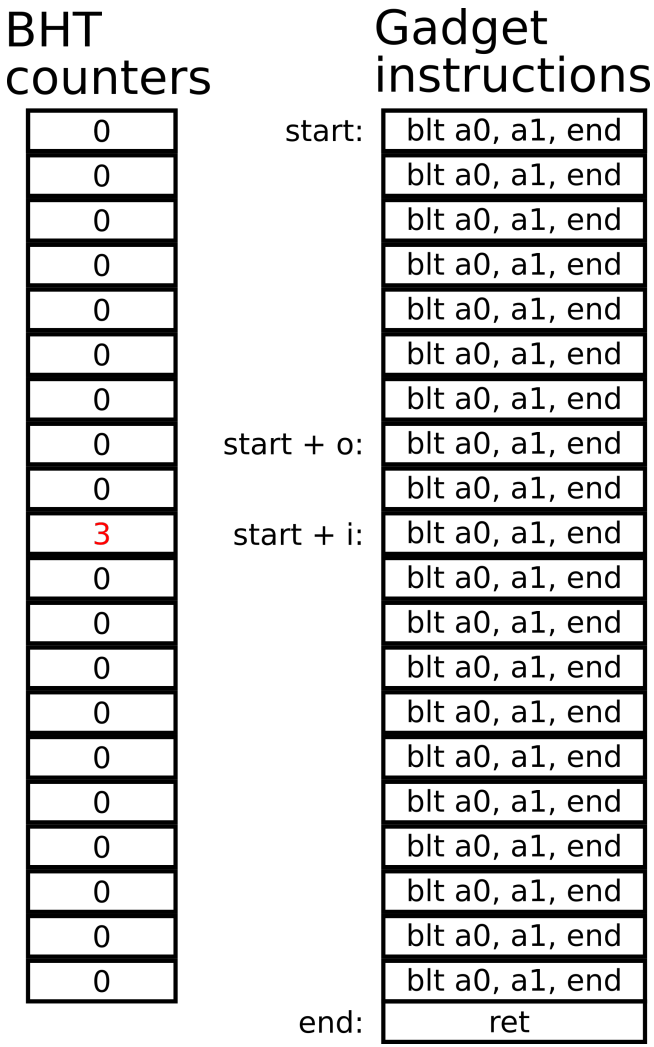

## Clearing the counters

```
Repeat (*start)(2,1);
```

## Encoding *i*, the trojan value

```
Repeat (*(start + i))(1,2);
```

# Example: Branch History Table (BHT)




## Clearing the counters

```
Repeat (*start)(2,1);
```

## Encoding *i*, the trojan value

```
Repeat (*(start + i))(1,2);
```

## Testing for *o*, the spy value

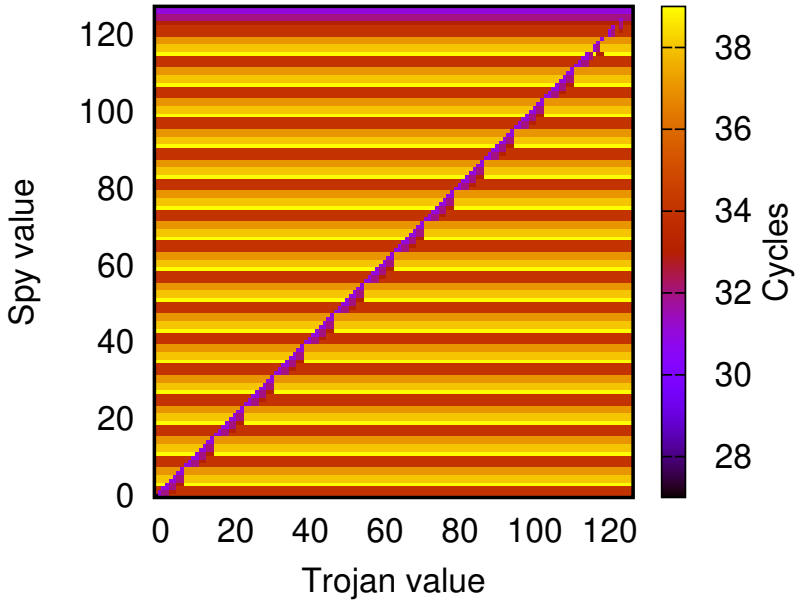
Execute:

```
 u32 a = read_time();
(*(start + o))(1,2);
return read_time() - a;
```

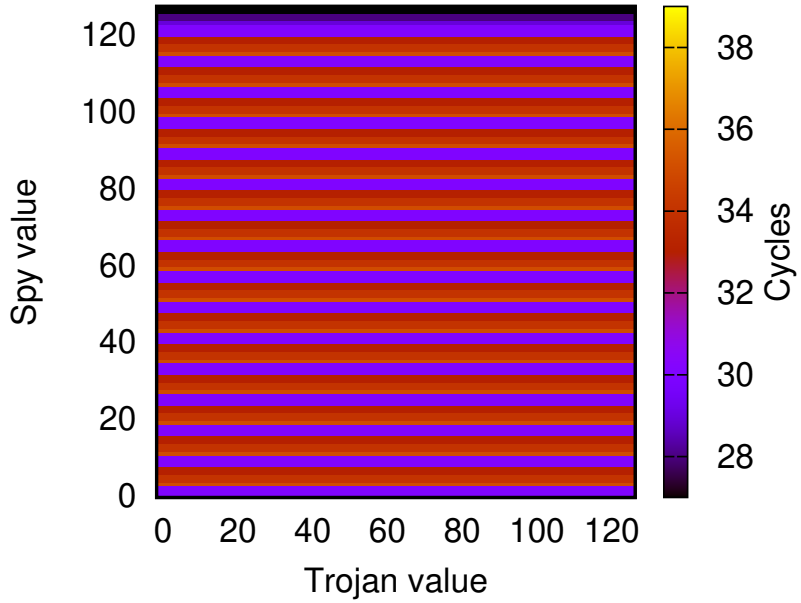


Inria

# Timing matrix




Vulnerable design




Secure design

## Visualizing leakage

Horizontal variation equals information leakage: we can detect if `i == o`.

## The difficulty

We want the benchmark to be highly portable: different chips, different ISAs, simulation / real execution, ...

## The solutions

- Follow the embench model: chip and board specific codes can be called from the benchmarks.
- Use the tools that C offers: `volatile`, `__attribute__ ((aligned (I_LINE_SIZE)))`, ...
- But only tested on our own RISC-V cores...



Inria

# BHT example

```
__attribute__ ((aligned (I_LINE_SIZE)))
uint32_t spy_poke_bht(uint32_t o) {
  sig_br* branch_address =
    (sig_br*) &gadget[o];
  //trigger address computation
  //before read_time()
  volatile uint32_t dummy =
    ((uint32_t*)branch_address);

  uint32_t start = read_time();
  branch_address(1, 2);
  uint32_t end = read_time();
  return (end - start);
}
```



Inria

## Is it usable yet ?

- 4 benchmarks: L1D, L1I, BTB, BHT.
- We need more: RAS, PHT, ...
- Port to other chips not tested.
- No multihart story yet (preliminary tests for cross-L1D and port contention).
- Documentation not complete.
- Dealing with non-determinism.
- Metrics...



Inria

## Quantification

Visualizing timing dependency is fun but not very scientific. We want a leakage metric: **bits of transmitted information per cycle (or per second)**.

## Channel matrix

For that we need the channel matrix: probability matrix that the spy reads $o$ when the trojan sends $i$.
We need

- A decision criteria: from the timing, what does the spy decide?
- Dealing with non-determinism.

## Computing channel capacity

The mutual information can demonstrate that there is no leakage. But useless as a metric. We want the **channel capacity**.

# Conclusion

- A benchmark suite to test easily common timing leakage sources in the microarchitecture.
- Misses non-determinism, multi-hart and metrics.

**Tell me about YOUR needs.**

`https://gitlab.inria.fr/rlasherm/timesecbench/`