Can we solve timing side-channels in the CPU?

Challenges and tradeoffs to design more secure cores

Ronan Lashermes

Inria

September 13, 2021

Contents

Inria

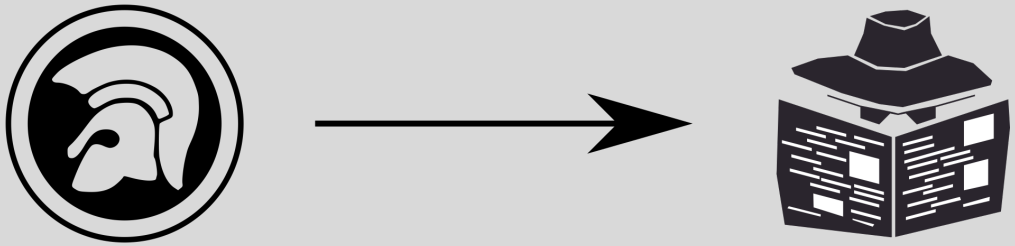
01

Motivation


```
tab[ (*s_addr) * 4096];
```

- $s \leftarrow$ load [s_addr]: reads the secret, should be forbidden.
- load [$tab + 4096 \cdot s$]: touches a cache line depending on s.
- the attacker needs to find the addresses of data in cache (*cache timing attacks*).

A communication channel between two security domains



Threat model for microarchitectural leakage

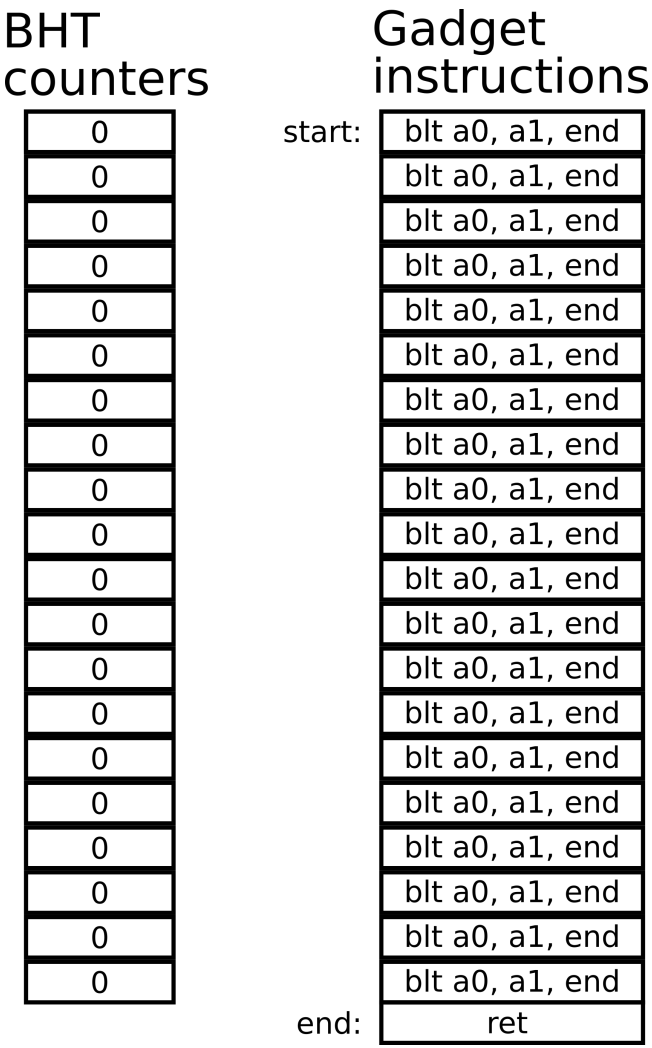
The attacker controls both trojan and spy but can only encode information in the microarchitectural state.

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

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

Inria

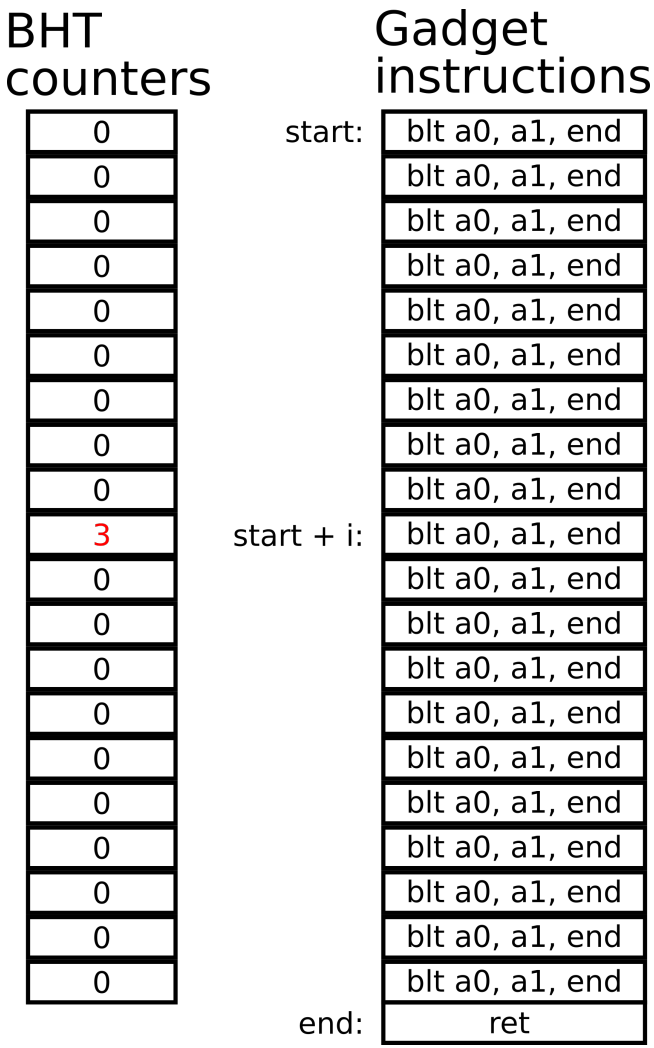
Example: Covert channel from BHT



Clearing the counters

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

Example: Covert channel from BHT



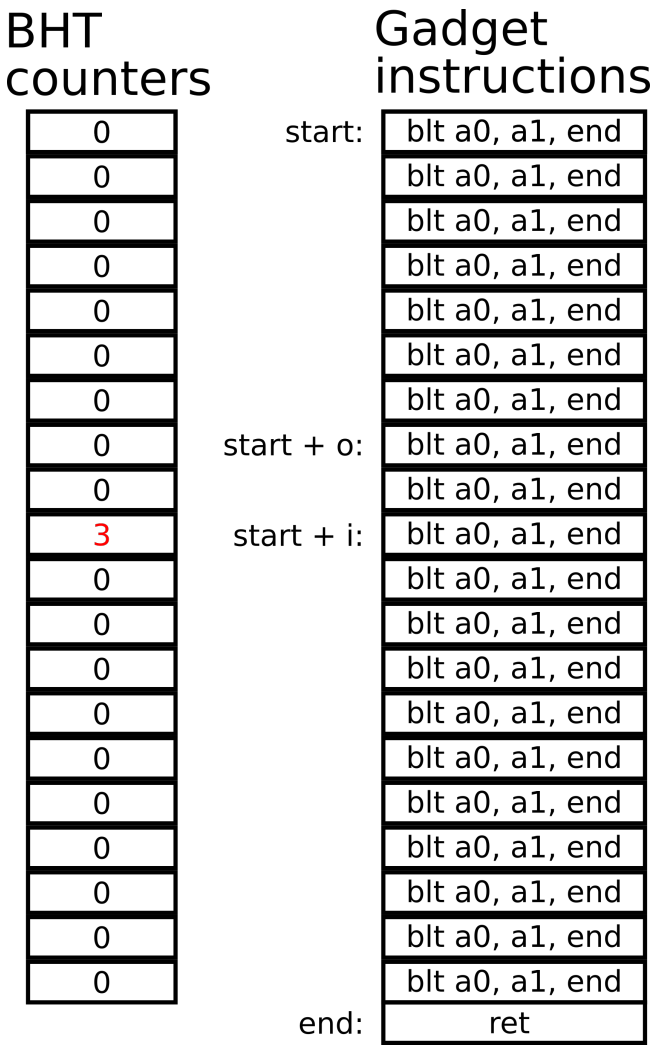
Clearing the counters

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

Encoding *i*, the trojan value

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

Example: Covert channel from 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

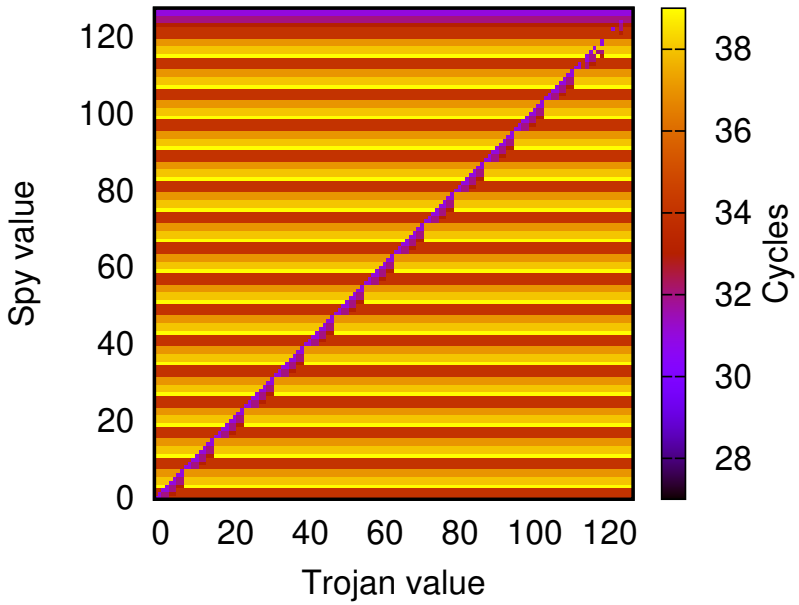


Figure: Vulnerable design

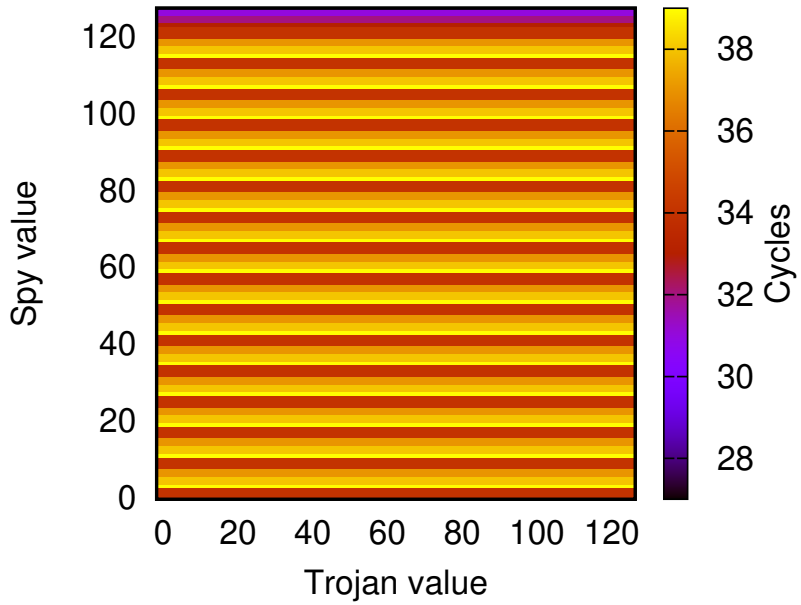


Figure: Secure design

Visualizing leakage

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

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

02

Thinking this issue through

This is my personnal point of view.

Shared resources

Any shared resource can be the support of a covert (or side) channel: any buffer, execution unit, FSM, bus, etc.

Ge et al., "No Security Without Time Protection: We Need a New Hardware-Software Contract"

Shared resources

Any shared resource can be the support of a covert (or side) channel: any buffer, execution unit, FSM, bus, etc.

Usually ok

In most case, this information leakage is desirable.

Ge et al., "No Security Without Time Protection: We Need a New Hardware-Software Contract"

Shared resources

Any shared resource can be the support of a covert (or side) channel: any buffer, execution unit, FSM, bus, etc.

Usually ok

In most case, this information leakage is desirable.

But in some cases not

The hardware cannot know when it must prevent leakage.

Ge et al., "No Security Without Time Protection: We Need a New Hardware-Software Contract"

Security in today's hardware

- Kernel/user (privileges): access control to core configuration.
- Processes (memory space): access control to memory addresses.

What about a unique server dealing simultaneously with different users ? A cloud machine with several VMs running ?

Security in today's hardware

- Kernel/user (privileges): access control to core configuration.
- Processes (memory space): access control to memory addresses.

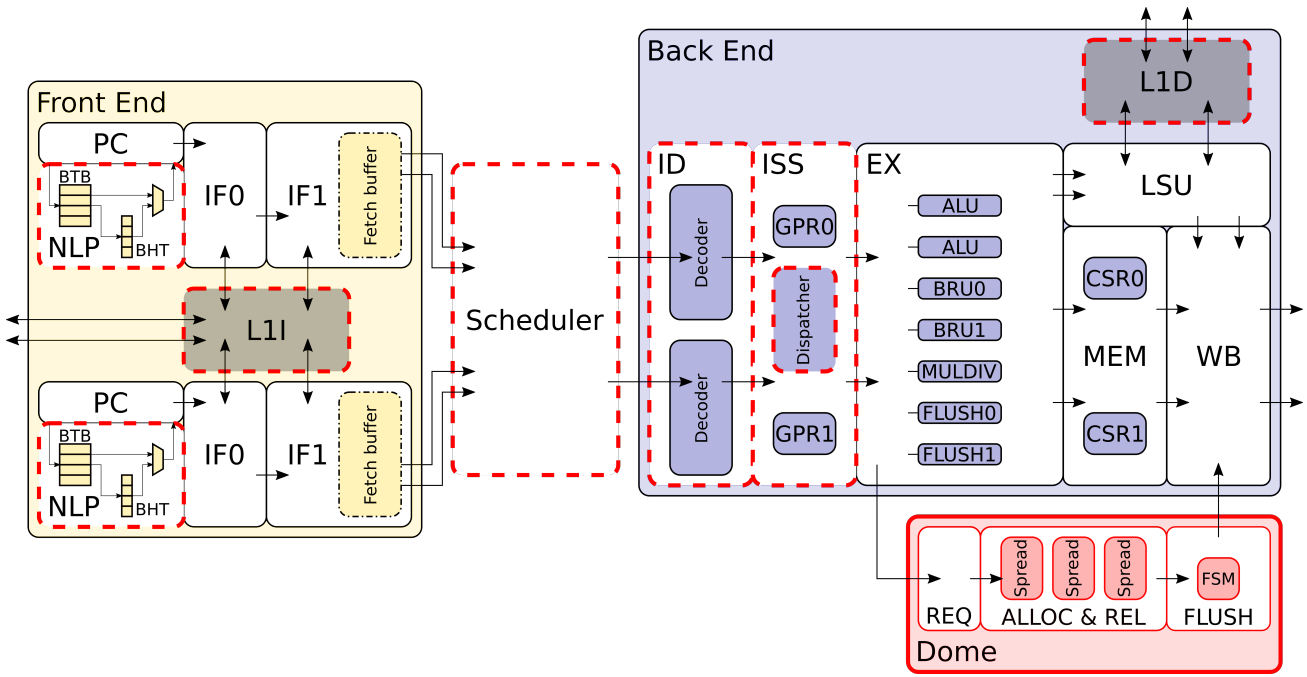
What about a unique server dealing simultaneously with different users ? A cloud machine with several VMs running ?

Security domains (SD)

- SD boundaries are defined by the app logic $\rightarrow$ software.
- SD must be enforced in the microarchitecture $\rightarrow$ hardware.
- Boundaries must be communicated to the hardware $\rightarrow$ ISA.

Kinds of sharing

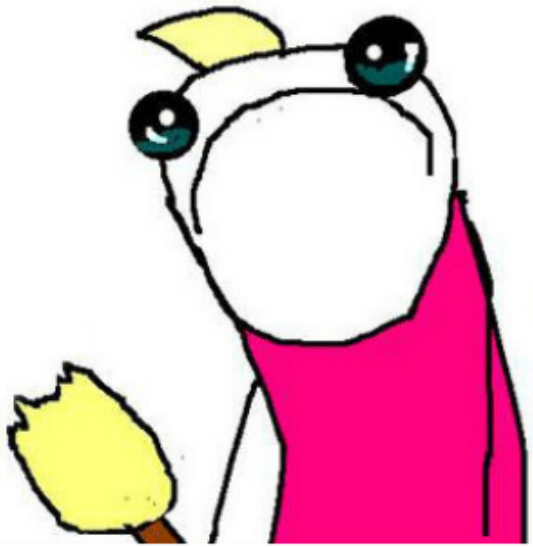
- Temporal sharing: SDs use the resource alternatively in time.
- Spatial sharing: SDs can use the same resource simultaneously.



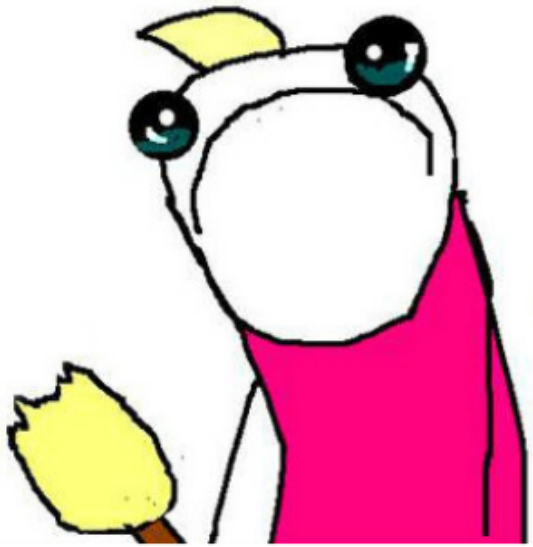
Salers core, secure multithreading, by M. Escouteloup, Inria.


FLUSH ALL THE THINGS!

Flush all the things?



Flush all the things?



Solutions

- Partition resources.
- Fine-grained multithreading (1 out of N time slots).
- Lock resources.

03

Proposing ISA modifications

What is the ISA role ?

- Interface to allow software to precisely control the machine.
- Interface to abstract the hardware, for the software.

You can only pick one !

What is the ISA role ?

- Interface to allow software to precisely control the machine.
- Interface to abstract the hardware, for the software.

You can only pick one !

Consequences

- Add instructions to control the microarchitectural structures.

OR

- Add instructions to define and handle security domains.

Fences

A *stateless* definition of security domains: explicitly delimit the boundaries only.

Contexts

Stateful security domains: each SD has an ID, and switching to a new SD requires to define the new ID.

Fence.t

Instructions

- `fence.t` : flush all necessary microstructures.
- `fence.t rs1` : flush all microstructures designated by `rs1`.

`rs1` can be:

- A bitmap addressing microstructures.
- A value designating an interface (process to kernel, process to process, . . .).
- A value designating a risk (low, medium, high).

No spatial sharing

Needs another mechanism for multicores systems.

"Microarchitectural Timing Channels and their Prevention on an Open-Source 64-bit RISC-V Core.", by Nils Wistoff, Moritz Schneider, Frank K. Gürkaynak, Luca Benini, Gernot Heiser at *DATE 2021*.

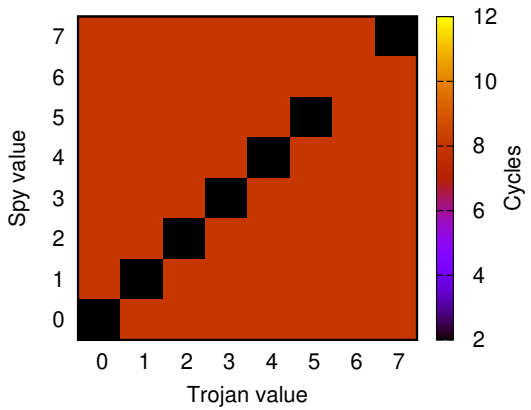
Inria

Instructions

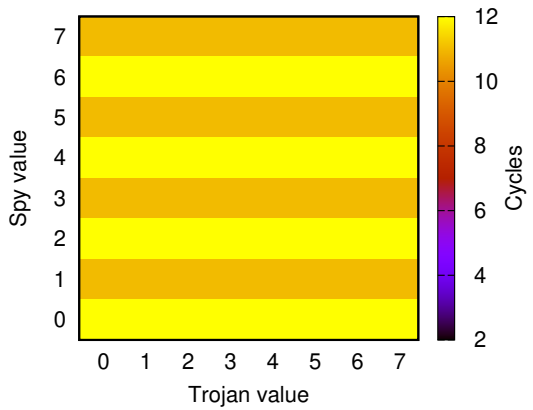
`dome.switch rs1`: switch to a new security domain with ID=`rs1`.

Cross L1D

Unprotected



Protected



Who is in control ?

Someone has to choose the security domains IDs. Can process ID and domain ID be fused ?

"Under the dome: preventing hardware timing information leakage", by Mathieu Escouteloup, Ronan Lashermes, Jacques Fournier, Jean-Louis Lanet at *CARDIS 2021*.

Confidential registers

Declare `x28-x31` registers as confidential. They are no longer authorized to leak data:

- Cannot be used to branch.
- Cannot be used for `load` or `store` addresses.
- Only constant-time instructions wrt these registers (replace Zkt).
- Additional hardware hardening at designer's discretion (e.g. masking).

Confidential registers

Declare `x28-x31` registers as confidential. They are no longer authorized to leak data:

- Cannot be used to branch.
- Cannot be used for `load` or `store` addresses.
- Only constant-time instructions wrt these registers (replace Zkt).
- Additional hardware hardening at designer's discretion (e.g. masking).

Inline memory encryption

Secure domain IDs could be adapted for cryptographic usage, allowing per domain encryption, in memory, upon `load` and `store`.

04

Conclusion

Inria

Avoiding bad design

Learn from the cryptography community

- Patching issues as they appear is untenable.
- Complexity is a security topic in itself: too many security decisions increases the attack surface.
- Clear and simple interfaces: security cannot be achieved by arcane techniques that a handful of people know.

Thank you !

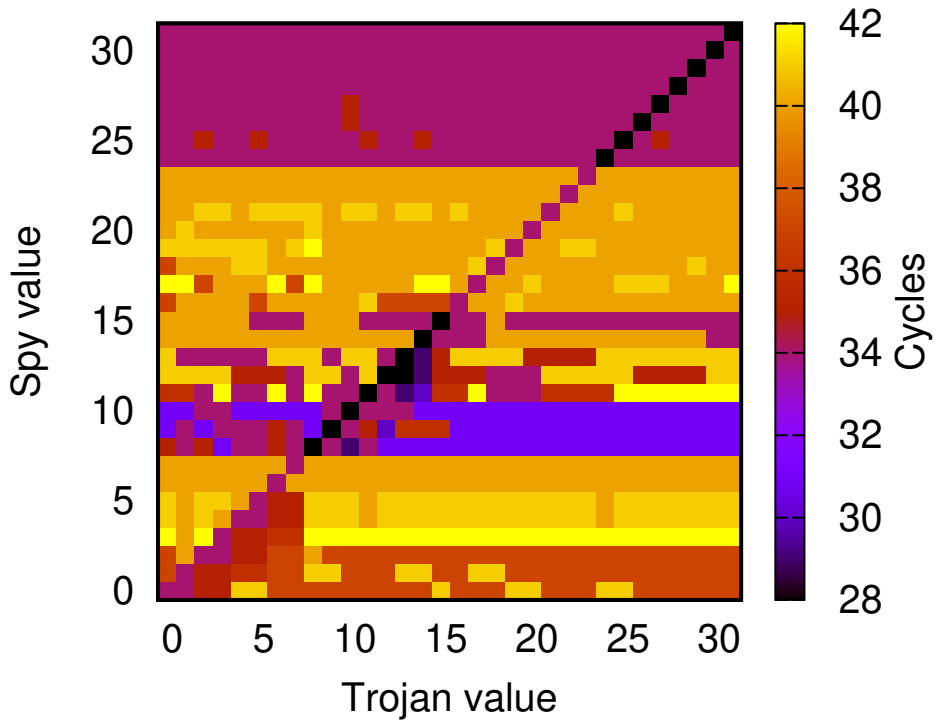


Figure: Targeting the BTB

I would love to hear your opinion !

05

Backup

```
if (x < tab1_size) {
   y = tab2[tab1[x]* 4096];
}
```

- $s \leftarrow$ load $[x + tab1]$: reads secret, should be forbidden for some x.
- load $[tab2 + 4096 \cdot s]$: touches a cache line depending on s.
- the attacker needs to find the addresses of data in cache (*cache timing attacks*).