Propositions to counter covert channels: explicit security domains

July 27th, 2022

General considerations

Document purpose

This document is a support for discussions on potential countermeasures against covert channels. Nothing is definitive, **everything is open to discussions. Speak up !**

Encumbered IP

We will not discuss competitors' solutions.

Strategy

Independance to microarchitecture specifics

- Instructions do not reference specific microarchitectural mechanisms.
- Make hardware design great again ! Avoid to unnecessarily constrain possible hardware.
- Orthogonal to other security mechanisms.

Security semantics

Add security annotations as an extension, where they are the most relevant. To be used to counter covert channels.

Explicit security domains I

Definition

A security domain delimits an execution context where the same security policy applies.

Security domain vs security context

Security context $\rightarrow$ all the execution context that impact security:

- Privilege levels.
- Address space identifier (ASID).
- **Security domain**.
- Virtualization elements.
- Memory protection elements.
- ...

Explicit security domains II

Consequences

At any time, the security context is tied to one security domain. The security policy defines when microarchitectural covert channels are forbidden.

Security policy

Will be the subject of much debate; need inputs from other SIGs and TGs.

Examples: covert channels are forbidden across privilege levels, covert channels are forbidden across security domain boundaries.

Explicit security domains III

Semantics

There are several possible ways to delimit the domains.

- `secdom.switch`: delimiting boundaries only (aka `fence.t`[2]).
- `secdom.switch rs1`: specifying the identifier of the new domain (aka `dome`[1]).
- `secdom.push`, `secdom.pop`, `secdom.switch`: stack semantics to deal with common use cases (interruptions, syscalls, driver calls, ...). A hint that we will return soon to the initial domain to mitigate costs.
- Tagged domains: `secdom.save rd`, `secdom.restore rs1` in order to save the current security domain and the ability to restore it, the most versatile option.

The semantics should not constrain unnecessarily the hardware implementation.

Microarchitectural consequences

According to the literature, there are several possibilities to avoid covert channels in the microarchitecture with different trade-offs. But beware, any persistent state can be used to build a covert channel !

- **Flush**: erase microarchitectural state at security domains boundaries.
- **Split**: partition state between security domains.
- **Lock**: dedicate a state to one security domain only, in special cases only.

The security domain semantics must allow all these behaviours, with different hardware implementation depending on the chip market (server, embedded, ...).

References I

[1] Mathieu Escouteloup, Ronan Lashermes, Jacques Fournier, and Jean-Louis Lanet.
Under the dome: preventing hardware timing information leakage.
In *CARDIS 2021-20th Smart Card Research and Advanced Application Conference*, pages 1–20, 2021.

[2] Nils Wistoff, Moritz Schneider, Frank K. Gürkaynak, Luca Benini, and Gernot Heiser.
Microarchitectural timing channels and their prevention on an open-source 64-bit RISC-V core.
In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*, pages 627–632. IEEE, 2021.