Propositions to counter covert channels and speculation vulnerabilities

June 13th, 2022

General considerations

Document purpose

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

This is a SIG

We must agree on the contours of an extension (or extensions), that's all. The actual work will take place in a TG.

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.

Section 2

Threats

Speculation vulnerabilities

Alternative reality

A speculative execution is an execution for an alternative reality inside the same security context.

The 4 horsemen of the speculation attack

1. Microarchitectural state injection : the attacker prepares the state before speculation.
2. Speculation gadget : gadget triggering speculative execution.
3. Window controlling : the attacker wants to increase the speculation window to reach the disclosure gadget.
4. Disclosure gadget : gadget used to exfiltrate data, with a covert channel or to an “outside” observer.

Section 3

Propositions

Strategy

Security semantics

Add security annotations as an extension, where they are the most relevant. To be used to counter covert channels and speculation attacks, but which may serve other purposes in the future.

The three components of security annotations

1. Explicit security domains.
2. Security properties for memory (coarse), $\rightarrow$ Memory Safety SIG.
3. Adversarily-controlled registers or another "manual security" mechanism.

Details in following slides.

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.
- A memory-page level element ?

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, requiring flushes and partitioning.

Security policy

Will be the subject of much debates ; need inputs from other SIGs and TGs.
One example : for each privilege level, define the speculation security mode : automatic or manual (cf adversary-controlled registers below).

Explicit security domains III

Semantics

There are several possible ways to delimit the domains.

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

Automatic vs manual speculation security

Automatic : the hardware's responsability

In this mode, the hardware is responsible for preventing speculation attacks, the software can be kept as is.

Manual : the software's responsability

New instruction hints must be inserted in the code to influence the speculation behavior.

Why two modes ?

The security "correct" answer is to have a secure hardware (automatic mode). To get better performances, we may rely on the application logic to loosen the security constraints where they do not matter. **The manual security mode will get misused, and will be the source of security vulnerabilities : it is very hard to get right.**

Against speculation barriers

Simplistic definition

An instruction that "stop" speculative execution.

Arguments against :

- Very hard to have a precise definition, totally independent from a microarchitecture.
- This instruction is dealing with a microarchitectural implementation feature, not a security property.
- We cannot expect portability of code with speculation barriers.

Adversary-controlled registers I

We may need to

1. State injection : prevent modifications in uarch state.
2. Speculation gadget : prevent speculation from happening.
3. Window controlling : stop ongoing speculation (speculation barrier).
4. Disclosure gadget : prevent usage of uarch state.

reg.ac hint

A `reg.ac rs1` instruction indicates an adversary-controlled register, and hardware must act accordingly. A tentative security-minded generalization of the speculation barrier. `reg.ac` is used to communicate a security property of the application to the hardware.

Adversary-controlled registers II

Adversary-controlled definition

An attacker can bias the register value, not necessarily set it directly.

Bound checks bypass 1

```
# a1 controlled by user,  a0 is array max index
# may prevent speculation on branch
    reg.ac a1
    blt  a0, a1, end
    ...
```

Adversary-controlled registers III

Bound checks bypass 2

```
# a1 controlled by user,  a0 is array max index
    blt  a0, a1, end
    ... # computing address
# t0 contains address to load

# reg.ac may prevent speculative load
    reg.ac t0
    lw   a0, 0(t0)
```

Retpoline example

Retpoline

```
reg.ac a0
jr a0
```

`a0` may not leave traces in the uarch, `jr` may not trigger speculation.

Only the hardware knows how to react in this case, depending on design.
For examples :

- Prevent speculation on `jr`.
- Allow speculation, but only with dedicated uarch structures.
- Prevent speculation plus `jr` execution does not update the uarch structures for jump predictions.
- No-op (simple in-order core).
- ...

Behaviour may be impacted by security policy.

A tale of two extensions

Security domain extension

Major debatable points :

- Semantics.
- Security policies.

Adversarily-controlled registers extension

Need far more feedback or alternatives.

Tests and validation

Waiting for your feedback

We need all the feedback we can get :

- Missing features ?
- Useless mechanisms ?
- Opinion on proposed semantics ?
- Remarks on performance impact ?
- ...

Speak up !