Propositions to counter covert channels and speculation vulnerabilities

April 5th, 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

Covert channels

Definition

An attacker build a communication channel between two entities in two different security domains. Emitter is Trojan, receiver is Spy.

Microarchitectural consequences

Any microarchitectural state can be used as support for a covert channel. The solution is to flush or partitionnate the microarchitecture at security domain boundaries [10, 2, 4].

Limits

- Hardware must be able to identify security domains.
- Costly for performances $\rightarrow$ necessarily coarse-grained.
- Cannot prevent leakage inside same security domains or with respect to "outside" observer.

Speculation vulnerabilities

Alternative reality

A speculative execution is an execution for an alternative reality inside the same security domain. Speculation must be prevented in critical cases.

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.

Speculation vulnerabilities, continued

Exploits

There are two main known exploits, depending on the speculation gadget.

- Branch Target Injection [1, 7, 8] : the attacker diverts the control flow to her desired location.
- Bound checks bypass [7] : the attacker inverts the direction of a branch.

These exploits are mostly used to read an arbitrary memory location.

We should not focus on mitigating know exploits, but on the underlying causes instead.

Protection should not target particular microarchitectural structures

We do not know what can be used to trigger speculation in future hardware. **Any** uarch state can be the support for a covert channel, and used by a disclosure gadget.

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).
3. Breach of trust in execution state.

Details in following slides.

Explicit security domains I

Principle

At any time, the execution context is tied to a security domain. Between security domains, microarchitectural covert channels are forbidden, requiring flushes and partitioning.

Explicit security domains II

Semantics

There are several possible ways to delimit the domains.

- `secdom.switch` : delimiting boundaries only (aka `fence.t` from [10]).
- `secdom.switch rs1` : specifying the identifier of the new domain (aka `dome` from [4]).
- `secdom.push`, `secdom.pop`, `secdom.switch` : stack semantics to deal with common use cases (interruptions, syscalls, driver calls, ...). Hint that we will return to the initial domain to mitigate costs.
- others ?

Explicit security domains III

Security policy

We may use an immediate value in the instruction encoding to specify a security policy. Extendable to vendor custom use cases. E.g. 0 => unsecure, 1 => hardened, 2 => manual (see “breach of trust”), . . .

Security properties for memory I

Why more security properties for memory ?

Branch target injection and Bound checks bypasses are used to read arbitrarily in memory. There is no need to guard a particular memory access rather than another one, since they are all potential vulnerabilities.
But we may annotate that some security regions are more security critical than others.

At address space granularity

The lowest `ASID` bit becomes a `confidential` bit. If `ASID` is odd, the hardware know to apply extra precautions.

Security properties for memory II

At memory page granularity

Add a `confidential` bit to memory page flags, using the reserved bits for Sv39, Sv48 and Sv57. The hardware should take extra precautions to deal with data coming from `confidential` memory pages. E.g. preventing speculative `load`.

There are lots of other mechanisms to protect memory

Is it the correct place to introduce this kind of annotation ? Probably a joint action with another SIG/TG (e.g. memory safety ?).

Breach of trust in execution state I

No need for speculation barriers...

It is possible to prevent dangerous speculative execution at the hardware level only, by detecting dangerous patterns [3] or by using dedicated microarchitectural states for speculation [5].

...but for various reasons, one may want to opt out of these automatic security mechanisms

Most probable reasons : ensuring compatibility with older IPs, prohibitive performance costs of secure execution.

Manually placing security related instructions is bad !

It can only be efficient with perfect knowledge of the hardware. This is a loaded gun given to developers [6].

Breach of trust in execution state II

We may need to

1. State injection : prevent modifications in uarch state.
2. Speculation gadget : prevent speculation from happening [9].
3. Window controlling : stop ongoing speculation (speculation barrier).
4. Disclosure gadget : prevent usage of uarch state.

`notrust` hint

An instruction hint to annotate a breach of trust at some point in a program. Precise definition needs to be properly established. It's a generalization of the speculation barrier.

Breach of trust in execution state III

A tentative definition

A `notrust` instruction indicates that hardware cannot trust the current execution state, including registers, and must act accordingly.

Variant with explicit registers

A `notrust register name` instruction indicates that hardware cannot trust the current execution state, including specified registers, and must act accordingly *(leaving encoding aside for now)*.

Breach of trust in execution state IV

Bound checks bypass 1

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

Breach of trust in execution state V

Bound checks bypass 2

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

# notrust may prevent speculative load,
# or act as a speculation barrier
    notrust t0
    lw   a0, 0(t0)
```

Retpoline example

Retpoline

```
notrust 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.

Hardware consequences

Now hardware designers can rely on the following information to create more trustful hardware :

- The privilege levels.
- The address space identifier `ASID`

- ...and its `confidential` bit.
- The security domain and the associated security policy.
- The memory page `confidential` flag.
- The presence of `notrust` instructions.

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 !

References I

[1] Enrico Barberis, Pietro Frigo, Marius Muench, Herbert Bos, and Cristiano Giuffrida.
Branch History Injection : On the Effectiveness of Hardware Mitigations Against Cross-Privilege Spectre-v2 Attacks.
In *USENIX Security*, August 2022.
Intel Bounty Reward.

[2] Thomas Bourgeat, Ilia A. Lebedev, Andrew Wright, Sizhuo Zhang, Arvind, and Srinivas Devadas.
MI6 : secure enclaves in a speculative out-of-order processor.
In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12-16, 2019*, pages 42–56. ACM, 2019.

References II

[3] Rutvik Choudhary, Jiyong Yu, Christopher Fletcher, and Adam Morrison.
Speculative privacy tracking (spt) : Leaking information from speculative execution without compromising privacy.
In *MICRO-54 : 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 607–622, 2021.

[4] 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.

References III

[5] Abraham Gonzalez, Ben Korpan, Jerry Zhao, Ed Younis, and K Asanovic.
Replicating and mitigating spectre attacks on an open source risc-v microarchitecture.
In *Third Workshop on Computer Architecture Research with RISC-V (CARRV 2019)*, 2019.

[6] Brian Johannesmeyer, Jakob Koschel, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida.
Kasper : Scanning for Generalized Transient Execution Gadgets in the Linux Kernel.
In *NDSS*, April 2022.

References IV

[7] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom.
Spectre attacks : Exploiting speculative execution.
In *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*, pages 1–19. IEEE, 2019.

[8] Alyssa Milburn, Ke Sun, and Henrique Kawakami.
You cannot always win the race : Analyzing the lfence/jmp mitigation for branch target injection.
arXiv preprint arXiv :2203.04277, 2022.

[9] Allison Randal.
Ghosting the spectre : fine-grained control over speculative execution.
2021.

References V

[10] 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.