
Contribution ID: **282** Type: **not specified**

# Address Space Isolation

*Friday, 20 September 2024 16:10 (20 minutes)*

Address Space Isolation (ASI) is a technique to mitigate broad classes of CPU vulnerabilities.

ASI logically separates memory into “sensitive”and “nonsensitive”, the former is memory that may contain secrets and the latter is memory that, though it might be owned by a privileged component, we don’t actually care about leaking. The core aim is to execute comprehensive mitigations for protecting sensitive memory, while avoiding the cost of protected nonsensitive memory. This is implemented by creating another “restricted”address space for the kernel, in which sensitive data is not mapped.

At LSF/MM/BPF this year I presented a conceptual overview and discussed strategy and some implementation details of interest to the mm community. [LWN]

I’m now in the process of preparing an updated RFC, the code is available and I plan to post it on LKML in the coming days. I’d like to spend this MC slot discussing feedback and questions that I expect the RFC to provoke. Some examples include:

- What should ASI’s default sensitivity be? Default-nonsensitive (known as an “denylist” model) provides a clear and pragmatic path to enablement, but doesn’t offer a highly principled mitigation without additional work. Default-sensitive (”allowlist”) lets us quickly high confidence in totally eliminating whole exploit classes, but presents a difficult road to actually running it in production.

Discussion with the mm community didn’t produce any strong objections either way. I’m currently hoping to start with a denylist, on the basis of “launch and iterate”. Do x86 folks support this strategy?

- Configuration of existing mitigations is organised on a per-vulnerability basis. ASI does not target any specific vulnerability. It also presents a new type of “policy” question to kernel users since it is not strictly equivalent to any other set of mitigations, as it works by deliberately dropping protection for certain data. How should ASI be enabled and how should this interact with existing defaults?
- How should ASI interact with KPTI? I think it’s always going to be a defensible security posture to enable ASI *and* KPTI at the same time (although I don’t imagine Google will ever do this). But do we want to support this complexity? Should ASI eventually replace KPTI? Assuming the answer is no, how entangled should the implementations be for these two features?
- Opinions of the new “critical section” concept that ASI introduces in order to deal with interrupts occuring in the guest/userspace return path.
- Discussion of the fact that ASI can make CR3 unstable even when preemption is disabled.

Other interesting topics that are less specifically x86-relevant include:

- The current RFC adds a page flag. How do we avoid doing this?
- Tricks to avoid unnecessary TLB flushes.

**Primary author:** JACKMAN, Brendan (Google)

**Presenter:** JACKMAN, Brendan (Google)

**Session Classification:** x86 MC

**Track Classification:** x86 Microconference