DOI:10.1145/3399740

To view the accompanying paper, visit doi.acm.org/10.1145/3399742

Technical Perspective
Why 'Correct' Computers Can Leak Your Information

By Mark D. Hill

INFORMATION SECURITY IS important, as much of life's private information is now stored on shared computers accessible from anywhere in the world. Many attacks begin by exploiting flaws in a system's implementation (bugs) or specification. Most exploited flaws today are in software, as software presents a large attack surface. While much rarer, hardware flaws can cause even correct software to leak information and fixing can even require new hardware.

As the complexity of modern systems has grown, we have become dependent on abstraction to manage it, and yet this gives rise to subtle classes of flaws when the assumptions that underpin these abstractions are violated. Abstractions in the logical systems can be perfect: once matrix properties are proven, they apply to all arrays of numbers. Abstractions of the physical world are approximations or models. For example, while light is neither a particle nor a wave, both are useful models.

A useful abstraction in computer science is the *instruction set architecture* (ISA) that separates software and hardware. Today, only a few commercially successful ISAs separate many millions of lines of software from scores of hardware implementations. Moreover, since IBM System/360 in 1964, these ISAs are specified as the *timing-independent functional behavior of* "*instructions*" that are somewhat primitive (for example, branches, loads, and adds) and where each processing core logically executes instructions sequentially.

In 2018, Spectre—detailed in the paper that follows—demonstrated that a computer the CS community believed to be correct—its implementation follows its ISA—could rapidly leak information to a malicious adversary. The public revelation began with three Spectre variants—including Meltdown—in January,[2,3,4] expanded to a dozen by October,[1] and has continued to grow since then. Spectre is possible because the ISA—like any abstraction of the physical world—is imperfect. In particular, the *timing-independent* ISA is implemented with a supposedly hidden microarchitecture that is *all about timing*: its purpose is to make a computer as fast as possible within cost constraints.

Spectre is possible because the ISA—like any abstraction of the physical world—is imperfect.

Spectre shows that current ISAs—call them Architecture 1.0—are inadequate to protect information. Spectre exploits two key micro-architecture techniques:

Instruction speculation. A processor core seeks to execute dozens of instructions concurrently by speculating past branches, committing ISA changes if speculation is correct and rolling them back when speculation is wrong. Perversely, Spectre speculatively executes instructions whose ISA changes it knows will be rolled back. Its subtle goal is to leave microarchitectural "breadcrumbs" of a supposedly hidden secret.

Caching. Each processor core uses a hierarchy of caches to make memory accesses 100X faster than DRAM memory. Like a hash table, each cache keeps data in buckets called sets to aid lookup. Caches are invisible to the ISA so their sets don't need to be restored on incorrect speculation. Spectre exploits this to place, and later find, "breadcrumbs" that reveal a secret. It thus uses the contents of a cache as a "side channel" to transmit a (secret) data value. Microarchitecture structures beyond caches have also been exploited.[3]

There has been some progress addressing Spectre since it was publicly released.[1] Software fixes include adding memory fences (that may retard speculation), placing secrets in separate address spaces, selectively flushing caches, and converting indirect branches into pseudo-returns. These changes can hurt performance and can rely on undocumented chip implementation features. Hardware fixes disable features or patch microcode of current chips—where possible—or await changes deployed in new chips.

I recommend the following paper as a much gentler introduction to Spectre than the original paper.[3] It excellently reviews how speculative execution and caches can be exploited, presents specific exploits using speculative branches that are direct (Variant 1) and indirect (Variant 2), touches on other variants, and concludes discussing software and hardware options for mitigating Spectre.

In the long run, do we manage or eliminate Spectre? We can manage it by working to discover and patch variants as they arise, much as society manages crime. More boldly, I assert we should seek to eliminate Spectre by defining an Architecture 2.0 that can be refined into implementations with provable properties regarding information exfiltration, including via microarchitecture timing. While this is hard (or even not completely possible), it is important as society depends on public computer systems to store our private information. C

References

1. Hill, M.D., Masters, J., Ranganathan, P., Turner, P. and Hennessy, J.L. On the Spectre and Meltdown processor security vulnerabilities. *IEEE Micro 39*, 2 (2019), 9–19.
2. Horn, J. Reading privileged memory with a side-channel. Project Zero; https://bit.ly/35Uuviu.
3. Kocher, P. et al. Spectre attacks: Exploiting speculative execution; arXiv preprint arXiv:1801.01203.
4. Lipp, M. et al. Meltdown: Reading kernel memory from user space. In *Proceedings of the in 27th USENIX Security Symp.* USENIX Association, 2018.

Mark D. Hill is John P. Morgridge Professor and Gene M. Amdahl Professor of Computer Sciences at the University of Wisconsin, Madison, WI, USA.