The dangers of speculation

With the growth in more subtle attacks, SoC and system designers now need to address the problems of secure execution up front, as **Chris Edwards** explains

The disclosure of the Meltdown and Spectre vulnerabilities in high-performance processors at the start of the year provided a wakeup call to the industry. It demonstrated the danger of subtle attacks that use sometimes highly indirect techniques to watch the execution of secure algorithms on a multitasking machine.

At the end of March the two attacks were joined by another called Branchscope that uses similar concepts, though Jonathan Valamehr, co-founder and COO of security-verification specialist Tortuga Logic, says: "Branchscope doesn't seem as serious. But these types of attack keep popping up all over the place."

One of the root causes of these vulnerabilities that are related is the push for performance from processors that cannot get any faster from simply increasing clock speed. High-performance processor use out-of-order and even speculative execution, where work is done but cancelled later if the wrong branch is taken, to continue working wherever possible. Processor designers assumed these would not unwittingly reveal information to hackers. But it has gradually become apparent that seemingly secure algorithms can leak information about their behaviour. The first experiments examined information leakage focused on electromagnetic detection – which demands the attacker is next to the system itself. But cache misses and miss-predicted branch, lead to timing differences that can be exploited by a remote attacker with network access.

In the case of Meltdown, the leakage was surprisingly direct because of the way speculatively executed instructions that are quickly abandoned affect caches. Hackers can insert instructions that try to read memory used by a secure processes to which they should not have access. Normally, this attempt should trigger an exception that terminates the program. The cunning part is that, in the processors affected by Meltdown, the hacker hides them behind an instruction guaranteed to avoid that supposedly illegal instruction. The processor may try to run it speculatively. But because the illegal instruction is never finalised, the processor never traps it. It simply ignores it in the expectation the result will be quickly overwritten by later instructions. But the data has been loaded into the cache, which changes the behaviour of the algorithm being targeted in a predictable way. Side-channel attacks that monitor cache behaviour, which have been the focus of research for more than a decade (see "Careless whispers", NE, 9 January 2018), make it possible to uncover what the victim algorithm is doing.

"Secure Check could have been used to detect the Meltdown and Spectre attack. However, the verification engineer would have to remodel and simplify the target design and simplify before applying the tool."
Joe Hupcey III

The worrisome aspect to vulnerabilities such as Meltdown and Spectre is that such security consequences are difficult to spot and often the combination of separate design decisions that span circuit design to software choices. The ability to read out cache data using Meltdown was made easier by the way that modern operating systems quietly share pages of data between processes in ways applications programmers do not anticipate.

The simplest strategy for overcoming the kinds of problems caused by these vulnerabilities, which depend on the interactions between software running on a shared processor and memory subsystem, is enforced separation. Mobile phones and TV-access protection have long used this approach, pushing the most sensitive algorithms into a dedicated processor that can be designed and verified independently.

"If I were creating a mission critical app, I would definitely go the separation way. But not doing that would be the best for performance. Designers have to play a game of how many security issues or what kinds of leaks can they tolerate to get the tradeoffs they want," Valamehr says.

Even separation of cores is not enough to guarantee security. Numerous leak paths can still exist between the supposedly secure subsystem and surrounding infrastructure. Debug channels, for example, have been the cause of disclosed vulnerabilities providing access to protected registers through channels the chip designers did not expect to be accessible by hackers.

Formal verification

One way around the problem of accidentally designing in leakage paths is to use formal verification. Formal assertions let verification engineers ask questions of the design that EDA tools can check exhaustively against the circuitry using mathematical proofs. Tool vendors such as Cadence Design Systems and Mentor, a Siemens business, have built applications on top of these formal-verification technologies.

"Consumer-electronics companies use Secure Check to exhaustively verify the paths to the internal registers holding the device's private key can't be compromised," says Joe Hupcey III, verification product technologist at Mentor. Developers of safety-critical systems tend go further, Hupcey says, using the tool to ensure that there is, for example, only one control path to an actuator output.

Formal tools by themselves tend to hit limits of tool capacity, though the tools vendors work constantly to push this limit higher. If software-hardware interactions are included in the analysis, the scale of the problem becomes much bigger. This calls for analysis using simulation or emulation, often alongside formal for more targeted checks. One option is to use formal to demonstrate that potential leakage paths exist and use that information to determine what needs to go into programmer guidelines.

```
raise_exception();
// the line below is never reached
access(probe_array[data * 4096];
```

Listing 1: A toy example to illustrate side-effects of out-of-order execution

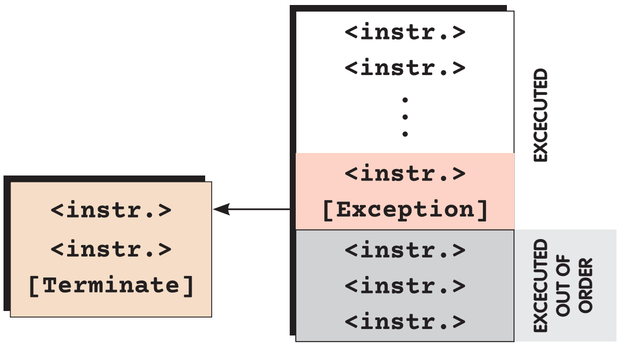

Figure 1: Hackers can exploit speculative execution in vulnerable processors by presenting instructions that will be aborted before protection logic kicks in

ARM used Cadence's SVP to evaluate the barriers between the secure and non-secure modes in recent embedded processor-core designs and determine what security guarantees the hardware can realistically enforce. To improve performance when switching modes, the ARM processors do not, by default, flush register contents when they switched out of secure mode. As a result, the hardware analysis cannot claim that this switch is secure by itself – the resulting guidelines tell programmers to empty any registers that hold sensitive data before switching modes.

A similar analysis of the situation encountered by Meltdown would probably lead to a similar result: that the hardware cannot provide a secure guarantee. Hupcey says: "This was a specification bug as much as anything else. Part of the bug was the 'residue' left in the cache lines.

"Secure Check could have been used to detect the Meltdown and Spectre attack. However, the verification engineer would have to remodel and simplify the target design and simplify before applying the tool," Hupcey adds.

One modification, Hupcey suggests is to define the desired situation as one where secure data value must not reach a cache line nor be used as an address in an unsecure mode. "Then the problem could have been detected," he says.

Spectre is a more subtle problem but still amenable to verification automation, if not in a pushbutton sense. Valamehr says the mix of dynamic and formal techniques his company's tools perform looks for problems such as timing side-channels. "We think we could have found Meltdown and Spectre had we been part of those design teams."

Hupcey notes: "A verification engineer versed in formal would have to get creative and structure the formal flow manually. Timing-related information can be used to target the problem."

Whereas simulation might demonstrate the timing difference between cache hit and miss scenarios directly, formal verification might look at the root cause that leads to the timing difference and check whether the valid-data bits in the cache can change state during execution of the secure algorithm. If that is the case, preloading and locking the cache might deter the bulk of cache-based timing attacks on that algorithm.

The requirement for SoC and system designers now is to address the problems of secure execution up front rather than play a game of cat and mouse with hackers and security researchers. Valamehr says: "If you find the problem earlier the cost of the fix is much lower. After the silicon has shipped it's a recall or you can try to patch it up and feel the effects of the negative press."