Microarchitectural Attacks

Row Hammer

- DRAM refreshes bits, default 64ms
  - Refresh is necessary as each bit is a capacitor slowly leaking
- Disturbance errors were known since 1970s in DRAM, with the first modules
- ~2014 researchers showed that specially crafted execution can induce errors

Related - NVM Wearout

- Non-volatile memory
  - Phase change memory (PCM)
  - Writing requires physical change

Side-channel Attacks

- SMT – shares some hardware
  - What if the thing we wanted to attack is on the other context / core?
- What data is accessed when?
  - Setup dummy data and time accesses
- RSA secret key
  - How does each bit flow through branches?
  - How does each bit flow through FUs?

Review: Superscalar

- Speculative execution
  - Instructions might run on predicted paths
  - Leave no side effects

- What about the cache?

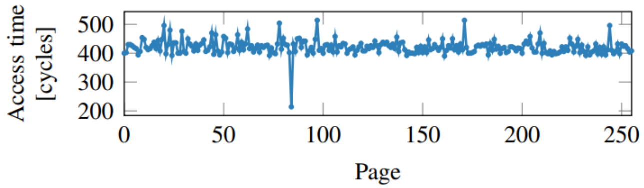
Spectre

- Train the branch predictor
- Prepare the cache
- Put speculative code that accesses memory location
  - Generally runs in JS (or similar) environment
- Find which cache set had an eviction from the speculative code

Meltdown

- In Spectre, targeting VM sandboxes to learn about the sandbox
- Meltdown targets the kernel

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



Microarchitectural Replay Attacks

- Suppose you do not entirely trust the software
  - Can you still do computation?
  - Yes. Hardware can fully prevent OS from accessing a running process's memory
    - Is hardware actually secure?

How can an OS observe?

- Side-channel attacks
  - But those may be noisy
- Eliminate noise
  - Rerun the secret code
  - How? Force a page fault, but not fix

Port Contention Attack*

Victim (in SGX):

```
if (secret) {
  // use shared resource
} else {
  // don't use shared resource
}
```

Attacker (controls OS):

```
while (true) {
  start = time();
  // use shared resource
  latency = time() - start;
}
```

Attacker can infer the secret based on the measured latency:

- If latency > threshold: secret = 1;
- If latency <= threshold: secret = 0;

However, this side-channel is noisy, attacker needs repeated victim execution to be confident

How to force victim to repeatedly execute vulnerable code?

*Aldaya et al. "Port contention for fun and profit." (SP'19)

Microarchitectural Replay Attacks* (MRAs)

Insight: Attacker triggers a large or unlimited number of pipeline squashes in the victim thread to replay vulnerable code

Squash

Victim (in SGX):

```
load x; // x is public
…
if (secret) {
  // use shared resource
} else {
  // don't use shared resource
}
```

✗ Page fault

Execute!

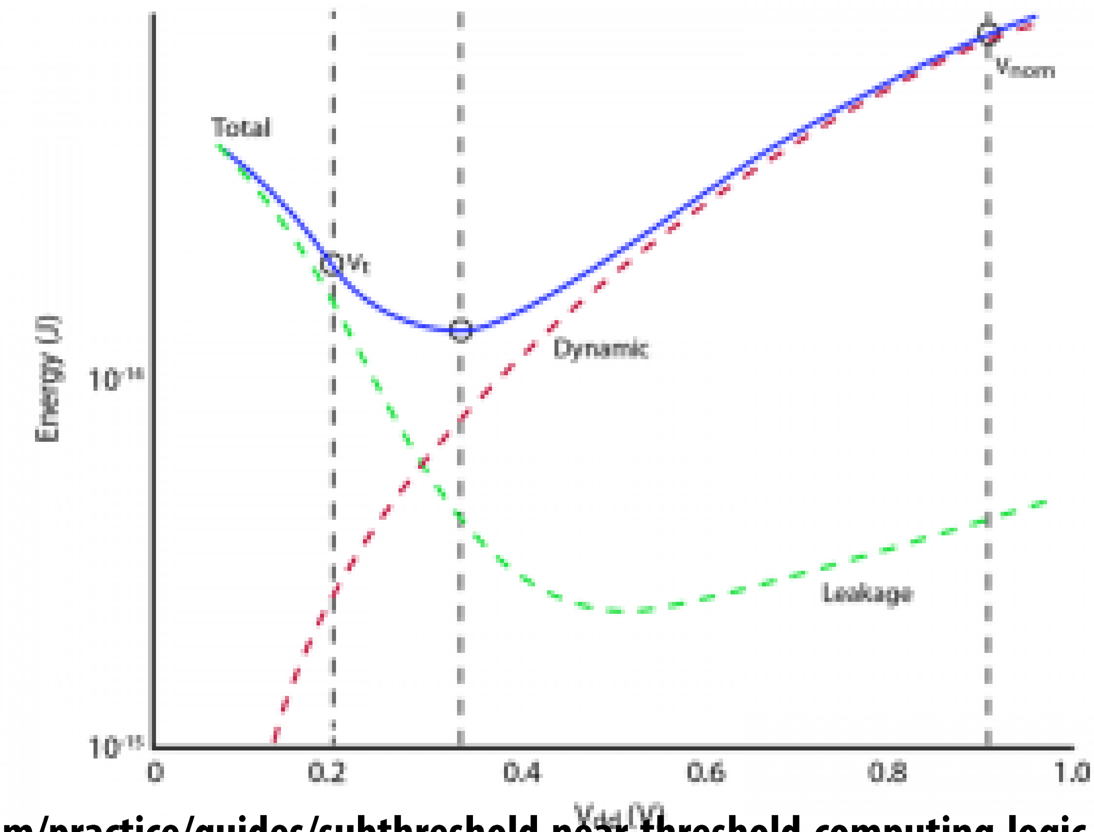
- Attacker clears page-table entry present bit of *x* and flushes TLB
- Victim speculatively executes vulnerable code
- Page fault occurs in the victim thread. Victim squashes the pipeline
- Victim invokes OS (controlled by attacker)

MRAs are beyond speculative execution side-channel attacks (e.g., Spectre)

* Skarlatos et al., "MicroScope: Enabling Microarchitectural Replay Attacks" (ISCA'19)

Near Threshold Computing

- **Transistors normally run at Vnom**
  - **Around 0.3, the transistor power is minimalized**
  - **But Vdd has to be >0.2 (Vt) on a normal transistor**



https://www.techdesignforums.com/practice/guides/subthreshold-near-threshold-computing-logic-ntv/

Approaching the Threshold

- As supply voltage drops
  - Transistor frequency has increased variation
  - Transistors may become unreliable

- For example, an adder may have some errors
- A cache or memory could
  - Lower refresh rate
  - Least significant floating point bits are dropped

Approximate Computing

- Can a program benefit from computing near the threshold?
  - Yes!
  - Many algorithms are already approximations
  - Increasing the error tolerance can permit
    - Faster running time (less iterations)
    - Using error-prone storage or ALUs

- One example:
  - K-means clustering could introduce 5% error to save 50x energy

Intermittent Computing

- We want to deploy many devices into the wild
  - Simple sensors, etc
- Devices cannot be part of the electrical grid
  - Derive power from ambient conditions (light, sound, RF, etc)
- Device stores power into capacitor
  - When capacitor is full, start computing
  - When capacitor is nearly empty, save results to flash, NVM, etc

Making computation stop

- **How do you design a processor that will be interruptible?**

- **How do you modify computation so that it can resume?**