Introductions

David Kohlbrenner (he/him)
Course objectives

1. Familiarize you with what aspects of physical systems affect software security

2. Gain a detailed understanding of current and historical work on specific families of hardware related attacks and defenses

3. Build hands-on experience with a major project in one specific sub-area
But why?

Aren’t most problems phishing/data breaches/etc?
What is/isn’t “hardware security?”
Side-channels: conceptually

A program’s implementation (that is, the final compiled version) is different from the conceptual description.

Side-effects of the difference between the implementation and conception can reveal unexpected information.

° Thus: Side-channels
Covert-channels

Many times a covert channel is demonstrated:
- unusual ways to have information flow from thing A to thing B

If this is an intentional usage of side effects, it is a covert channel

*Unintentional* means it is a side-channel

The same mechanism can be used as a covert-channel, or abused as a side-channel
Class plan
Course structure

- Per-class readings & writeups

- Present papers + lead discussion

- Project
  - Writeup
  - Presentation
Leading discussions

- 2 of the 3 classes per-topic are student led

- You will need to sign up to lead $N$
  - This means presenting a summary of the readings
  - Providing a series of discussion questions based on the class questions

Signup sheet on edstem
Project

- Group of 2-3
- Will want to start planning soon
- I’ll meet with groups about projects
  - This will happen next week
- Let's find something fun and interesting for every group
- Required:
  - Presentation (10 minutes-ish)
  - Paper-style writeup (6-10 pages)
Project ideas

- MUST: have hardware as a critical component
  - E.g. if you can imagine the project without a specific hardware implementation mattering, it isn’t an appropriate project

- Otherwise, very open to ideas!
Course misc:

- We’ll follow school COVID policies
  - Right now that means: masks are optional but “encouraged for the first 2 weeks”
  - If it changes, we’ll change to match

- Attendance is (for now) required in-person
  - This is a discussion course!
Hardware basics
- 9th-generation Intel Core i7-9750H 6-core processor
- 16x Micron MT40A1G8SA-07S 8 Gb DDR4 SDRAM (16 GB total)
- AMD Radeon Pro 5300M mobile GPU
- 4x Samsung K4Z80325BC-HC14 8 Gb GDDR6 RAM (4 GB total)
- Toshiba TSB4227VE8434CHNA11926 and TSB4227VE8437CHNA11926 flash storage (512 GB total)
- Apple T2 APL1027 339S00536 coprocessor
- Intel JHL7540 Thunderbolt 3 controller
- Intel SR40F platform controller hub
- Texas Instruments CD3217B12 (likely power controllers)
- 338500267-A0 (likely Apple PMIC)
- Texas Instruments TPS51980B power controller
- 339500610 (likely Apple Wi-Fi/Bluetooth module)
- Intersil 6277 PWM modulator
- Renesas 225101C
- Apple **APL1W81** A12 Bionic SoC layered over Micron MT53D512M64D45B-046 4 GB LPDDR4X SDRAM
- STMicroelectronics STB601A0 power management IC (possibly for Face ID)
- 3x Apple/Cirrus Logic 338S00411 audio amplifiers, two for stereo and one for haptics
- Apple/Dialog Semiconductor 338S00383-A0 power management IC
- Apple 338S00456 power management IC
- Apple/Dialog Semiconductor 338S00375 system power management IC
- Texas Instruments SN2600B1 battery charger IC
The diagram illustrates the components of a computer's processor, including the front-end, memory pipeline, and out-of-order engine. The front-end includes instruction predecode and fetch (16 bytes), instruction queue (2x25 entries), instruction multiplexers, and branch prediction unit. The memory pipeline shows L1i Cache (32 kib 8-way), L1d Cache (32 kib 8-way), L2 Cache (256 kib 4-way), line fill buffer (10 entries), and instruction multiplexers. The out-of-order engine includes physical register file, retirement unit, uOP scheduler, unified reservation station (97 entries), execution units, and control data busses. The diagram also includes reference to System Agent, Display Controller, PCIe, and Memory Controller.
Threat models
Starting with threat models

What are my adversary capabilities?

What does my adversary gain?
“Realistic” threat models

- Physical access?
  - OK, but *how much*?
- “Nearby” access?
- User-level access?
- Interaction with remote servers?
Examples:

Microarchitectural attacks
- Adversary can run usermode code
- Or can run JS/similar
- Attempts to read secret data they don’t own

Physical side-channels
- Adversary can touch/measure machine
- Tries to learn secret data
Examples:

Glitch attacks

- Adversary can partially take apart system
- Has access to sophisticated equipment
- Tries to induce incorrect computation

Remote timing

- Adversary can interact with a server via ‘normal’ methods
- Attempts to learn secrets via timing variation
Concrete example:

M1RACLES (CVE-2021-30747) is a covert channel vulnerability in the Apple Silicon "M1" chip.
M1RACLES

“A flaw in the design of the Apple Silicon “M1” chip allows any two applications running under an OS to covertly exchange data between them, without using memory, sockets, files, or any other normal operating system features. This works between processes running as different users and under different privilege levels, creating a covert channel for surreptitious data exchange.

The vulnerability is baked into Apple Silicon chips, and cannot be fixed without a new silicon revision.”

https://m1racles.com/
So you’re telling me I shouldn’t worry?

Yes.

What, really?

Really, nobody’s going to actually find a nefarious use for this flaw in practical circumstances. Besides, there are already a million side channels you can use for cooperative cross-process communication (e.g. cache stuff), on every system. Covert channels can’t leak data from uncooperative apps or systems.

Actually, that one’s worth repeating: **Covert channels are completely useless unless your system is already compromised.**

So how is this a vulnerability if you can’t exploit it?

It violates the OS security model. You’re not supposed to be able to send data from one process to another secretly. And even if harmless in this case, you’re not supposed to be able to write to random CPU system registers from userspace either.

It was fairly lucky that the bug can be mitigated in VMs (as the register still responds to VM-related access controls); had this not been the case, the impact would have been more severe.
Other common topics
Constant-time programming

Defensive programming techniques

Goal: program whose runtime is independent of secret inputs

Will include many platform specifics
Isolation

Processes
Sandboxing
Kernel-userspace isolation
Enclaves and Trusted Execution
Cache side-channels

Attacker wants to observe cache state

Many attacker models/defenses