CS111, Lecture 26
Ethics and Trust, continued

Learning Goals

- Reflect on aspects of trust, when we trust systems/others, and how we choose to trust systems/others
- Learn about examples of trust / isolation not being upheld in systems

Trust + Operating Systems

- Properties of OSes (**immense scale**) provide a unique lens through which to examine how we **trust** software.
- All software, especially OSes, can have a large impact on the people that use it, and they can put significant trust in that software.
- Examining ideas about trust can help us better consider how we might approach building trust into the software we build.

Plan For Today

- Who/what do we trust, and why? (from last time)
- What do we do when trust is not upheld?
- How might we approach building trust into the software we build?

Plan For Today

- **Who/what do we trust, and why?**
- What do we do when trust is not upheld?
- How might we approach building trust into the software we build?

Trust

Trust is to stop questioning the dependability of something.

- Efficiency/safety tradeoff: trust lowers the barriers of monitoring and questioning (more efficient)
- Involves *intentions, dependence, vulnerability/risk*
- **Agency:** our capacity to take actions that align with our goals
  - "when we trust, we try to make something a part of our agency... To unquestioningly trust something is to let it in—to attempt to bring it inside one's practical functioning." - CT Nguyen
  - **Trusting software is extending agency**
- ***Agential gullibility:*** *trusting more than is warranted*

3 Paths to Trust

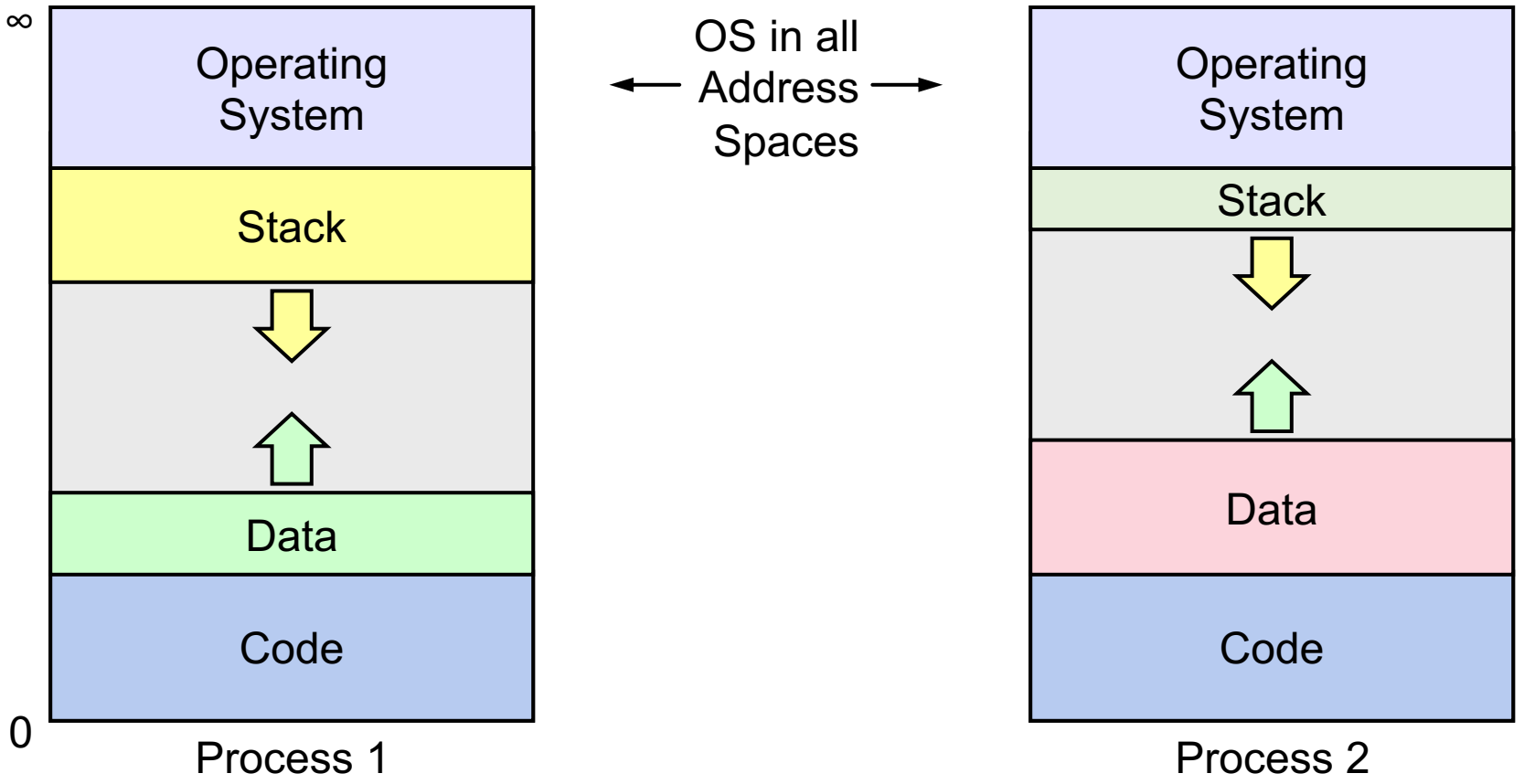
1. **Trust by Assumption:** trust absent any clues to warrant it
    1. Example: using unknown 3rd party library because deadline is approaching
    2. Example: warnings from others about imminent danger (e.g. “look out for the car!”)
2. **Trust by Inference:** trust based on information, e.g. past performance, characteristics, institutions
    1. Example: trust in brands or affiliation (weaker)
    2. Example: past performance (stronger)
    3. Example: trust in prior versions of software
3. **Trust by Substitution:** trust by implementing system to partly replace the need to trust something (“Plan B”)
    1. **E.g.** set an alarm on a second device in case the alarm on your phone doesn’t work
    2. **E.g.** using unique, fake per-app emails for login, in case your personal info is leaked

Paul B. de Laat: *How can contributors to open-source communities be trusted? On the assumption, inference, and substitution of trust*

Plan For Today

- Who/what do we trust, and why?
- **What do we do when trust is not upheld?**
  - **Case study:** Meltdown vulnerability
- How might we approach building trust into the software we build?

OS and User in Same Address Space



OS Execution

How does virtual memory work when the OS runs?

OS has space in every process's virtual address space. Not a duplicate of OS; every virtual space could map to same physical memory.

Problem: don't want user program accessing OS pages.

Solution: new bit in page table that marks kernel-only pages. When in user mode, not accessible, but accessible when OS is running.

Meltdown

Meltdown is a **hardware vulnerability** publicly disclosed in 2018 that allows a program to access kernel-only pages. (https://meltdownattack.com)

> "Meltdown is a novel attack that allows overcoming memory isolation completely by providing a simple way for any user process to read the entire kernel memory of the machine it executes on, including all physical memory mapped in the kernel region."

Meltdown

Meltdown is a **hardware vulnerability** publicly disclosed in 2018 that allows a program to access kernel-only pages. (https://meltdownattack.com)

- fixes in later processors, patched in OSes

Discussion Question #1

How do we decide how / whether to fix a potential vulnerability / violation of trust? How do we evaluate tradeoffs in performance penalties, user convenience, and other factors?

- E.g., what if vulnerability is unlikely or difficult to exploit?
- E.g., what if fix causes a performance penalty or other user inconvenience?
- E.g., do we make a fix opt-out or opt-in?

Respond on PollEv: pollev.com/cs111
or text CS111 to 22333 once to join.

Discussion Question #2

How, if at all, do we hold parties accountable? Who holds them accountable? Examples of accountable parties could include:

- Hardware designers (e.g. Intel)
- OS designers (e.g. Microsoft Windows, Google Android, Apple iOS)
- App developers
- Users

Plan For Today

- Who/what do we trust, and why?
- What do we do when trust is not upheld?
  - **Case study:** Meltdown vulnerability
- **How might we approach building trust into the software we build?**

Building In Trust

Considerations might include:

- Who are the stakeholders?
- How pervasive is it?
- What are the long-term intentions?

Stakeholders

Direct stakeholders: directly interact with the system

Indirect stakeholders: affected by the system without directly using it

Example: medical office use device

- **Direct stakeholders:** medical professional operating the device, service technician maintaining device
- **Indirect stakeholders**: patients

Pervasiveness

- How widespread is the use?
- What is the use case? (personal, recreation, critical infrastructure)
- Considerations about crossing national boundaries (different rules, customs, infrastructure)
- Considerations about cultural and political implications

Time

- Support duration (long-term support) - assign2
- Obsolescence: how do we manage end of support?
  - **Example:** long-term support for operating systems (assign2)
  - **Example:** software updates
  - **Example:** online services (e.g. games with online components)
- Other scenarios where product/support may not be guaranteed forever
  - **Example:** subscription content services
  - **Example:** company going out of business, no longer maintains/supports product
- What does halting use look like?
  - **Example:** Threads app initially not supporting account deletion without deleting Instagram account

Discussion Question #3

What are other examples of products you have used where you may or may not have factored in how long that product/software might be supported? What can/should the product creators do?

Plan For Today

- Who/what do we trust, and why?
- What do we do when trust is not upheld?
- How might we approach building trust into the software we build?

Key Takeaways

Trust is often required, powerful, and dangerous. Key design challenge is how we design structures that enable us to substitute trust.

1. Trust amongst parties can be intertwined
2. Trust is about extending agency, enabling “unquestioning attitude”
3. Trust emerges through assumption, inference, substitution
4. Can design ways to (partially) substitute need to trust

Considerations for trust include stakeholders, pervasiveness, time

Reflections on Trusting Trust

TURING AWARD LECTURE

Reflections on Trusting Trust

To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software.

KEN THOMPSON

https://www.cs.cmu.edu/~rdriley/487/papers/Thompson_1984_ReflectionsonTrustingTrust.pdf

Plan For Today

- Who/what do we trust, and why?
- What do we do when trust is not upheld?
- How might we approach building trust into the software we build?

Lecture 26 takeaway:
Trust is often required, powerful, and dangerous. Key design challenge is how we design structures that enable us to substitute trust.