CS111, Lecture 16
Trust and Operating Systems + assign4

口罩强烈推荐
CS111 Lecture
Trust and Operating Systems

Benjamin Xie, Ph.D.
Embedded Ethics Fellow
benjixie@stanford.edu | benjixie.com

made with William Grant Ray III, Xiyu Zhang, Liana Keesing, Swayam Parida, Prof. Nick Troccoli, Prof. John Ousterhout

Do Now:
1. Say hello to your neighbor!
2. Think of an OS you use. Discuss what you use it for and how you trust it. Add any thoughts on pollEV!
Think of an OS you use. What do you use it for and how do you trust it?

Nobody has responded yet.

Hang tight! Responses are coming in.
Hi, I’m Benji!

Why I’m here: Embedding ethics into CS courses (14 so far!)

Research: human-data interactions (computing education + HCI research)

My path:

• BS + M.Eng. ("co-term") in CS at MIT
• Ph.D. at University of Washington
• Embedded Ethics Postdoctoral Fellow at Stanford HAI, Ethics Center
What is an OS that you use? For what?

How do you trust that OS?
Plan For Today

- Motivation: Importance of trust in OS
- What is trust?
- How does trust emerge?
- Example: Trusting Linux
Plan For Today

• Motivation: Importance of trust in OS
• What is trust?
• How does trust emerge?
• Example: Trusting Linux
Learning Goals

Understand how trust emerges and manifests with operating systems in given contexts

Appendix B: Is “Software Engineering” an Oxymoron?

By Alan Kay

Real Software Engineering is still in the future. There is nothing in current SE that is like the construction of the Empire State building in less than a year by less than 3000 people: they used powerful ideas and power tools that we don’t yet have in software development. If software does “engineering” at all, it is too often at the same level as the ancient Egyptians before the invention of the arch (literally before the making of arches: architecture), who made huge structures with hundreds of thousands of slaves toiling for


CS111 Ethics Topic: Trust

Assign 2

Long term support

This lecture

Trust (what it is, how it manifests)

Assign 4

Trust & Race Conditions

Future lecture

Trust in context
How do we trust OS (open vs closed)?

The Linux kernel and its entire ecosystem of operating system distributions are built around the values of openness, transparency, agility and trustworthiness. These values are what lay the foundation for modern software security.
• OS provides efficiency through standardization
• Users rely on technology built on OS
• App developers build off of OS
• Systems programmers make decisions that ripple far and long
Plan For Today

• Motivation: Importance of trust in OS
  • OS is public infrastructure of software

• What is trust?

• How does trust emerge?

• Example: Trusting Linux
Trust as an unquestioning attitude

• Trust is to stop questioning the dependability of a thing

• Efficiency/safety tradeoff:
  • Trust lowers the barriers of monitoring and questioning (more efficient)

• Involves
  • Intentions
  • Dependence
  • Vulnerability/Risk

• Example: what/who did you trust to get to class today?
Trusting software is extending agency

- **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.”

- Example: glucose monitoring

CT Nguyen: Trust as an unquestioning attitude
Risk: Agential Gullibility

- Trusting more than warranted
- Difficult to b/c software changes, hard to inspect
- Example: glucose monitoring issues w/ Android update
Takeaway: Trust is powerful, necessary, risky

If I trust people or things (e.g. SW), I …
- Integrate it with my own functioning
- Work more efficiently with them (stop questioning)
- Feel betrayed when they fail us

=> Trust (by extending agency) with great care!
Think back to the person/thing/service you trusted...
How does trusting them extend your agency/functioning?

How might/did you exhibit *agential gullibility*? (trust more than is warranted)

What would be/was the result of your trust being violated?
Think back to the person/thing/service you trusted... TurboTax Tax Preparation Software

How does trusting them extend your agency/functioning?
> Able to complete taxes more efficiently and had more confidence I did it correctly.

How might/did you exhibit *agential gullibility*? (trust more than is warranted)
> Tricked into paying for service even though it was legally supposed to be free.

What would be/was the result of your trust being violated?
> Feeling of betrayal. Stopped using software.

Plan For Today

• Motivation: Importance of trust in OS
  • OS is public infrastructure of software

• What is trust?
  • Extending agency to software through unquestioning attitude

• How does trust emerge?

• Example: Trusting Linux
Three paths to trust

1. Assumption: trust absent any clauses to warrant it
   a. E.g. using unknown third party library b/c deadline nearing

2. Inference: reputation is based on past performance, characteristics, institutions
   a. Some weaker (e.g. trust in brands or affiliation)
   b. Some stronger (e.g. past performance)
   c. Trust in prior versions of software

3. Substitution: structural arrangements that partly replace need for trust
   a. Often involves separation of code, responsibilities
   b. E.g. user permissions of file system, keeping personal info off work accounts, devices

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

*Identify one person/thing/service that you trust by...*

**Assumption** (trust absent clues to warrant it)

**Inference** (trust from evidence of past performance, characteristics, institutions)

**Substitution** (structural arrangement to partly decrease the need for trust)
Self-assessment on how trust manifests

*Identify one person/thing/service that you trust by...*

Assumption (trust absent clues to warrant it)

> Anyone warning me about imminent danger (e.g. “look out for the car!”)

Inference (trust from evidence of past performance, characteristics, institutions)

> Password management service (inferred trust based on online reviews, review of privacy policy)

Substitution (structural arrangement to partly decrease the need for trust)

> Keep some important passwords stored locally and not on app
Plan For Today

• Motivation: Importance of trust in OS
  • Trust amongst tech users, app developers, and OS developers is intertwined

• What is trust?
  • Extending agency to software

• How does trust emerge?
  • Assumption, inference, substitution

• Example: Trusting Linux
Linux is hard to trust

1.1 million commits
13.9k contributors
8+ million lines of code
Users Trusting Linux

- Why: People use Linux-based tools to extend their agency
  - Android smartphones
  - 13.6% of servers
  - Almost all supercomputers

- How trust emerges?
  - Assumption
    - “never thought about it”
    - ”no other option”
  - Inference
    - open source
    - previous use
  - Substitution
    - Redundant security protocols (e.g. strong password, isolate/encrypt sensitive files)
- Why: Standardization and tools of OS enable efficiency
  - High cost to build and maintain new OS
  - Familiar => lowers learning time developers

- How trust emerges?
  - Assumption: rare given affordances to infer trust
  - Inference
    - Used by other app developers (lots of stars on GitHub)
    - trust Linus Torvalds
  - Substitution
    - code is open source (read it, fork it)
    - Add “redundant” checks in code (ex: spurious wakeup)
- Why: No single person can build & maintain an OS. Need to extend agency to others to support.

- How trust emerges?
  - Assumption: rarely happens
  - Inference
    - Known in community
    - Quality of previous code submissions
  - Substitution
    - Formalization: tools and procedures to streamline cooperation
    - Division of roles
    - Decision making: Linus has final authority

“I don’t like the idea of having developers do their own updates in my kernel source tree. (…) there really aren’t that many people that I trust enough to give write permissions to the kernel tree.”

– Linus Torvalds
Abstractions as way to substitute trust

**strlcat**: size bound string copying & concatenation

Since 1998 (few changes since)

```c
size_t strlcat(char *dst, const char *src, size_t siz)
{  
    char *d = dst;  
    const char *s = src;  
    size_t n = siz;  
    size_t dlen;  
    /* Find the end of dst and adjust bytes left but don't go past end */  
    while (n-- && *d != '\0')  
        d++;  
    dlen = d - dst;  
    n = siz - dlen;  
    if (n == 0)  
        return(dlen + strlcpy(d, src, siz));  
    while (*s != '\0') {  
        if (n == dlen)  
            *d++ = *s;  
            n++;  
        }  
        *d++ = '\0';  
    return(dlen + (s - src));  
}  
```

**curl**: tool for transferring data from or to a server using URLs. (used by 20 bil.)

Trust is getting harder b/c code complexity beyond comprehension of single person. (example of substitution: SOLID, Barbara Liskov)
Old does not (necessarily) mean trustworthy!

- SOCKS5: enables anonymous network communication (e.g. when using Tor to access internet, VPNs)
- Hostname can only be 255 bytes
- Bug introduced where long hostname (e.g. https://aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa/)
  overflows buffer that stores

```bash
https://daniel.haxx.se/blog/2023/10/11/how-i-made-a-heap-overflow-in-curl/#comments
```

- Bug existed for 3.6 yrs
- Resolution: patch made (throw error), test case added
- Robust substitution: rewriting in memory-safe language (Rust)

““Every human make mistake but spotting the mistake, acknowledging it and explaining it to a wide audience takes a very good human... this makes Curl even more trustable than before.”

- commenter on dev blogpost
1. Trust amongst tech users, app developers, and system programmers is intertwined.

2. Trust is about extending agency, enabling “unquestioning attitude”.

3. Trust emerges through assumption, inference, substitution.

4. Linux kernel to used broadly and large, so users, app developers, system programmers must trust through inference and substitution.

5. Can design ways to (partially) substitute need to trust.

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

Benjamin Xie, Ph.D.
Embedded Ethics Fellow
benjixie@stanford.edu | benjixie.com
Assignment 4 consists of an ethics exploration + implementing 2 monitor pattern classes for 2 multithreaded programs.
Unique Locks

• It is common to acquire a lock and hold onto it until the end of some scope (e.g. end of function, end of loop, etc.).

• There is a convenient variable type called `unique_lock` that when created can automatically lock a mutex, and when destroyed (e.g. when it goes out of scope) can automatically unlock a mutex.

• Particularly useful if you have many paths to exit a function and you must unlock in all paths.
We lock at the beginning of this function and unlock at the end.

```c++
void Bridge::leave_eastbound(size_t id) {
    bridge_lock.lock();
    n_crossing_eastbound--;
    if (n_crossing_eastbound == 0) {
        none_crossing_eastbound.notify_all();
    }
    print(id, "crossed", true);
    bridge_lock.unlock();
}
```
We lock at the beginning of this function and unlock at the end.

```cpp
void Bridge::leave_eastbound(size_t id) {
    unique_lock<mutex> lock(bridge_lock);
    n_crossing_eastbound--;
    if (n_crossing_eastbound == 0) {
        none_crossing_eastbound.notify_all();
    }
    print(id, "crossed", true);
}
```

*Auto-locks permitsLock here*
We lock at the beginning of this function and unlock at the end.

```c++
void Bridge::leave_eastbound(size_t id) {
    unique_lock<mutex> lock(bridge_lock);
    n_crossing_eastbound--;
    if (n_crossing_eastbound == 0) {
        none_crossing_eastbound.notify_all();
    }
    print(id, "crossed", true);
}
```

*Auto-unlocks permitsLock here (goes out of scope)*
void Bridge::arrive_eastbound(size_t id) {
    bridge_lock.lock();
    print(id, "arrived", true);
    while (n_crossing_westbound > 0) {
        none_crossing_westbound.wait(bridge_lock);
    }
    n_crossing_eastbound++;
    print(id, "crossing", true);
    bridge_lock.unlock();
}
void Bridge::arrive_eastbound(size_t id) {
    unique_lock<mutex> lock(bridge_lock);
    print(id, "arrived", true);
    while (n_crossing_westbound > 0) {
        none_crossing_westbound.wait(lock);
    }
    n_crossing_eastbound++;
    print(id, "crossing", true);
}
void Bridge::arrive_eastbound(size_t id) {
    unique_lock<mutex> lock(bridge_lock);
    print(id, "arrived", true);
    while (n_crossing_westbound > 0) {
        none_crossing_westbound.wait(lock);
    }
    n_crossing_eastbound++;
    print(id, "crossing", true);
}

Use it with CV instead of original lock (it has wrapper methods for manually locking/unlocking!)
void Bridge::arrive_eastbound(size_t id) {
    unique_lock<mutex> lock(bridge_lock);
    print(id, "arrived", true);
    while (n_crossing_westbound > 0) {
        none_crossing_westbound.wait(lock);
    }
    n_crossing_eastbound++;
    print(id, "crossing", true);
}

Auto-unlocks permitsLock here (goes out of scope)
Assign4 Data Structures

• Data structures can be used to store condition variables or state
• Structs also helpful to bundle state together and make multiple instances of structs
• **Key note: condition variables cannot be copied.** E.g. cannot create a condition variable and push onto vector. Consider how pointers might help!
Recap

• Trust and Operating Systems
• assign4

Next time: how does the OS run and switch between threads?

Lecture 16 takeaway: Trust is often required, powerful, and dangerous. Key design challenge is how we design structures that enable us to substitute trust. For assign4, you’ll explore these topics and use the monitor pattern to write multithreaded programs.

cp -r /afs/ir/class/cs111/lecture-code/lect16.