Reflections on Trusting Trust
Ken Thompson
CACM 1984, Turing Award Lecture

Grand Overview:
- 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.

Who is Ken Thompson?
- Ken Thompson is a prolific programmer responsible for some of the most important modern day computing infrastructure. This includes Unix (with Dennis Ritchie), B (with Dennis Ritchie), UTF-8 (with Rob Pike), and Go (with Rob Pike, Robert Griesemer, and Ken Thompson).
  - Aside: Dennis Ritchie, best known for creating C. Another prolific programmer. Passed away in 2011; sad day.

Introduction
- This is Ken Thompson's Turing award speech.
- Ken Thompson is known for Unix, of course, but says he hasn't worked on it for some time, so he won't be talking about it.
- Thanks Dennis Ritchie for a great number of years of collaboration.
  - One misstep: wrote, character for character, the exact same ASM program.
- Will talk about "the cutest program [he] every wrote".

Stage 1
- A self-reproducing program, also known as "quine", produces an exact copy of its source code when executed.
  - Crucially, the program takes no input from anywhere. No reading files.
- Figure 1 is an ~example of such a program.
  - How does it work?
    - Key to quines: code is data and data is code. A quine simultaneously views itself through both lenses.
    - The `char s[]` array is the program's code as data.
    - The `main` function interprets this data twice.
      1. The first loop replicates the data.
      2. The second loop replicates the data as code (text).
  - Is it really a quine? No! Why?
    - The first loops uses `%d`.
  - Is this C? Not modern C.
  - Let's do it. See `code/stage1`. Try `almost_quine.c` first. Then `make_quine.py` for a more obvious note of what's happening. Finally, `short_quine.c` for a mention about escape characters.

Stage 2:
- We can "teach" programs new facts.
- If the fact is taught to a compiler, a new version of a compiler that uses that fact can be created. The new version's source code no longer includes the teaching of fact; it just "knows" it's there.
  - The example in stage 2 is about escape sequences.
  - We "teach" by having the compiler interpret '\' 'v' as 11. This is compiler A:
    ```
    if seq == ['\', 'v']: return 11
    ```
  - We write the source to directly use '\v' to recognize '\v'. We must compile this new compiler with A. This is compiler B:
    ```
    if seq == ['\', 'v']: return '\v'
    ```
  - We can them compile the source again with B. This is compiler C:
    ```
    if seq == ['\', 'v']: return '\v'
    ```

Stage 3:
- We can self-replicate our program, and we can teach a program new facts. Let's teach our C compiler (a program) to self-replicate with a trojan horse.
- We can modify the C compiler to recognize a specific program, here a login application, and insert a bug into it.

- We can them modify the C compiler to regonize itself and insert the code that inserts the bug into the compiler. Additionally, insert code so that the code that inserts this code also gets inserted. This is self-replication.

Moral:
- Don’t trust Ken Thompson.
- In fact, don’t trust code you did not totally create yourself.
  - Okay. So I write my own C compiler. That good enough?
    - In what? Assembly? Well, that depends on the assembler.
  - Okay. So I write my own assembler. That good enough?
    - For what? X86? Well, that depends on the CPU.
  - Okay. So I design my own CPU.
    - Fab it where? TSMC? Well, they can do whatever they want.
  - Okay. So I design and fab my own CPU with my own instructions set with my own assembler with my own C compiler.
    - With whose machines?
  - Okay. I design and write the code for the machines.
    - What did you write the code on the machines in?
    - Which CPUs are they using?
    - See step 1.
  - Okay. So I assembler the CPU by hand on a breadboard.
    - Did you make the breadboard?
  - Okay. So I source metal, melt it. Source plastic, melt it. Make breadboard, writes. Make my own CPU.
    - Make sure you keep an eye on the CPU at all times.
    - How did you design the CPU? Hope it’s paper and pencil.
    - How are you getting output? Hope you made your own monitor.
    - How are you getting input? Hope you made your own keyboard.
  - Okay. So I don’t use anything or base anything off of anything unless I made it myself. I don’t use anything else to make these things unless I make those myself. I don’t let anyone touch or see anything I make. Am I good now?
    - Yes.