Programming Languages and Trust

The what, why, and how (and the hacks)

Veit Heller

January 3, 2019

Datengarten | CCCB

whoami

- I work at a consultancy.
- I hack on languages in my free time.
- zepto, Carp, cspfuck...
- I'm secretly a turtle.

Compilers

How do programming languages work?

- We usually start with a messy dichotomy and juxtapose compilers and interpreters.
  - Compilers transform source code into some form of executable code.
  - Interpreters take in source code and evaluate it directly.

How do programming languages really work?

- Most “real” implementations transform their source code first in some way.
- This representation has many names: Abstract Syntax Tree (AST), Intermediate Representation (IR), Byte Code, Bit Code, et al.
- And what about transpilers? Oh my.

How do programming languages really work?

- We have some sort of pipeline:
  - Takes in source code,
  - Transforms, and
  - Spits out another representation or evaluates it directly.

A pipeline?

A pipeline?

```
void eval(char* str) {
  int tape[3000];
  int tape_head = 0;
  while(str) {
    switch(str) {
      case '+': tape[tape_head]++; break;
      case '-': tape[tape_head]--; break;
      case '>': tape_head++; break;
      case '<': tape_head--; break;
      case '.': printf("%c", t[h]); break;
      case ',': scanf("%c", (char*)&t[h]); break;
      case '[': if(!t[h]) str = search_end_loop(str); break;
      case ']': if(!t[h]) str = search_begin_loop(str); break;
    }
    ++str;
  }
}
```

Listing 1: A silly Brainfuck VM

A pipeline?

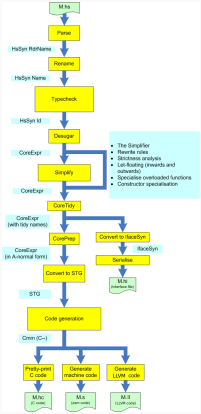

Figure 1: The GHC pipeline.

This picture was sourced from https://ghc.haskell.org/trac/ghc/wiki/Commentary/Compiler/HscMain.

Why pipelines?

- Huge pipelines might seem overkill.
- These days, they buy us modularity, clarity, and a low barrier of entry.
- Independent passes are great! I can finally do proper testing!
- The extreme end of this spectrum is nanopass compilers (cool stuff!).

Demo I—enter cspfuck

Trust

Trusting Trust

- In 1984, Ken Thompson was rightfully awarded the Turing Award.
- He wrote a three-page paper with a scary idea: malicious compilers.

The idea

- Have you heard about Quines? They are self-replicating programs.
- Have you heard about bootstrapping compilers? They are compilers that can compile themselves.
- What if we compile a “buggy” version of our compiler and ship it?

The idea

If you cannot trust your compiler, all of the programs you compile are possibly malicious. It doesn't even need to compromise existing functionality.

The idea

"The actual bug I planted in the compiler would match code in the UNIX "login" command. The replacement code would miscompile the login command so that it would accept either the intended encrypted password or a particular known password. Thus if this code were installed in binary and the binary were used to compile the login command, I could log into that system as any user."

— Ken Thompson, Trusting Trust, page 3

The idea

"Such blatant code would not go undetected for long. Even the most casual perusal of the source of the C compiler would raise suspicions."

— Ken Thompson, Trusting Trust, page 3

The idea

"This [second approach] simply adds a second Trojan horse to the one that already exists. The second pattern is aimed at the C compiler. The replacement code is a Stage I self-reproducing program that inserts both Trojan horses into the compiler. This requires a learning phase as in the Stage II example. First we compile the modified source with the normal C compiler to produce a bugged binary. We install this binary as the official C. We can now remove the bugs from the source of the compiler and the new binary will reinsert the bugs whenever it is compiled. Of course, the login command will remain bugged with no trace in source anywhere."

— Ken Thompson, Trusting Trust, page 3

Demo II—enter Michael Arntzenius

References

- A talk on Nanopass Compilers:
  `https://www.youtube.com/watch?v=Os7FE3J-U5Q`
- Ken Thompson, Trusting Trust:
  `https://www.win.tue.nl/ aeb/linux/hh/thompson/trust.html`
- Michael Arntzenius' reflections on Trusting Trust:
  `https://github.com/rntz/rotten`
- My favorite talks on Compilers and Interpreters:
  `https://github.com/hellerve/programming-talks#compilersinterpreters`
- Some of my favorite (and loathed) papers:
  `https://github.com/hellerve/ptolemy/blob/master/done.md`
- These slides: `https://github.com/hellerve/talks`

Thank you!

Questions?

Slides at https://github.com/hellerve/talks