Where does *your* compiler come from?

Vincent Ambo

2018-03-13

Norwegian Unix User Group

Introduction

Chicken and egg

Self-hosted compilers are often built using themselves, for example:

- C-family compilers bootstrap themselves & each other
- (Some!) Common Lisp compilers can bootstrap each other
- `rustc` bootstraps itself with a previous version
- ... same for many other languages!

Chicken, egg and ... lizard?

It's not just compilers: Languages have runtimes, too.

- JVM is implemented in C++
- Erlang-VM is C
- Haskell runtime is C

... we can't ever get away from C, can we?

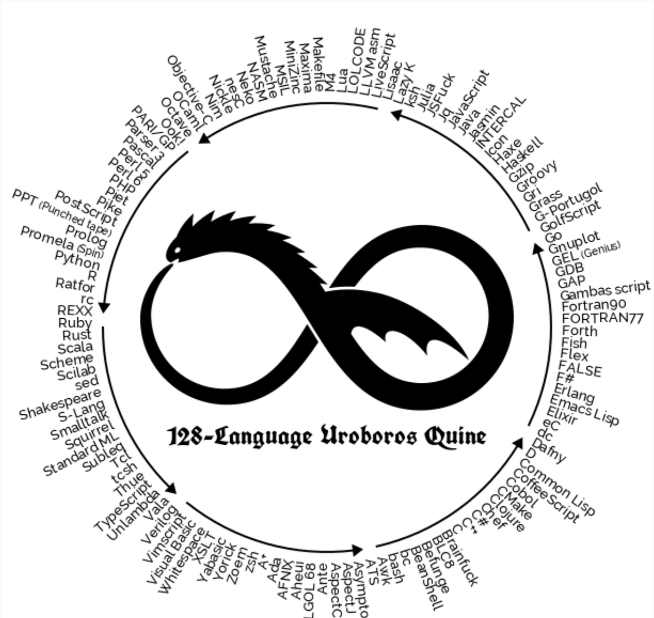
Trusting Trust

Could this be exploited?

Short interlude: A quine

```
((lambda (x) (list x (list 'quote x)))
  '(lambda (x) (list x (list 'quote x))))
```

Short interlude: Quine Relay



Trusting Trust

An attack described by Ken Thompson in 1983:

1. Modify a compiler to detect when it's compiling itself.
2. Let the modification insert *itself* into the new compiler.
3. Add arbitrary attack code to the modification.
4. *Optional!* Remove the attack from the source after compilation.

Damage potential?

Let your imagination run wild!

Countermeasures

Diverse Double-Compiling

Assume we have:

- Target language compilers A and T
- The source code of A: S_A

Diverse Double-Compiling

Apply the first stage (functional equivalence):

- $X = A(S_A)$
- $Y = T(S_A)$

Apply the second stage (bit-for-bit equivalence):

- $V = X(S_A)$
- $W = Y(S_A)$

Now we have a new problem: Reproducibility!

Reproducibility

Bit-for-bit equivalent output is hard, for example:

- Timestamps in output artifacts
- Non-deterministic linking order in concurrent builds
- Non-deterministic VM & memory states in outputs
- Randomness in builds (sic!)

Reproducibility

Without reproducibility, we can never trust that any shipped binary matches the source code!

(Partial) State of the Union

The Desired State

1. Full-source bootstrap!
2. All packages reproducible!

Bootstrapping Debian

- Sparse information on the Debian-wiki
- Bootstrapping discussions mostly resolve around new architectures
- GCC is compiled by depending on previous versions of GCC

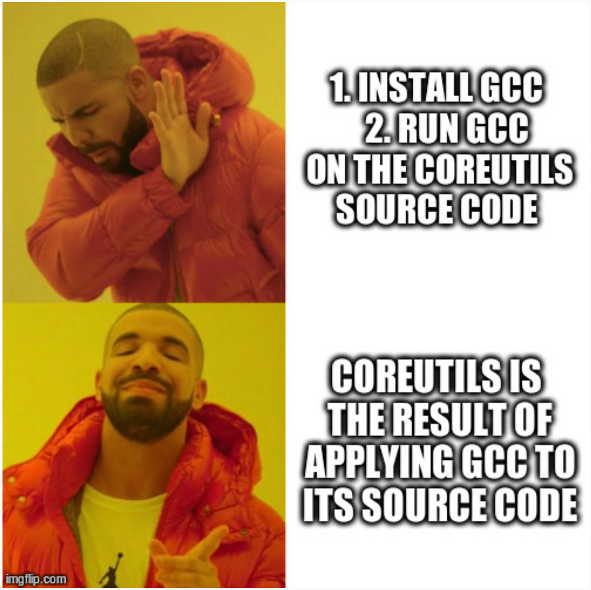
Reproducing Debian

Debian has a very active effort for reproducible builds:

- Organised information about reproducibility status
- Over 90% reproducibility in Debian package base!

Short interlude: Nix

Short interlude: Nix



Short interlude: Nix

Bootstrapping NixOS

Nix evaluation can not recurse forever: The bootstrap can not simply depend on a previous GCC.

Workaround: `bootstrap-tools` tarball from a previous binary cache is fetched and used.

An unfortunate magic binary blob ...

Reproducing NixOS

Not all reproducibility patches have been ported from Debian.

However: Builds are fully repeatable via the Nix fundamentals!

Future Developments

Bootstrappable: stage0

Hand-rolled “Cthulhu’s Path to Madness” hex-programs:

- No non-auditable binary blobs
- Aims for understandability by 70% of programmers
- End goal is a full-source bootstrap of GCC

Bootstrappable: MES

Bootstrapping the “Maxwell Equations of Software”:

- Minimal C-compiler written in Scheme
- Minimal Scheme-interpreter (currently in C, but intended to be rewritten in stage0 macros)
- End goal is full-source bootstrap of the entire GuixSD

Other platforms

- Nix for Darwin is actively maintained
- F-Droid Android repository works towards fully reproducible builds of (open) Android software
- Mobile devices (phones, tablets, etc.) are a lost cause at the moment

Thanks!

Resources:

- bootstrappable.org
- reproducible-builds.org

@tazjin | mail@tazj.in