Easier supply chain security

A container-native, full-source-bootstrapped, reproducible, and multi-signed toolchain to build all the things.

Do you know who built your base image?

```
FROM trusted/distro
RUN pkg install openssl
ENTRYPOINT ["/usr/bin/openssl"]
CMD ["rand", "-hex", "12"]
```

```
> docker build -t local/randtest -f Containerfile .

> docker run local/randtest

2a2a2a2a2a2a2a2a2a2a2a2a

> docker run local/randtest

2a2a2a2a2a2a2a2a2a2a2a2a
```

Woops, someone included libfakerand!

width: auto

Problem Statement

Build software without expecting users to trust any single human or computer.

We really didn't want to make yet another Linux distro.

Sadly, nothing we evaluated fit our threat model.

Let's first break down the buzzword bingo:

Stagex is a container-native, full-source-bootstrapped, reproducible, and multi-signed toolchain to build all the things.

Container-native

- Most sysadmins today do everything in containers
- Most release engineers today build everything in containers
- OCI container spec solves the problems of most linux package managers
- OCI syntax allows for very succinct and simple package definitions
- OCI has many competing implementations, and this is a good thing!
- OCI has official support for multiple signatures, and signing policies
- Every stagex package is just an OCI layer, which can be hash locked

Our actual BC "package" is just OCI layers all the way down

```
FROM scratch AS build
ARG VERSION
COPY --from=stagex/core-busybox . /
COPY --from=stagex/core-gcc . /
COPY --from=stagex/core-binutils . /
COPY --from=stagex/core-make . /
COPY --from=stagex/core-musl . /
COPY --from=fetch . .
RUN tar -xf sed-${VERSION}.tar.xz
WORKDIR /sed-${VERSION}
RUN --network=none <<-EOF
  set -eux
  ./configure --prefix=/usr
  make -j "$(nproc)"
  make DESTDIR=/rootfs install
EOF
FROM stagex/core-filesystem AS package
COPY --from=build /rootfs/ /
```

Full source bootstrapped

TL;DR: Solve Ken Thompsan's Trusting Trust problem:

- Stage 0: Bootstrap primitive binaries from 256 bytes of hex0
- Stage 1: Bootstrap all the way up to x86 GCC
- Stage 2: Build GCC cross compiler toolchain
- Stage 3: Building a native hosted GCC toolchain
- Stage X: Build whatever you want

Only possible because of heroic work by the Bootstrappable Builds team!

Stage0: Build bootstrap seeds with self hosting hex0 "compiler"

```
0000000 457f 464c 0101 0301 0000 0000 0000 0000
0000010 0002 0003 0001 0000 804c 0804 002c 0000
0000020 0000 0000 0000 0000 0034 0020 0001 0000
0000030 0000 0000 8000 0804 8000 0804 00b5 0000
0000040 00b5 0000 0001 0000 0001 0000 5b58 315b
0000050 6ac9 5805 cd99 5b80 6650 41b9 6602 c0ba
0000060 6a01 5805 80cd 4299 3197 89ed 4ed6 895b
0000070 6ae1 5803 80cd 8553 75c0 4005 db31 80cd
0000080 018a 0a3c e574 f685 e475 233c df74 3b3c
0000090 db74 302c 0a2c 0872 072c df24 073c ce73
00000a0 e5c1 0404 010a f7c5 7cdf 89c3 8929 b0fb
00000b0 cd04 eb80 00b4
00000b5
```

It makes more sense with comments:

https://github.com/oriansj/stage0-posix-x86/blob/master/hex0_x86.hex0

Stage1: Bootstrap up to x86 gcc toolchain from bootstrap seeds

It is basically building up from the 1970s all over again

- Step 1: hex0
- Step 9: M0 C (cc_x86)
- Step 19: scheme (GNU Mes)
- Step 20: ~C99 C (Tinycc)
- Step 56: perl 5x
- Step 83: gcc 4x
- Step 148: python 2x
- Step 158: gcc 13x (still x86)

Full steps: https://github.com/fosslinux/live-bootstrap/blob/master/parts.rst

Stage2: Pivot from x86 -> $TARGET cross toolchain

It is not that exciting.

- Step 1: Build cross binutils
- Step 2: Build minimal cross gcc (without libgcc)
- Step 3: Build libgcc against musl headers
- Step 4: Build musl
- Step 5: Build rest of gcc

Stage3: Build native $TARGET toolchain

Also not that exciting.

- Step 1: Build musl
- Step 2: Build binutils
- Step 3: Build make
- Step 4: Build gcc
- Step 5: Build busybox

This covers "full source bootstrapped"

Now we can talk about what we mean by reproducible.

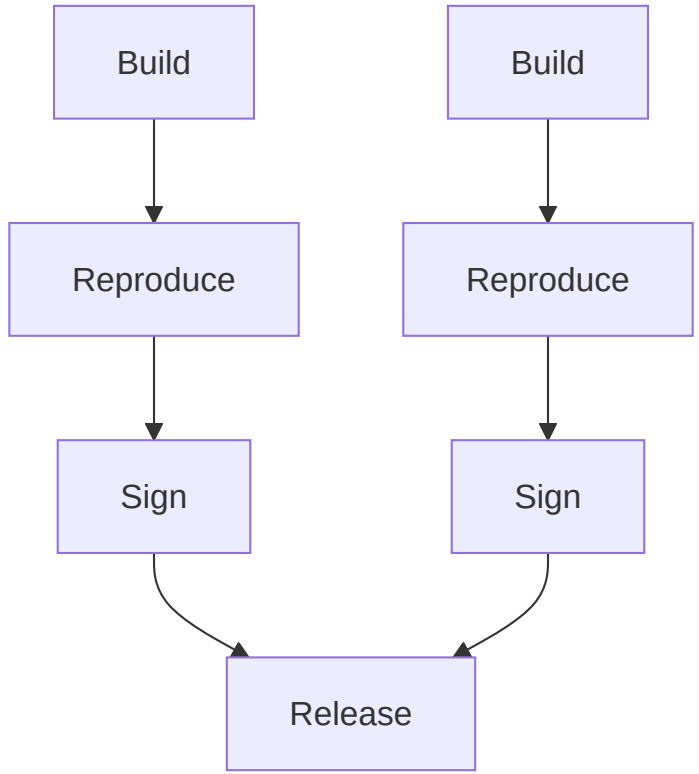
Reproducible

- 100% reproducible with no compromises
- Must be easy and doable at any time: Clone our repo and run "make"
- We disagree with the metrics most distros use for reproducibility
- Requiring a non-reproducible binary to acheive reproduction, is cheating
- We build -everything- from 0. Rust, random firmware blobs in qemu. Everything.

(In fairness, ignoring desktop use cases has made these goals a -lot- easier.)

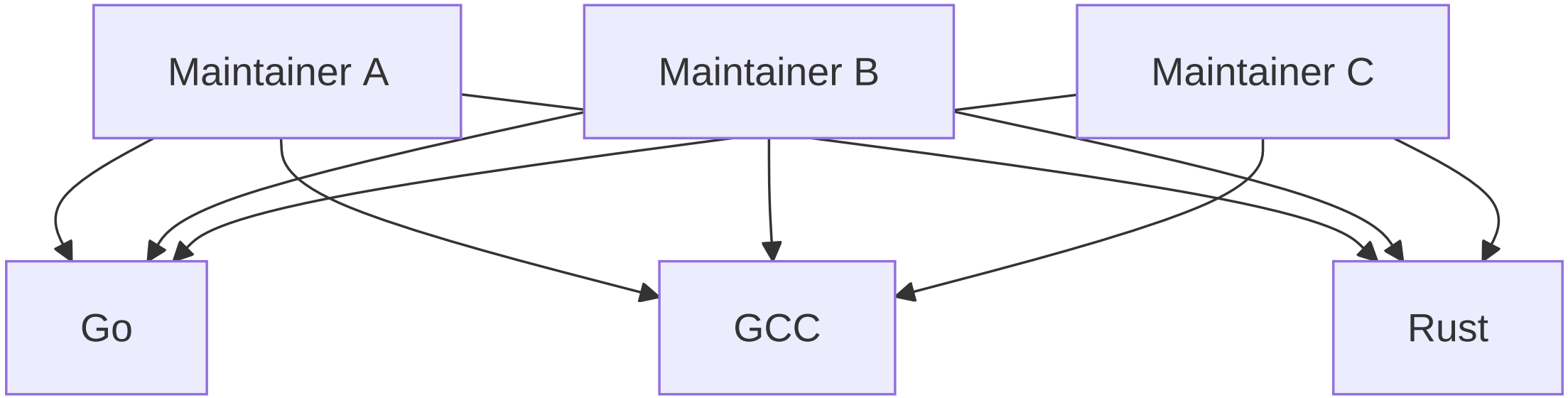
Multi-signed

- Centralized Trust is dangerous
- Distributed Trust is maybe even more dangerous
- We went with Decentralized trust: trust a quorum, not any individual.
- Sign every commit with a hardware token
- Sign every review with a hardware token
- Sign every reproductions on diverse systems with a hardware token
- Reproduce and submit your own signatures for the whole tree at any time!



Decentralized Trust

Multiple maintainers can each sign individual images, so OCI runtime policies can enforce *multiple* signatures by maintainers to ensure no individual maintainer could have tampered with an image.



Distro	Trust Model	OCI	Packaging	Bootstrapped	Reproducible
Stagex	Decentralized	Native	Declarative	Yes	Yes
Guix	Distributed	Exported	Declarative	Yes	Mostly
Debian	Distributed	Published	Imperative	No	Mostly
Arch	Distributed	Published	Imperative	No	Mostly
Nix	Centralized	Exported	Declarative	Partial	Mostly
Yocto	Centralized	Exported	Source Only	No	Mostly
Buildroot	Centralized	Exported	Source Only	No	Mostly
Alpine	Centralized	Published	Imperative	No	No
Fedora	Centralized	Published	Imperative	No	No

TL;DR: Give everyone a shorter path to better supply chain security

A reproducible full source bootstrapped Containerfile for a rust project:

```
ARG RUST_VERSION
FROM stagex/pallet-rust@${RUST_VERSION} AS build
ADD src/ .
ENV RUSTFLAGS='-C target-feature=+crt-static'
RUN cargo build --release --target-dir .

FROM stagex/filesystem AS package
COPY --from=build hello /usr/bin/hello
CMD ["/usr/bin/hello"]
```

Common toolchain dependencies

StageX comes with developer-loved tooling and languages, such as:

- `rust`
- `go`
- `python`
- `curl`
- `git`

If you are interested in additionally software being added feel free to open a PR or let us know what you would like to see added.

Key Takeaways

- StageX packages the software you're already using, securely.
- By leveraging OCI (Docker, podman, etc), we avoid mixing package managers and build contexts.
- Your software, at every point in the bootstrapped toolchain, can all be built deterministically.

What's Next?

- Packaging more software and updating existing software faster
- Adding additional container runtimes like Podman and Kaniko
- Adding additional chip architecture support such as ARM and RISC-V
- Crosschains: Cross compiler packages so you can target any arch from your native arch
- Mirrors: Much more maintainable source/mirror management in review
- Pallets: sets of deps to be imported/locked with one line
- Boxes: Easily output stagex builds as bootable ISO images, firmware, packages for other distros, etc

Links

Matrix Chat: #stagex:matrix.org

Git Repo: https://codeberg.org/stagex/stagex

Slides: https://git.distrust.co/public/presentations

ReprOS: https://codeberg.org/stagex/repros

AirgapOS: https://git.distrust.co/public/airgap

EnclaveOS: https://git.distrust.co/public/enclaveos

Keyfork: https://git.distrust.co/public/keyfork

Big thank you to sponsors who have supported the development of this project:

Turnkey, Distrust, Mysten Labs