Rootless, Reproducible & Hermetic

Secure container build showdown

Andrew Martin and Pi Unnerup

@sublimino and @controlplaneio

controlplane

I’m:

- Andy
- Dev-like
- Sec-ish
- Ops-y

controlplane

controlplane

1781

- Container image build security
- Rootlessness, reproducibility, and hermeticism
- Attack and defense for an OCI image build
- Comparison of present and future tooling
- Securing untrusted builds

The Root of All Evil

The Root of All Evil is

unnecessarily

running

processes

as

root

controlplane

Running Containerised *Processes* as root

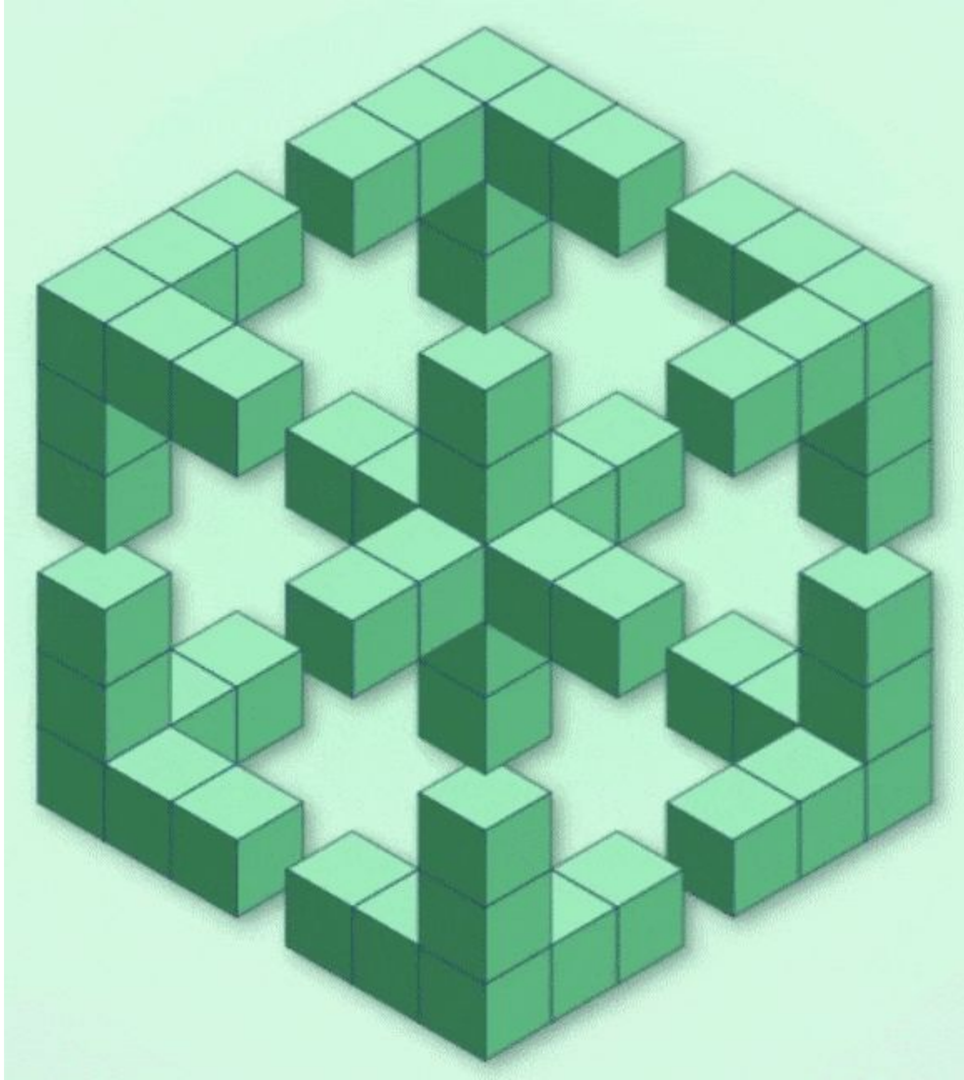
- Not a “vulnerability”
- Handy basis to pivot - many operations gated by root/capabilities
- Dangerous when any namespace is disabled (see `--privileged`)
  - E.g. can permit reconfiguration of network namespace’s NICs
  - Or traversal of host-mounted paths
- Prerequisite for many attacks
  - See runC container breakout (CVE-2019-5736)
- No protection from accidental misconfiguration of other security features
  - Access to /proc, /sys, or /dev may be...terminal
- Attackers **want root** inside a compromised container

controlplane

Running Container *Runtimes* as root

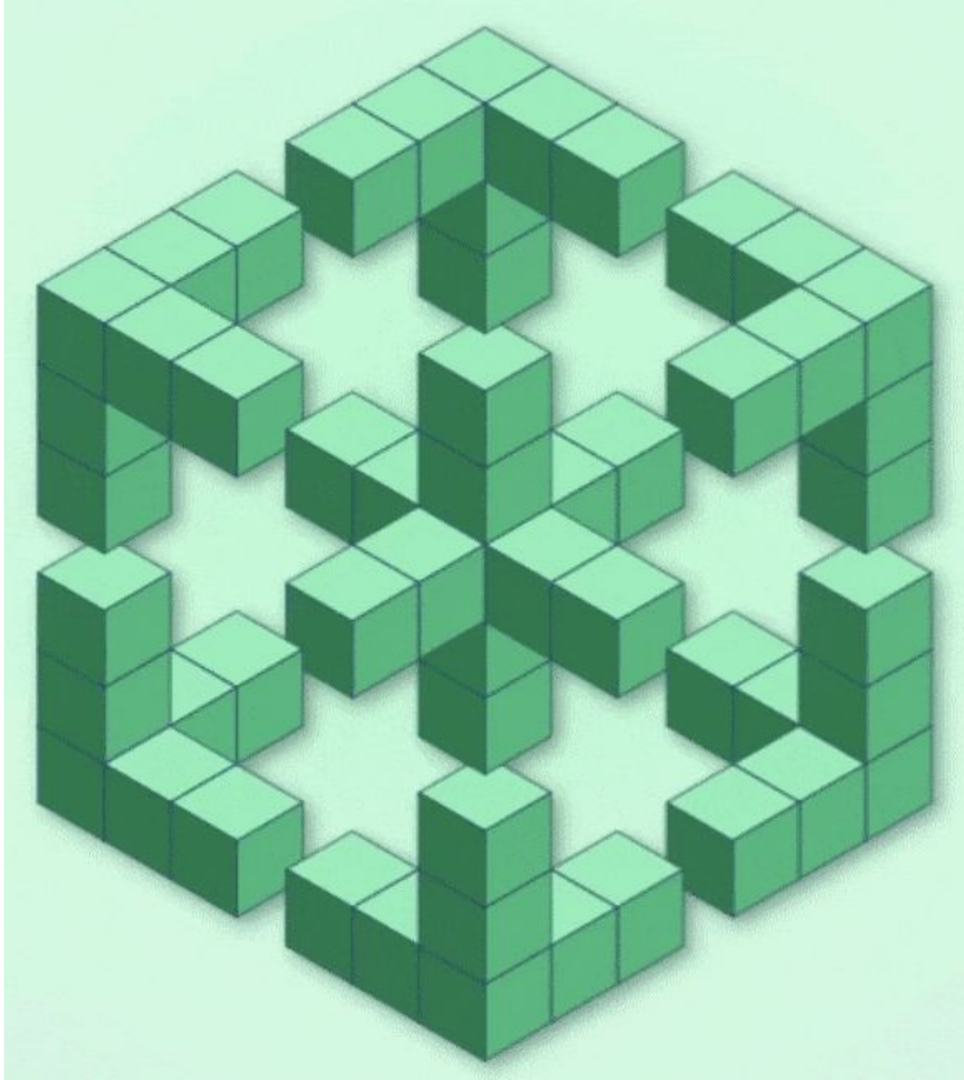
Rootless container build?

- No id 0 process on host (e.g. container runtime)
- No root capabilities for RUN commands in build namespaces



Rootless container build?

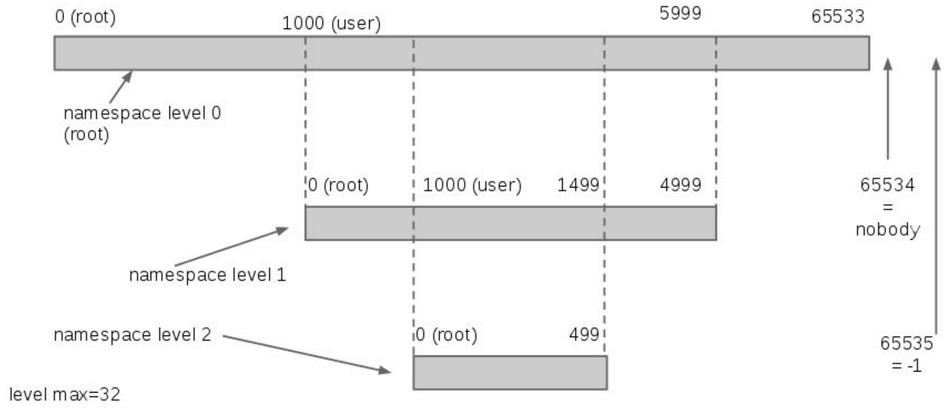
- No id 0 process on host (e.g. container runtime)
- No root capabilities for RUN commands in build namespaces
- Protects from a class of privilege escalation attacks
- User namespaces go a long way to fixing this



User Namespaces

- V1 still problematic
- Guest to host UIDs remapping
- Guest root is UID 0 in the UserNS with full capabilities, with restrictions
    - Inaccessible files, inability to insert kernel modules, rebooting disabled, ...
    - Root can be mapped to any user on the host
- Unprivileged users can only map their UID/GID (to itself or root)
- ShiftFS incoming

Hierarchy of user namespaces



https://endocode.com/blog/2016/01/22/linux-containers-and-user-namespaces/

h/t Aleksa Sarai and Akihiro Suda: The State of Rootless Containers

controlplane

Reproducibility: Attacking Compiler Builds

- Xcodeghost
- Win32 Induc(tion) virus

controlplane

Reproducible Builds

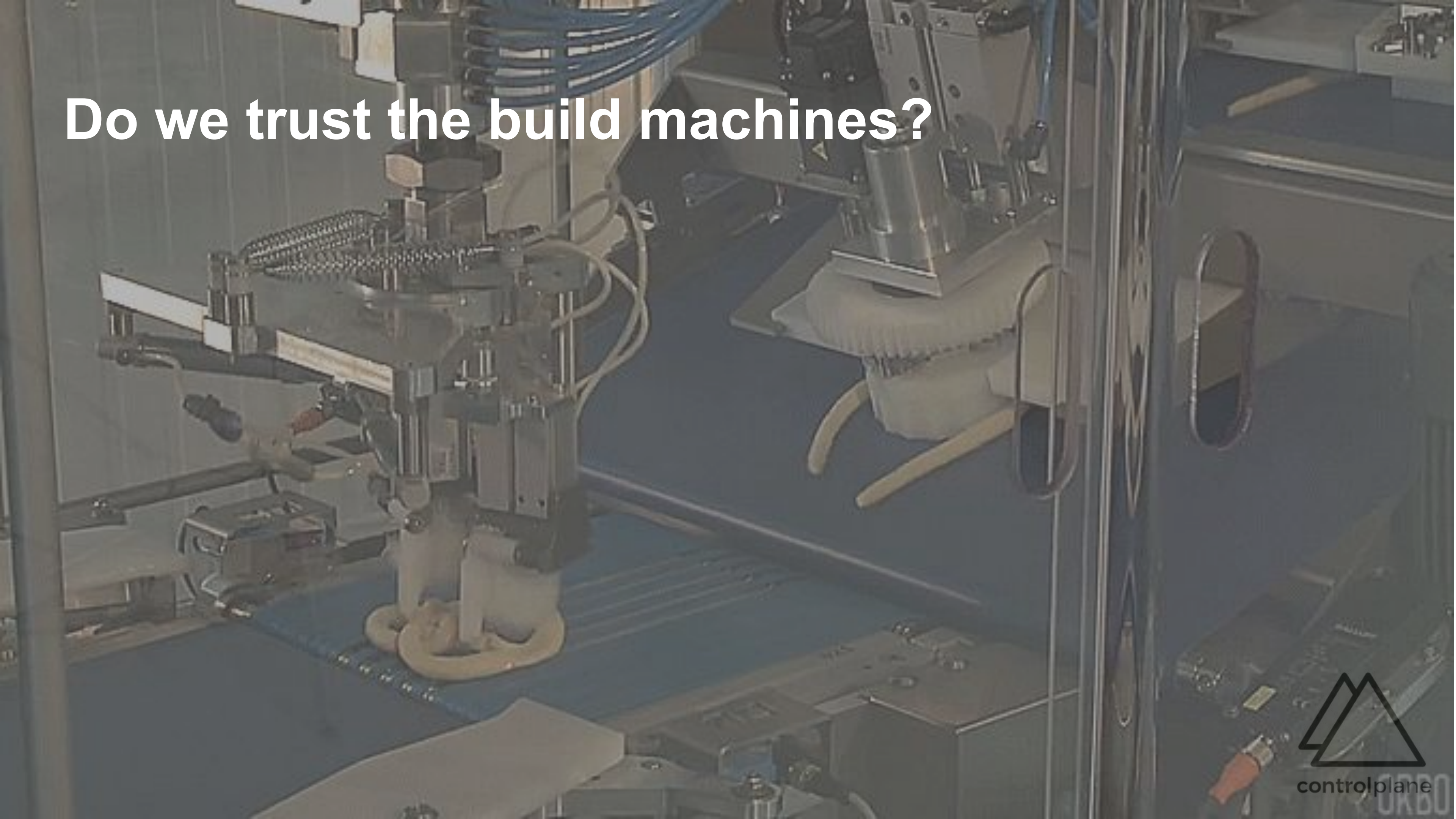
- Reflections on Trusting Trust (Thompson)
  Communications of the ACM, 27:8, Aug 1984
- Debian Reproducible Builds (SoC '19)
- in-toto build attestations
- Reproducible Builds Project

Contributing Projects: Arch Linux, Baserock, Bitcoin, coreboot, Debian, ElectroBSD, F-Droid, FreeBSD, Fedora, GNU Guix, Monero, NetBSD, NixOS, OpenEmbedded, openSUSE, OpenWrt, Qubes OS, Symfony, Tails, Tor Browser, Webconverger, Yocto Project

Trust These Men

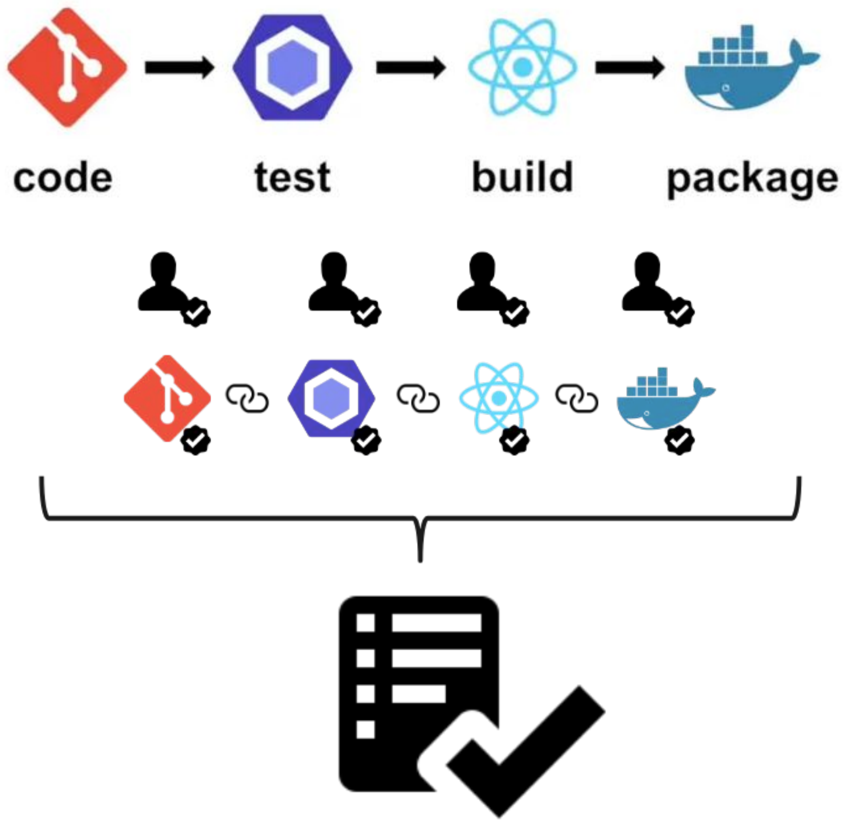
- Reflections on Trusting Trust (Thompson) Communications of the ACM, 27:8, Aug 1984

Do we trust the build machines?



controlplane

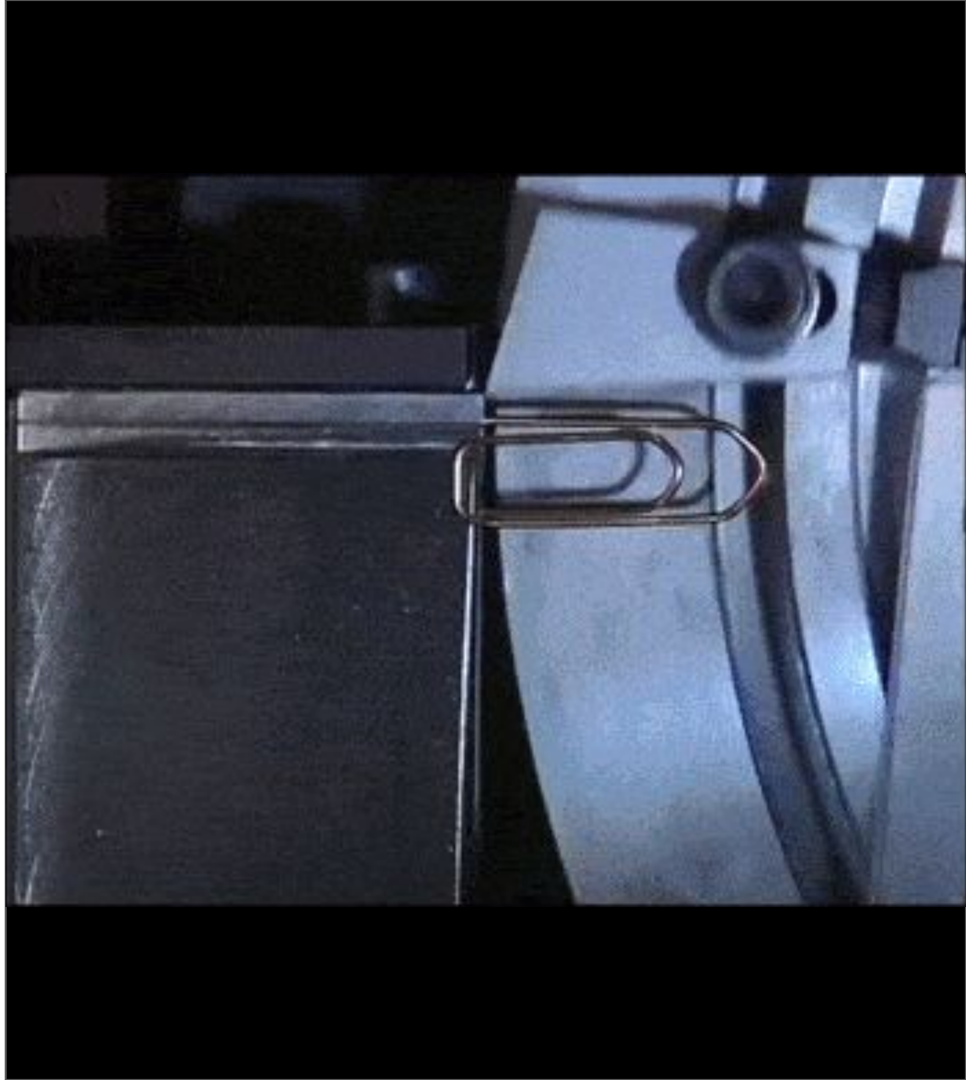
in-toto



controlplane

Reproducible OCI Builds

- Local or pinned dependencies
- No non-deternministic/network calls
- Identical product every time
    - No time-based behaviour or output
    - Identical output ordering
    - Bit-for-bit similarity
- Signable and tamper-proof output



OCI v2

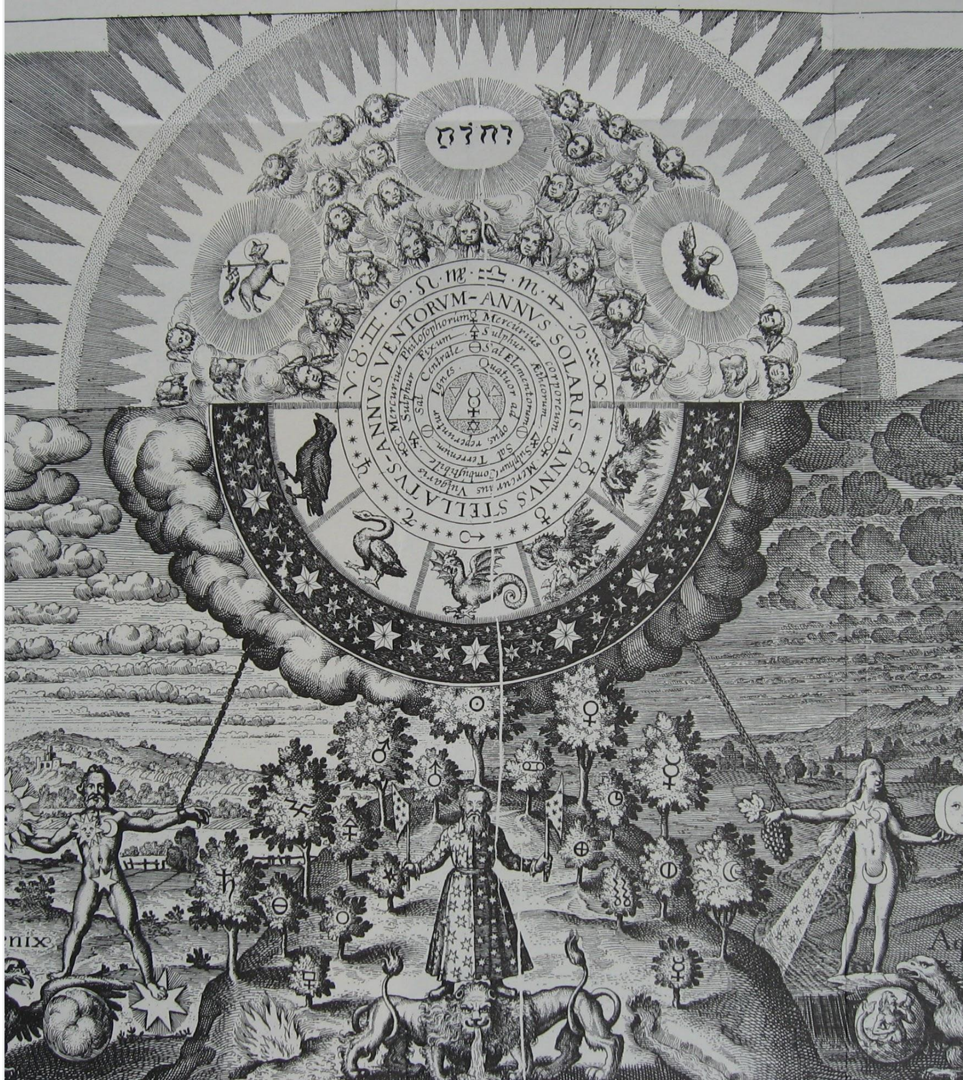
- Issues with Golang’s GNU Tar implementation
  - OCI spec includes “bugs and all”
- OCI v2 looking to fix layering, distribution, and reproducibility
  - All in progress
  - ca-sync a possible solution
- Encrypted image proposal in-flight

OPEN CONTAINER INITIATIVE

controlplane

Hermetic Builds

- No, not Hermetism...



Hermetic Builds

- No, not Hermetism...
- Hermetic!
  - no impact on other builds
  - doesn't rely on external inputs
  - able to run untrusted Dockerfiles?

Attacking Container Image Builds

- Malicious commands in RUN directive can attack host
  - Host's non-loopback network ports/services
  - Enumeration of other network entities (cloud provider, build infrastructure, network routes to production)
- Malicious FROM image has access to build secrets
- Malicious image has ONBUILD directive
- Docker-in-docker can lead to host breakout
- 0days, kernel bugs, network attack surface

controlplane

Defending

- Prevent network egress
- Isolate from the host's kernel
- Execute RUN commands a non-root user in container filesystem
- Run build process as a non-root user
  - or in a user namespace
- Share nothing non-essential

controlplane

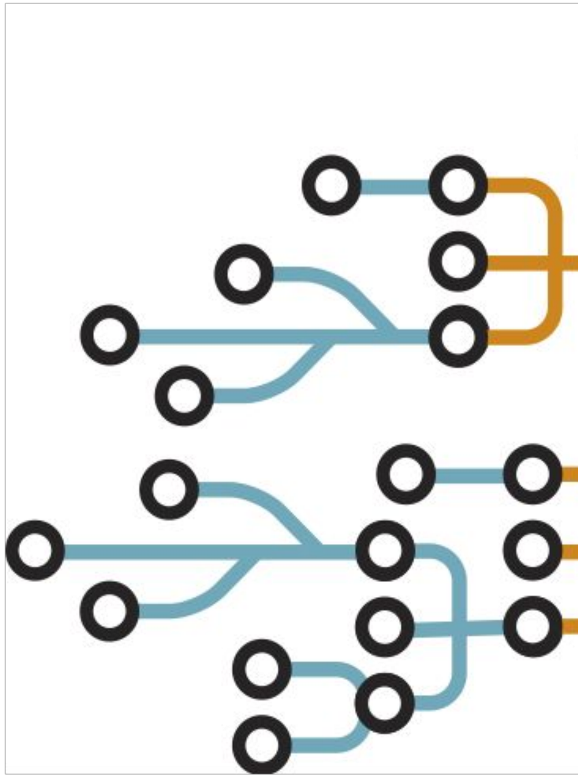
Where to Get Burned -->

- User rootlessnesses
    - Build tool (e.g. docker build, daemon)
    - RUN command invocation during build
- Host hermeticism
    - Kernel
    - Filesystem, sockets/IPC
    - Other containers/builds
- Network hermeticism
    - Host
    - Other reachable hosts
    - Other builds
- Dockerfile reproducibility, hermeticism
    - Possible malicious FROM or RUN

Secure Container Build Showdown

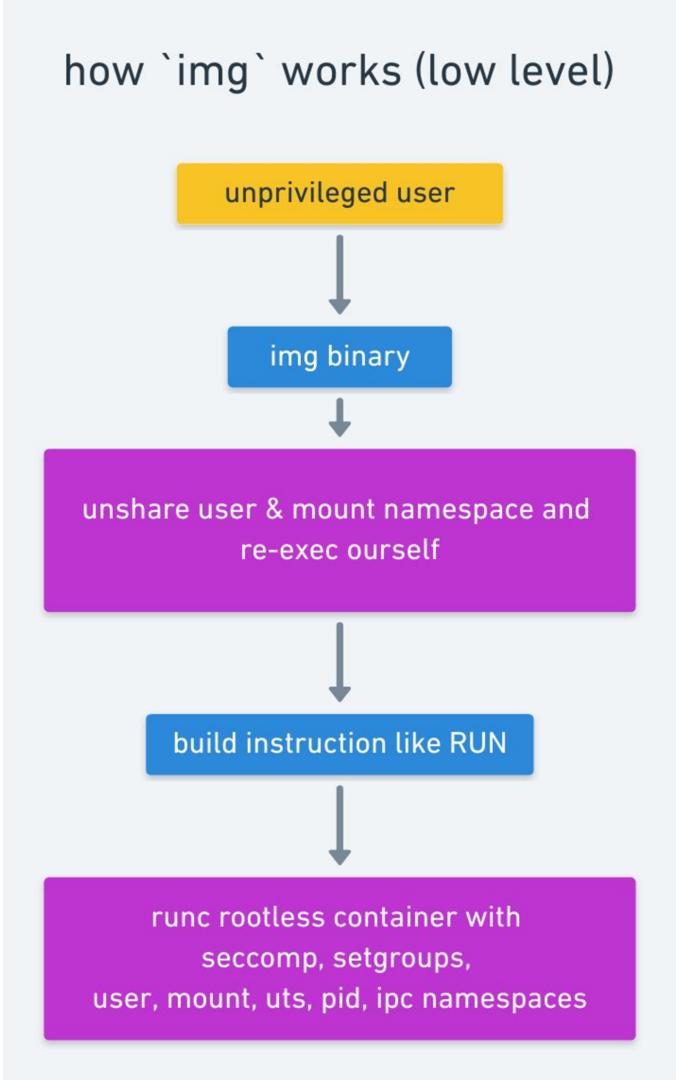
Buildkit

- rootless (optional)
- daemon-less (optional)
- Can run rootless (currently experimental)
  - UserNS, rootlesskit, SUID binary for apt
- Integrated with Docker from v18.06
  - Can run as standalone daemon
- Rootless BuildKit can be executed inside Docker and Kubernetes
  - Thanks to ProcMount in securityContext / PodSecurityPolicy
- Possible entitlement model (inspired by Moby Entitlements)?
  - Fine-grained permissions around RUN commands in build
- Can parallelize step execution via Low-level Builder (LLB) DAG



Img

- rootless
- daemon-less
- Consumes BuildKit as a library
  - Still requires SUID binaries for apt
    - `newuidmap(1)/newgidmap(1)`
    - prepares SUBUIDs/SUBGIDs
- Uses unprivileged mounting
  - Hardened build with seccomp
  - User namespaces enabled


how `img` works (low level)
unprivileged user
img binary
unshare user & mount namespace and re-exec ourself
build instruction like RUN
runc rootless container with seccomp, setgroups, user, mount, uts, pid, ipc namespaces

LXC

- Unprivileged since 2013
- Lower-level components than dev-facing Docker
  - Powered Docker (pre-libcontainer)
- Supports OCI via script
  - Runs fully rootless
- Otherwise LXD requires daemon to be run as root
- Supports Dockerfile-like builds with https://github.com/hverr/lxdfile

controlplane

umoci

- rootless (and SUID-less, but slower than SUID approach)
- daemon-less
- umoci modifies Open Container images
  - Wraps runC (@lordcyphar also a runC maintainer)
    - runC rootless support: 1.0.0-rc4 (March 2017)
  - Emulates `CAP_DAC_OVERRIDE` with recursive chmod
    - And some other syscalls, using `ptrace`
- Does not require setting up SUBUIDs/SUBGIDs (which require SUID binary) for unpacking archives with multiple UIDs/GIDs
  - Uses `user.rootlesscontainers` xattr instead of `chown(2)`
  - No kernel namespacing, VFS-based

controlplane

Kaniko

- rootless
- daemon-less (but runs inside a container without `--privileged`)
- Multiple build modes
    - Kubernetes
    - gVisor
    - Google Cloud Build
    - Docker
- Backend for Knative Build
- Executes RUN commands in the same namespace/rootfs as Kaniko itself
    - Commands are run as root inside the container
    - Does not isolate `/secret` mount from the FROM image

controlplane

buildah

- rootless (optional)
- daemon-less
- Multiple isolation modes via `--isolation=ISOLATION`
  - OCI (default)
  - Rootless (with UserNS)
  - Chroot (root in unprivileged container)
- Uses Slip4netns for networking

controlplane

makisu

- rootless
- daemon-less
- Inspired by Bazel, but addresses lack of RUN
- Similar model to Kaniko
    - Addresses performance for “large images”
    - Doesn’t require nested containers
- https://github.com/uber/makisu

controlplane

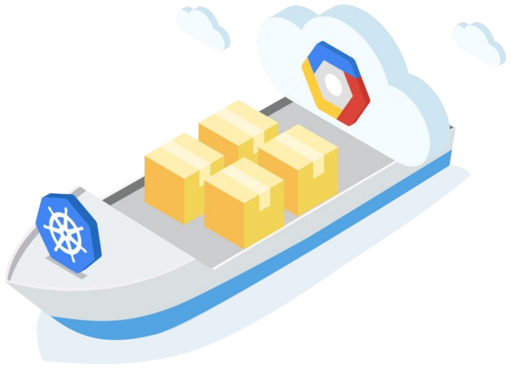
Bazel

- rootless
- daemon-less
- Ultimate hermetism, reproducibility
  - Because RUN command don't exist
- Can build Java, C++, Android, iOS, Go, and more
- Has https://github.com/bazelbuild/rules_docker for Docker builds
- Minimum usability

controlplane

Google Cloud Build

- Well isolated, rootlessness not so worrisome
- Hermetic with regards to other builds
    - But not the internet
- Reliant on cloud provider’s security model

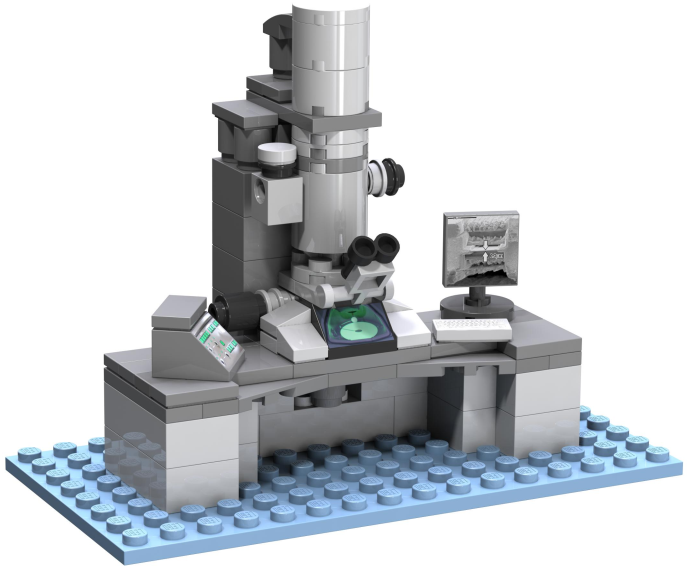


controlplane

Fulfilment of requirements?

Build Tools Analysis Summary

- Rootless
  - Everybody, but implementation-specific caveats abound
- Reproducible
  - No build tools are un-reproducible by design
  - But output is a function of RUN behaviour
- Hermetic
  - Varying degrees, nothing absolute



controlplane

But Why Choose One: CBI Edition

- Container Builder Interface for Kubernetes
- Provides a vendor-neutral abstraction for building and pushing container images in Kubernetes
- Supports
  - Docker, BuildKit, Buildah, kaniko, img, Google Cloud Container Builder, Azure Container Registry Build, OpenShift Source-to-Image...
- https://github.com/containerbuilding/cbi

controlplane

Untrusted Image Builds?

controlplane

Untrusted Image Builds

- Are scary
- Will “fix everything”
- Are almost ready
  - Requires user namespace, a hypervisor, or root emulation
  - Kaniko with gVisor
  - Hosted tooling
  - ...
- BuildKit probably closest
  - Used in OpenFaaS Cloud for user-supplied builds
  - But still templated, not fully untrusted
- If in doubt: **isolate with a VM**

The Future

- Rootless runC
- Usernetes
- ShiftFS
- Hypervisor proliferation
- Unikernels?

Exciting Times Ahead

http://explosm.net/shorts/66/the-painting

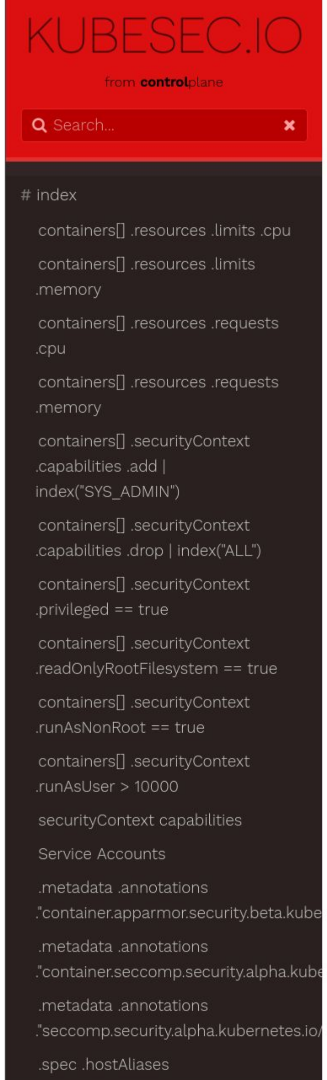
controlplane

https://kubesec.io

github.com/controlplaneio/kubesec

Security risk Analysis for Kubernetes Resources

Now with 100% more Open Sauce!



KUBESEC.IO

from **control**plane

Search...

# index

containers[] .resources .limits .cpu

containers[] .resources .limits .memory

containers[] .resources .requests .cpu

containers[] .resources .requests .memory

containers[] .securityContext .capabilities .add | index("SYS_ADMIN")

containers[] .securityContext .capabilities .drop | index("ALL")

containers[] .securityContext .privileged == true

containers[] .securityContext .readOnlyRootFilesystem == true

containers[] .securityContext .runAsNonRoot == true

containers[] .securityContext .runAsUser > 10000

securityContext capabilities

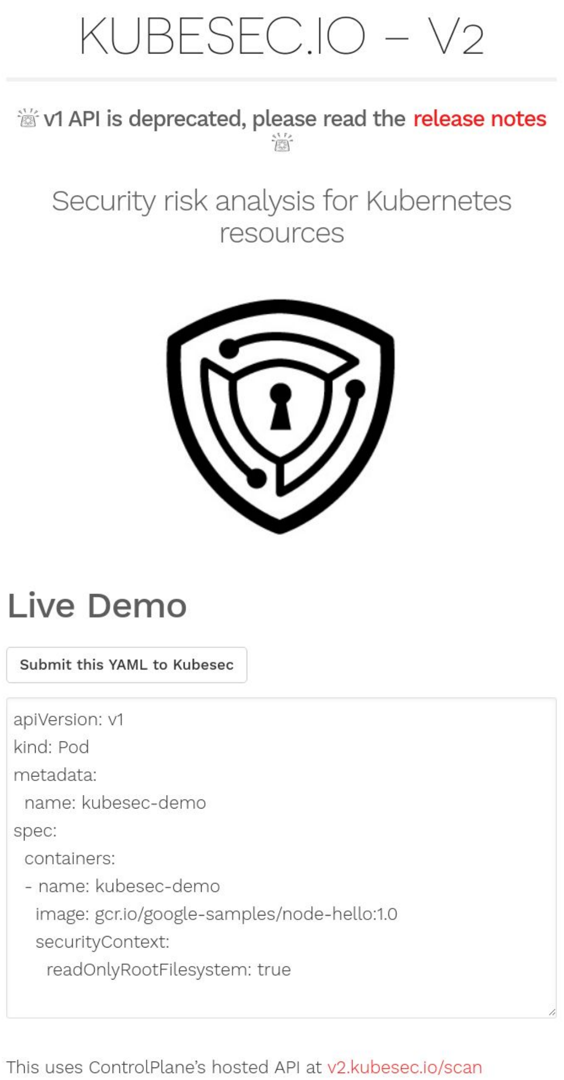
Service Accounts

.metadata .annotations ."container.apparmor.security.beta.kube

.metadata .annotations ."container.seccomp.security.alpha.kube

.metadata .annotations ."seccomp.security.alpha.kubernetes.io/

.spec .hostAliases



KUBESEC.IO – V2

v1 API is deprecated, please read the release notes

Security risk analysis for Kubernetes resources

Live Demo

Submit this YAML to Kubesec

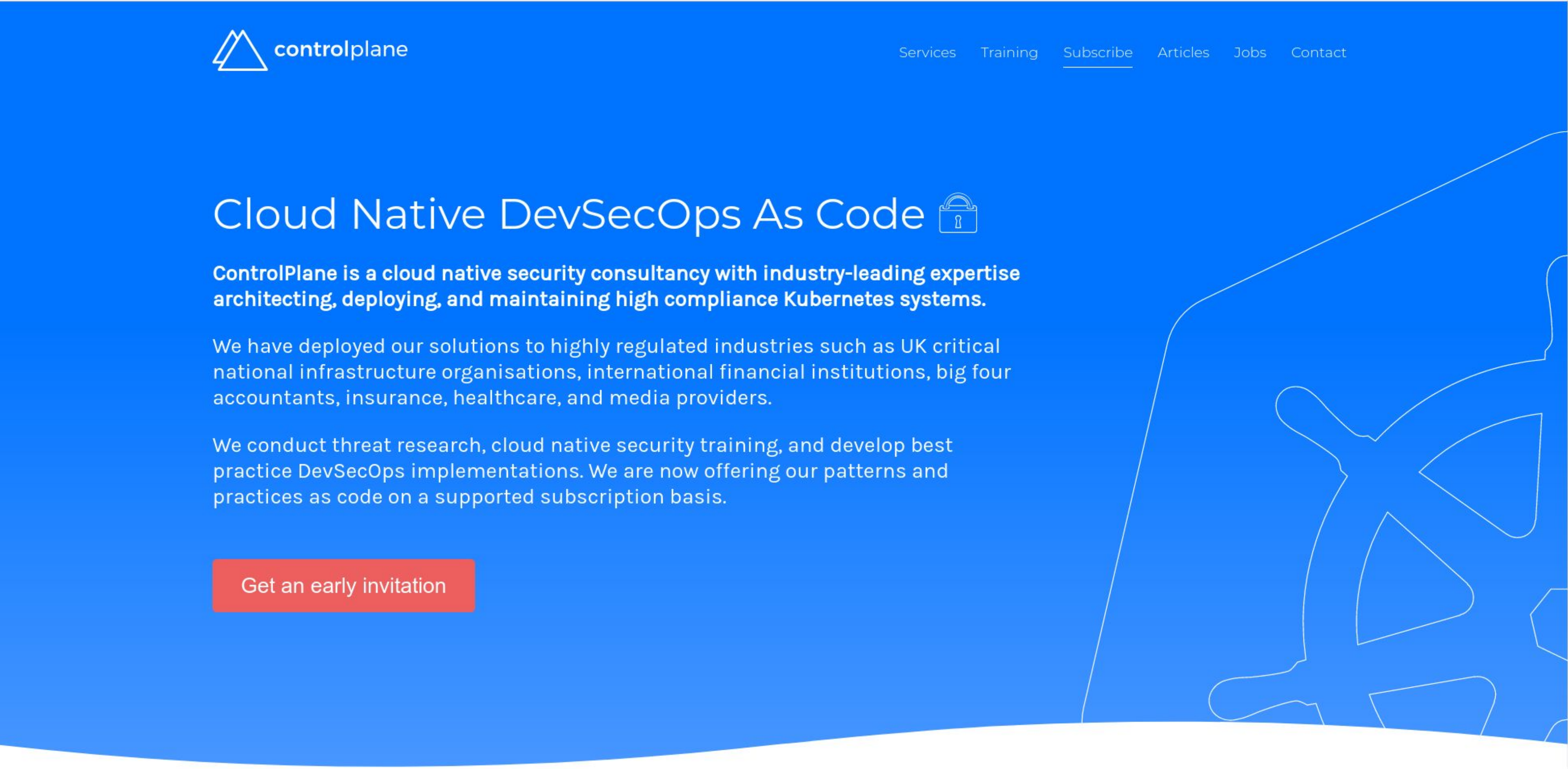
```
apiVersion: v1
kind: Pod
metadata:
  name: kubesec-demo
spec:
  containers:
  - name: kubesec-demo
    image: gcr.io/google-samples/node-hello:1.0
    securityContext:
      readOnlyRootFilesystem: true
```

This uses ControlPlane's hosted API at v2.kubesec.io/scan

kubesec.io - example insecure pod

```
[
  {
    "object": "Pod/kubesec-demo.default",
    "valid": true,
    "message": "Passed with a score of 1 points",
    "score": 1,
    "scoring": {
      "advise": [
        {
          "selector": "containers[] .securityContext .capabilities .drop",
          "reason": "Reducing kernel capabilities available to a container limits its attack
surface"
        },
        {
          "selector": ".spec .serviceAccountName",
          "reason": "Service accounts restrict Kubernetes API access and should be configured
with least privilege"
        },
        {
          "selector": "containers[] .resources .requests .cpu",
          "reason": "Enforcing CPU requests aids a fair balancing of resources across the
cluster"
        },
   ...
```

controlplane



Cloud Native DevSecOps As Code 🔒

ControlPlane is a cloud native security consultancy with industry-leading expertise architecting, deploying, and maintaining high compliance Kubernetes systems.

We have deployed our solutions to highly regulated industries such as UK critical national infrastructure organisations, international financial institutions, big four accountants, insurance, healthcare, and media providers.

We conduct threat research, cloud native security training, and develop best practice DevSecOps implementations. We are now offering our patterns and practices as code on a supported subscription basis.

https://control-plane.io/cnsec

Fin!

With thanks to:

- Aleksa Sarai
- Akihiro Suda
- Christian Brauner

References:

- https://rootlesscontaine.rs/
- https://github.com/rootless-containers/rootlesskit
- https://github.com/AkihiroSuda/buildbench
- https://github.com/rootless-containers/slirp4netns
- https://www.slideshare.net/AkihiroSuda/the-state-of-rootless-containers
- https://events.linuxfoundation.org/wp-content/uploads/2017/11/Comparing-Next-Generation-Container-Image-Building-Tools-OSS-Akihiro-Suda.pdf
- https://in-toto.github.io
- https://kernel.ubuntu.com/git/sforshee/linux.git/log/?h=shiftfs-demo
- https://github.com/rootless-containers/usernetes
- https://nabla-containers.github.io/
- https://github.com/Solo5/solo5

controlplane