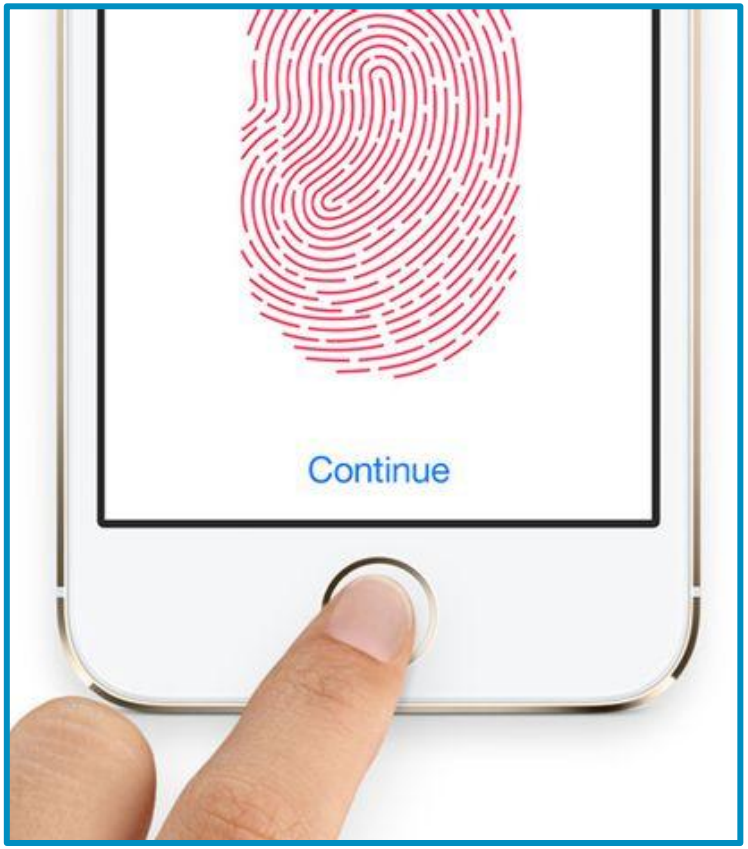
Entitlements

Understandable Container Security Controls

Nassim Eddequiouaq, Justin Cormack
Security at Docker, Inc.
nass@docker.com
justin.cormack@docker.com

How People See Container Security

How People Should See Container Security



The Motto: ***Unusable security is not security.***

Container (and OS) Security

So… what do we have here?

Capabilities
Seccomp
AppArmor
SElinux
Namespaces
Cgroups
No_New_Privs
Image Signing
Network Security
Audit Subsystem
Integrity Measurement Architecture (IMA)
...

“Please make sure your container is securely configured.”

Docker Runtime Security

```
$ docker run --help | grep security-stuff
  --cap-add              Add Linux capabilities
  --cap-drop             Drop Linux capabilities
  --cgroup-parent        Optional parent cgroup for the container
  --device               Add a host device to the container
  --device-cgroup-rule   Add a rule to the cgroup allowed devices list
  --isolation            Container isolation technology
  --network              Connect a container to a network
  --pid                  PID namespace to use
  --privileged           Give extended privileges to this container
  --read-only            Mount the container's root filesystem as read only
  --security-opt         Security Options (Seccomp, AppArmor, ..)
  --sysctl               Sysctl options
  --user                 Username or UID
  --userns               User namespace to use
  --uts                  UTS namespace to use
  [...]
```

Capabilities

Granular Root Permissions

- Slice root privileges into smaller permission chunks
- Can be added or dropped from the whitelist
- ~Granular control:
  - Kernel auditing
  - User permissions bypass
  - File permissions bypass
  - MAC (LSM) permissions bypass
  - [...] x 40

```
105         // CapSetpcap -
106         // * If file capabilities are not supported: grant or r
107         //         capability set to or from any other process.
108         // *  If file capabilities are supported: add any capab
109         //     inheritable set; drop capabilities from the boun
110         CapSetpcap types.Capability = "CAP_SETPCAP"
111
112         // CapSetuid -
113         // * Make arbitrary manipulations of process UIDs
114         // * forge UID when passing socket credentials via UNIX
115         // * write a user ID mapping in a user namespace
116         CapSetuid types.Capability = "CAP_SETUID"
117
118         // CapSysAdmin - Perform administrative operations on t
119         CapSysAdmin types.Capability = "CAP_SYS_ADMIN"
120
121         // CapSysBoot - Use reboot and kexec_load.
122         CapSysBoot types.Capability = "CAP_SYS_BOOT"
123
124         // CapSysChroot - Use chroot.
125         CapSysChroot types.Capability = "CAP_SYS_CHROOT"
```

```
$ docker run --rm -it --cap-drop NET_BIND_SERVICE alpine sh
```

Namespaces

Resources Segmentation

- Partition kernel resources
- Scope of visibility restricted to your own namespace
- Segment:
  - Processes
  - Network stacks, devices, ports, etc..
  - Mount points
  - IPCs
  - Users
  - [...]

```
190         // PIDNamespace for isolating process IDs
191         PIDNamespace LinuxNamespaceType = "pid"
192         // NetworkNamespace for isolating network
193         NetworkNamespace = "network"
194         // MountNamespace for isolating mount poin
195         MountNamespace = "mount"
196         // IPCNamespace for isolating System V IPC
197         IPCNamespace = "ipc"
198         // UTSNamespace for isolating hostname and
199         UTSNamespace = "uts"
200         // UserNamespace for isolating user and gr
201         UserNamespace = "user"
202         // CgroupNamespace for isolating cgroup hi
203         CgroupNamespace = "cgroup"
```

```
$ docker run --rm -it --pid=host alpine sh
```

Seccomp

Syscall Firewall

- Kernel module based on eBPF
- Can allow / block:
  - System calls
  - System call arguments (no deref)
- Can be applied per architecture
- Users can override Docker defaults

```
{
        "names": [
                "clone"
        ],
        "action": "SCMP_ACT_ALLOW",
        "args": [
                {
                        "index": 0,
                        "value": 2080505856,
                        "valueTwo": 0,
                        "op": "SCMP_CMP_MASKED_EQ"
                }
        ],
        "comment": "",
        "includes": {},
        "excludes": {
                "caps": [
                        "CAP_SYS_ADMIN"
                ],
                "arches": [
                        "s390",
                        "s390x"
                ]
        }
},
```

```
$ docker run --rm -it \
  --security-opt seccomp=/path/to/seccomp/profile.json \
  hello-world
```

AppArmor

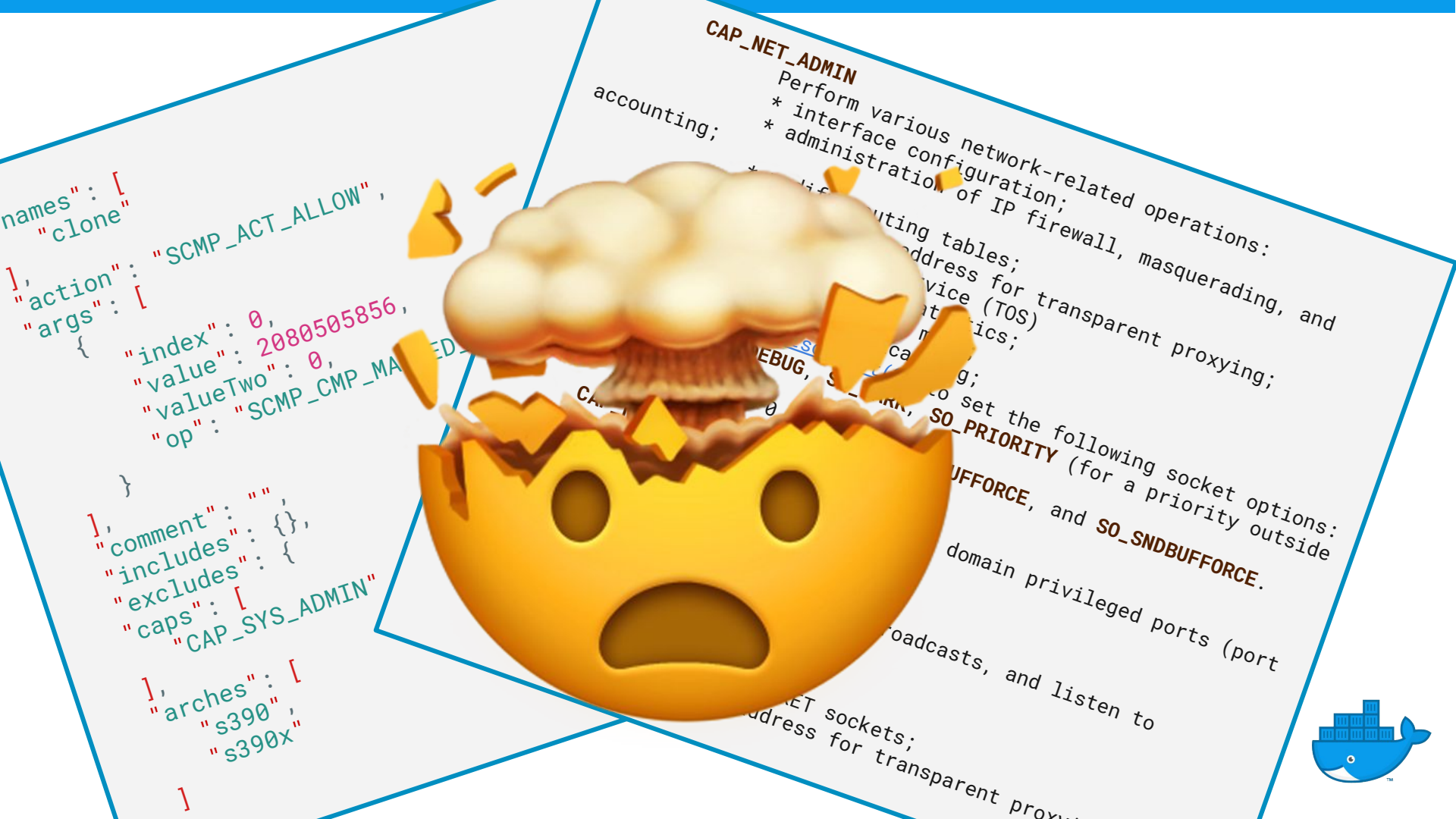
And other LSMs

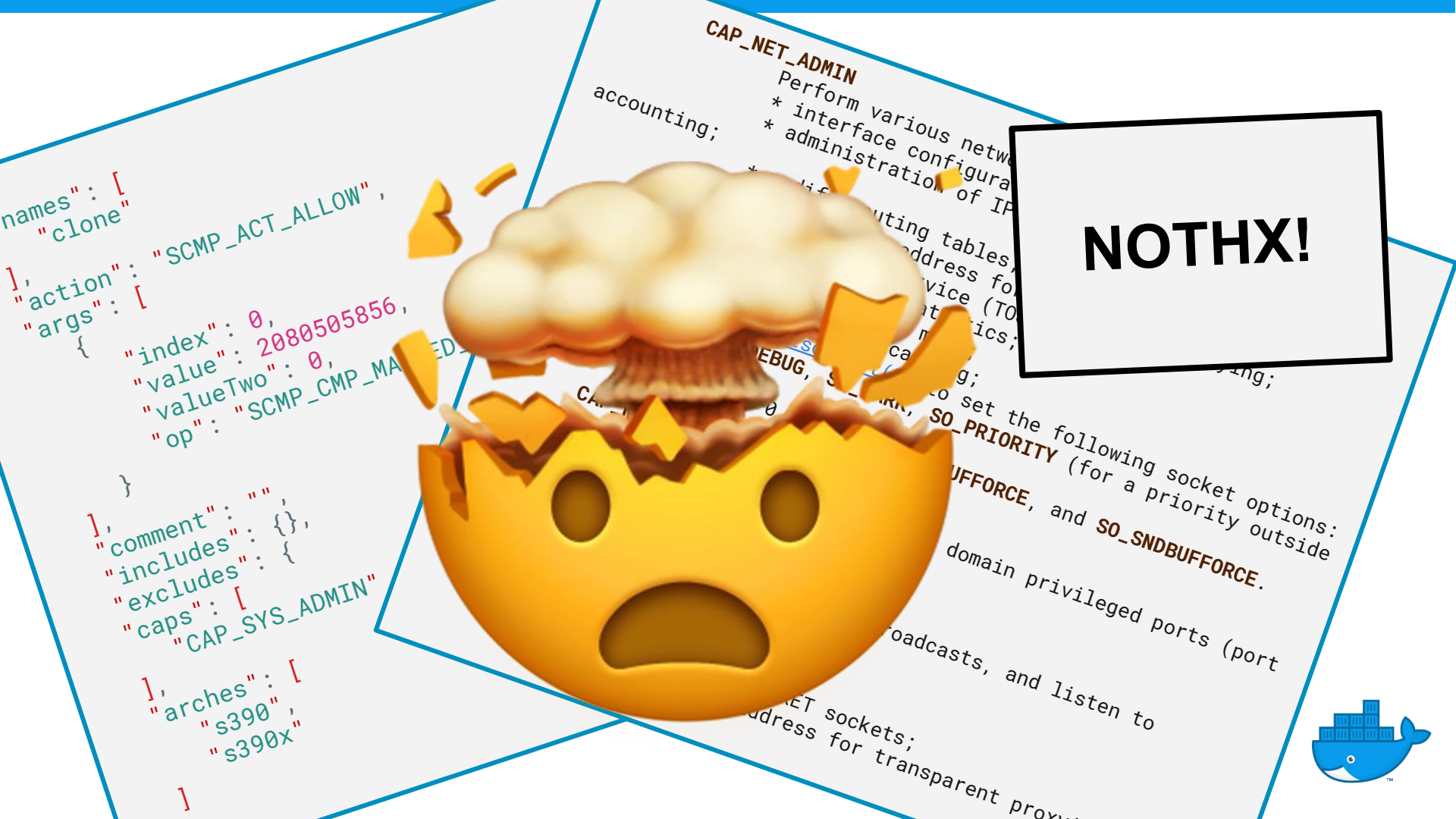
- Linux Security Modules allow additional resource restriction:
  - Files
  - Capabilities
  - Network (network features, protocols, IPv4/6, ...)
  - Tracing, Signals, Mounts…
  - [...]
- AppArmor is a MAC permission system
- Bind access control to programs (path-based)

```
#include <abstractions/openssl>

# needlessly chown'ing the PID
deny capability chown,

capability net_bind_service,
capability setgid,
capability setuid,
capability sys_chroot,
capability sys_resource,

# root trust anchor
owner /var/lib/unbound/root.key* rw,

# root hints from dns-data-root
/usr/share/dns/root.* r,

# non-chrooted paths
/etc/unbound/** r,
owner /etc/unbound/*.key* rw,
audit deny /etc/unbound/unbound_control.{key,pem} rw
audit deny /etc/unbound/unbound_server.key w,

# chrooted paths
/var/lib/unbound/** r,
owner /var/lib/unbound/**/*.key* rw,
audit deny /var/lib/unbound/**/unbound_control.{key,
audit deny /var/lib/unbound/**/unbound_server.key w,
```

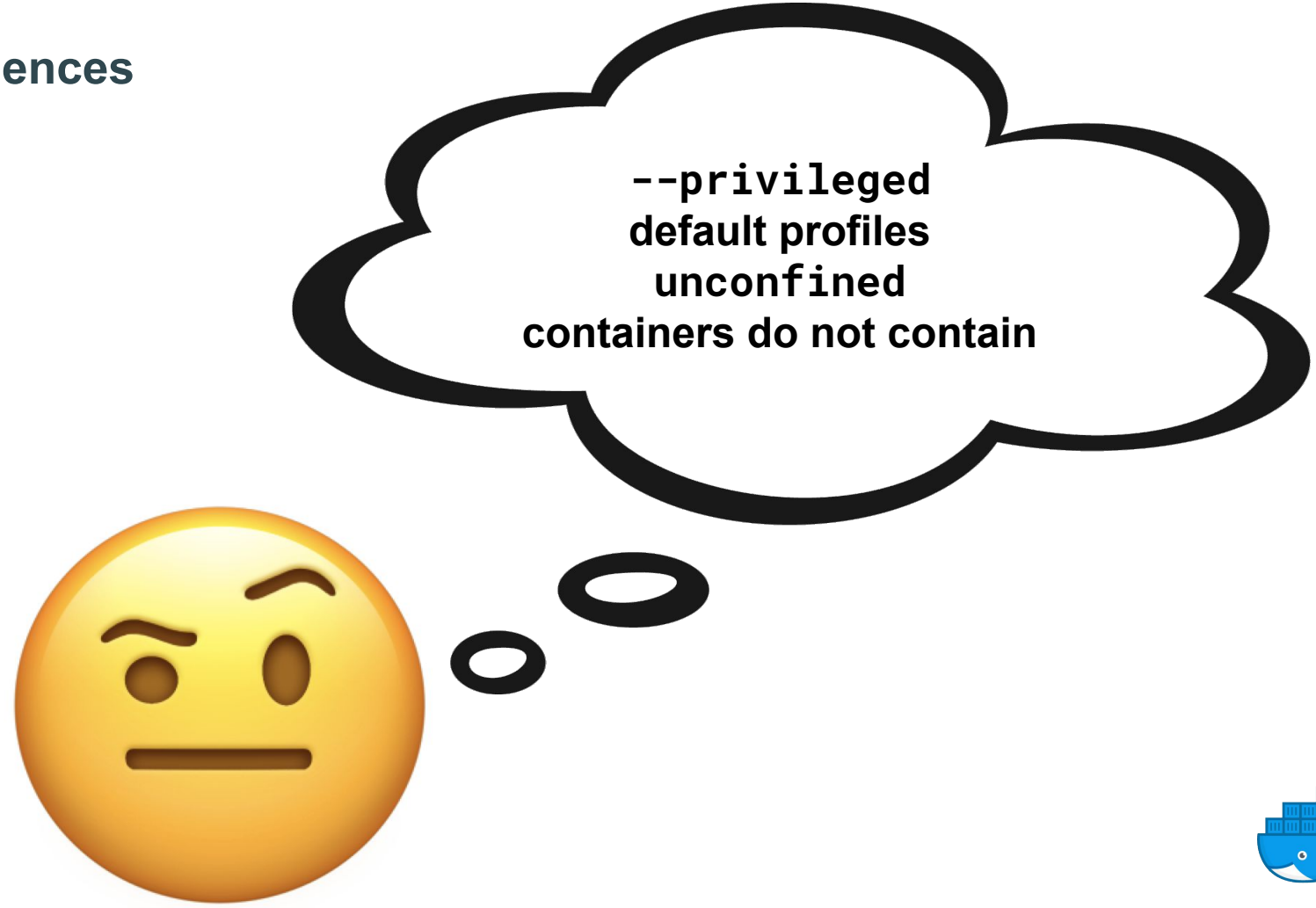
```
$ docker run --rm -it \
  --security-opt apparmor=/path/to/aa/profile.json \
  hello-world
```

“Can you generate the security profile for the webapp?”


names": [
"clone"
],
"action": "SCMP_ACT_ALLOW",
"args": [
{
"index": 0,
"value": 2080505856,
"valueTwo": 0,
"op": "SCMP_CMP_MA
}
],
"comment": "",
"includes": {},
"excludes": {
"caps": [
"CAP_SYS_ADMIN"
],
"arches": [
"s390",
"s390x"
]
CAP_NET_ADMIN
Perform various network-related operations:
* interface configuration;
* administration of IP firewall, masquerading, and
accounting;
uting tables;
ddress for transparent proxying;
vice (TOS)
ics;
g;
EBUG,
0
o set the following socket options:
SO_PRIORITY (for a priority outside
UFFORCE, and SO_SNDBUFFORCE.
domain privileged ports (port
oadcasts, and listen to
ET sockets;
ddress for transparent proxy


names": [
"clone"
],
"action": "SCMP_ACT_ALLOW",
"args": [
{
"index": 0,
"value": 2080505856,
"valueTwo": 0,
"op": "SCMP_CMP_MA
}
],
"comment": "",
"includes": {},
"excludes": {
"caps": [
"CAP_SYS_ADMIN"
],
"arches": [
"s390",
"s390x"
]
CAP_NET_ADMIN
Perform various netw
* interface configura
* administration of IP
accounting;
uting tables,
ddress fo
vice (TO
ics;
EBUG,
g;
0
o set the following socket options:
SO_PRIORITY (for a priority outside
UFFORCE, and SO_SNDBUFFORCE.
domain privileged ports (port
oadcasts, and listen to
ET sockets;
ddress for transparent proxy
NOTHX!

Consequences



Solution: Docker Entitlements

Docker Entitlements Proposal

AKA ”Let’s simplify all this.”

- ☐ network.access=confined
- ☐ network.access=user
- ☐ network.access=proxy
- ☑ network.access=admin

- ☐ security.access=confined
- ☑ security.access=viewer
- ☐ security.access=admin
- ☑ security.fs=read-only

- ☑ host.devices.access=none
- ☐ host.devices.access=admin

...

Docker Entitlements Proposal

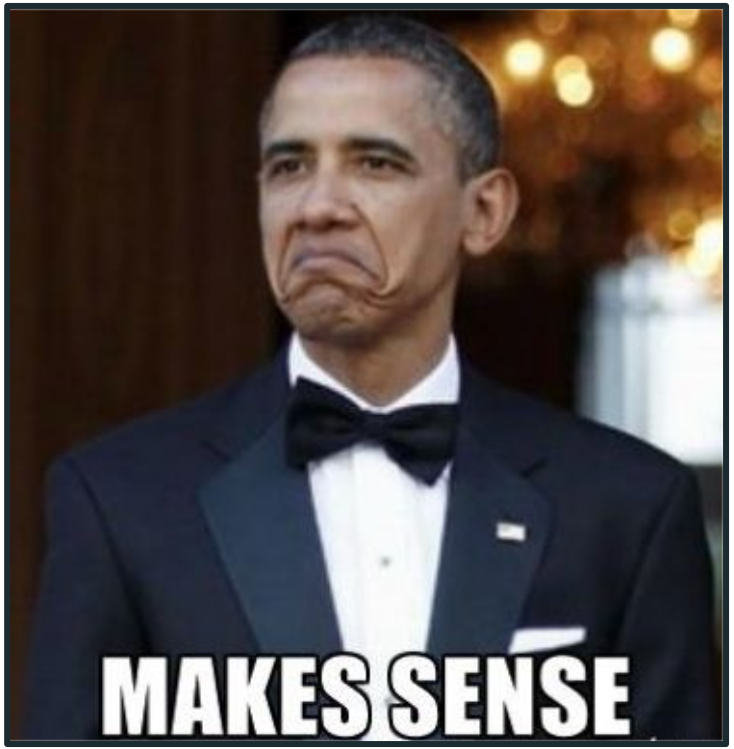
AKA ”Let’s simplify all this.”

- ☐ network.access=confined
- ☐ network.access=user
- ☐ network.access=proxy
- ☑ network.access=admin

- ☐ security.access=confined
- ☑ security.access=viewer
- ☐ security.access=admin
- ☑ security.fs=read-only

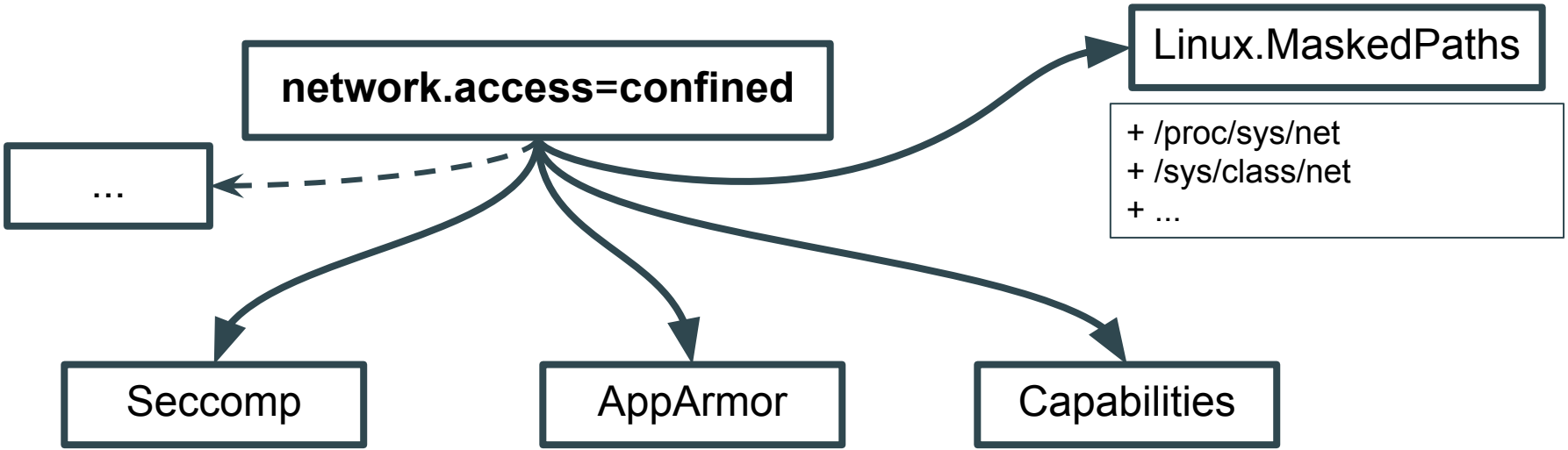
- ☑ host.devices.access=none
- ☐ host.devices.access=admin

...



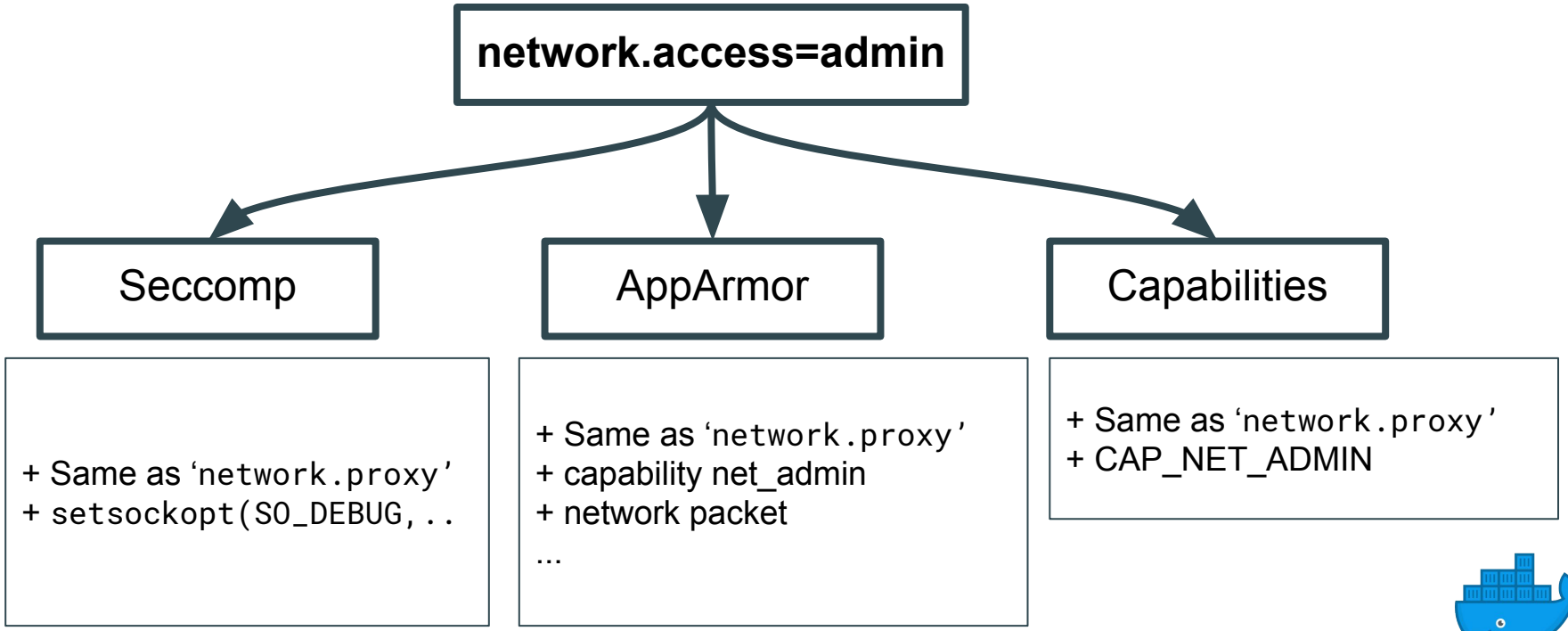
Behind the scenes

AKA ”Don’t worry, I got this.”



Behind the scenes

AKA ”Don’t worry, I got this.”



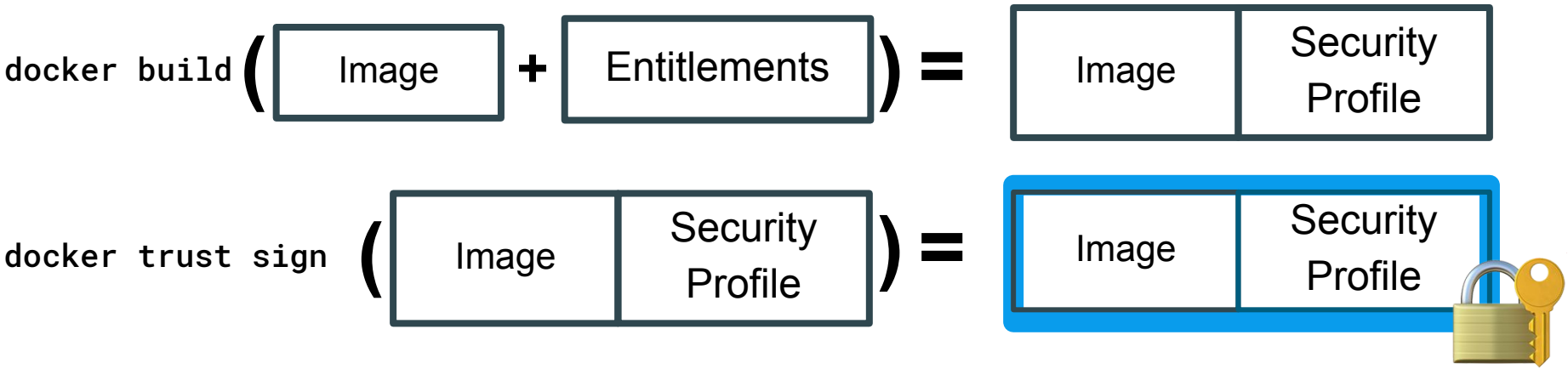
Wait.. But there’s more to do!

Integration with Image Signing

Permissions as part of a Trusted Bundle

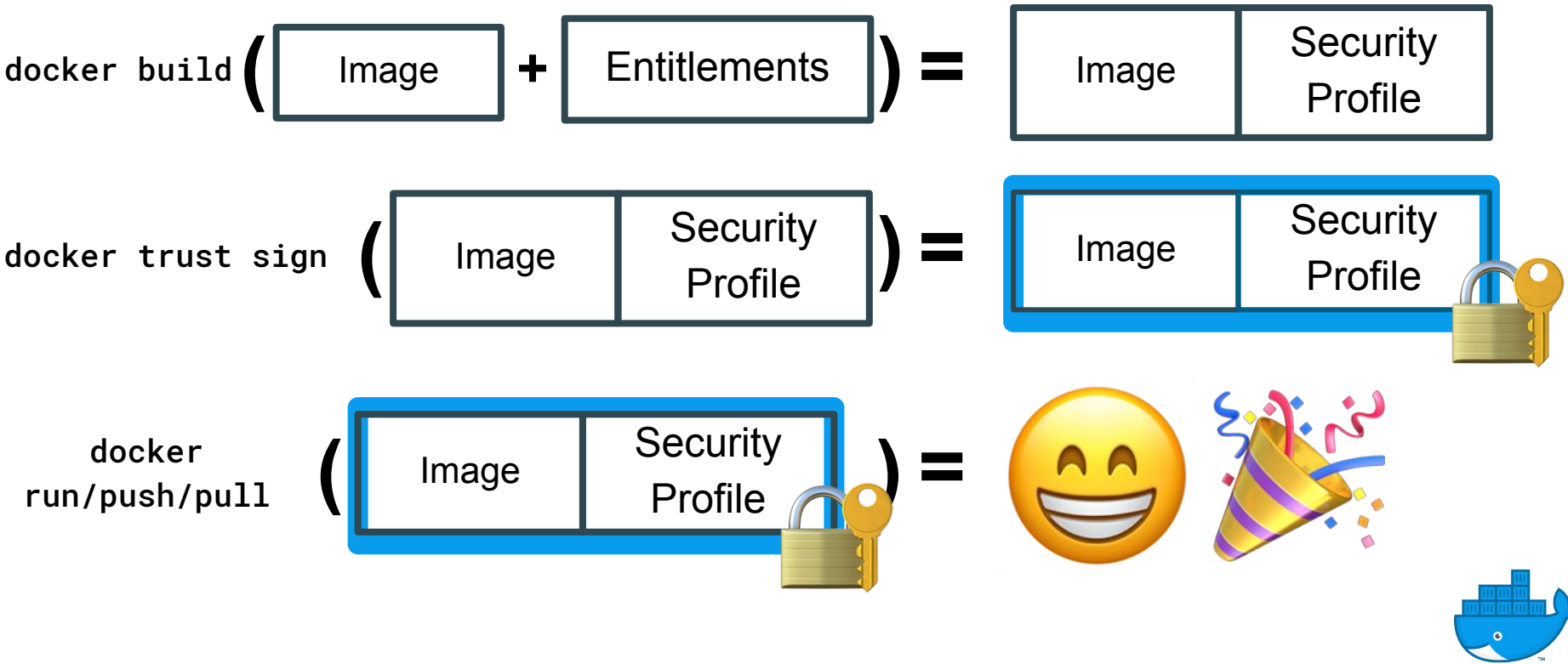
Integration with Image Signing

Permissions as part of a Trusted Bundle



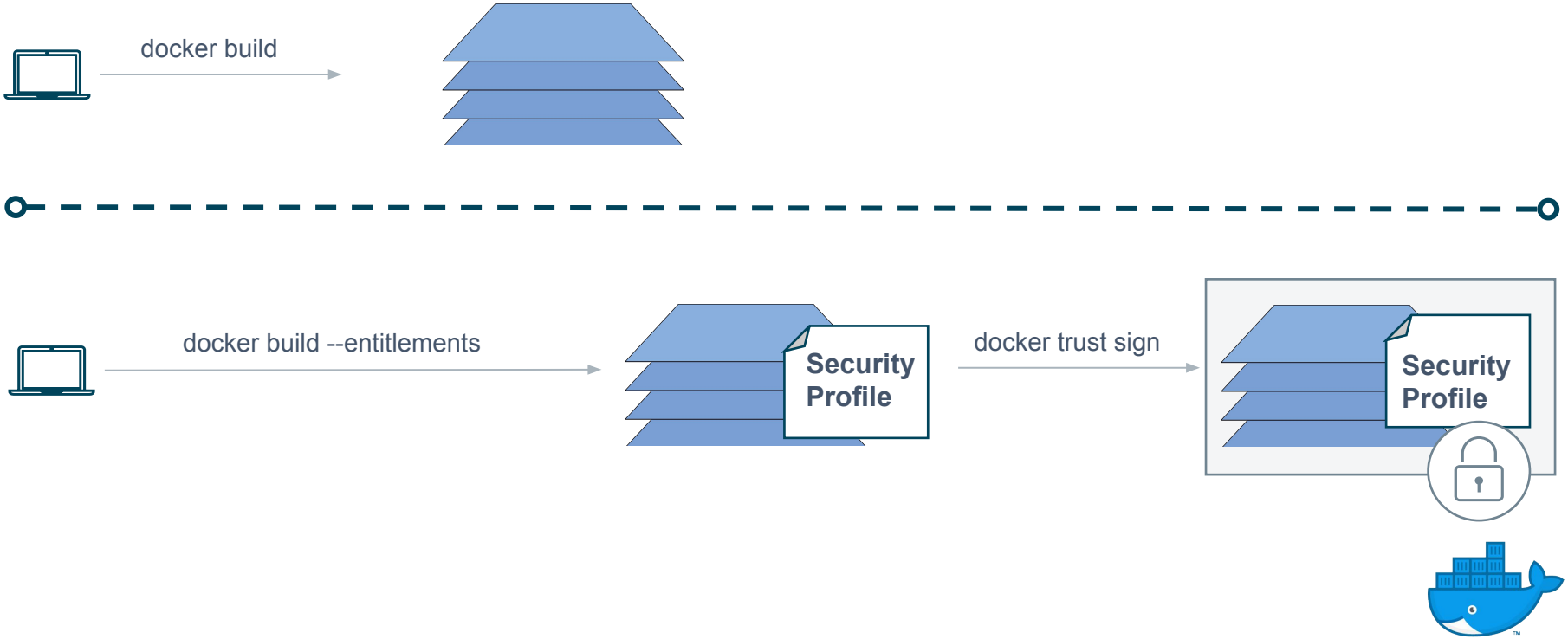
Integration with Image Signing

Permissions as part of a trusted bundle



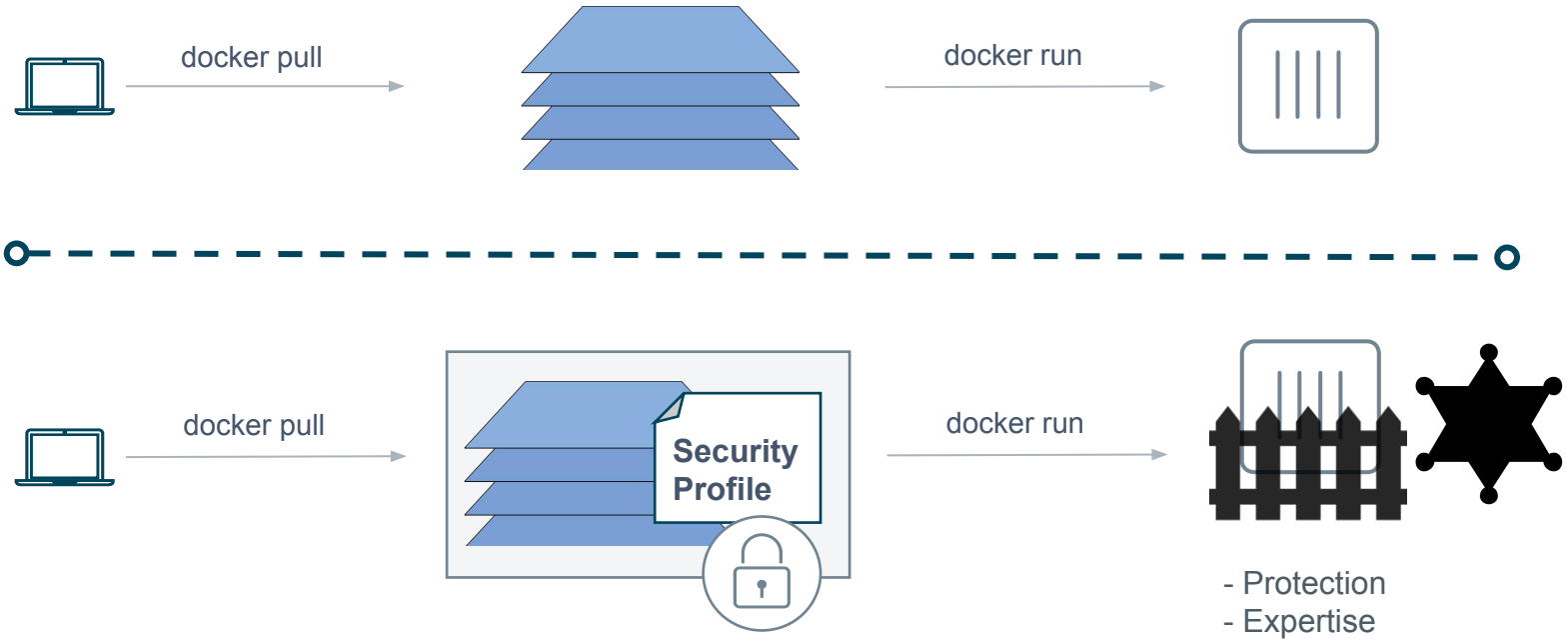
For App Publishers

Allow Content Publishers to advertise the best security settings



For Users

Who shouldn't have to deal with all that



Key Goals

- Great User Experience
- Empower both the developers and the devops
- New High-Level Permissions Standard
- Supported by most platforms
- Deprecate the infamous `--privileged` flag
- No universal default config
- Tie security profiles to images securely

But Also...

- Custom entitlements
- API Access Control
- Service-to-Service communication control (integration with service mesh)
- Many more, if you have additional ideas

Demo time? ^_^

What's left?

- Moby
  - Cleanup integration in Moby, Docker CLI and SwarmKit
  - As much community feedback as possible on default entitlements 🙏
  - Improve integration with `docker trust`
- Kubernetes
  - Finish the PRD
  - Community proposal
  - Implementation
- Docker integration design (image format, versioning, custom entitlements ..)

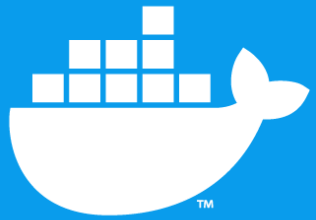
Pain Points

"Hey! Not so fast"

- "Collisions" on resource restriction
- Backward compatibility
- Standards are hard to define
- Baked-in entitlements trust management

How to Contribute?

- **Github repo:** https://github.com/moby/libentitlement
- <3 Feedback <3
  - Usability
  - Do default entitlements make sense?
  - Design opinion
- Integration PRs need more cleanup, stay tuned
- Reach out / open issues



THANK YOU :)

Nassim Eddequiouaq, Justin Cormack
nass@docker.com justin.cormack@docker.com

github@n4ss
twitter@n4zs_

github@justincormack
twitter@justincormack