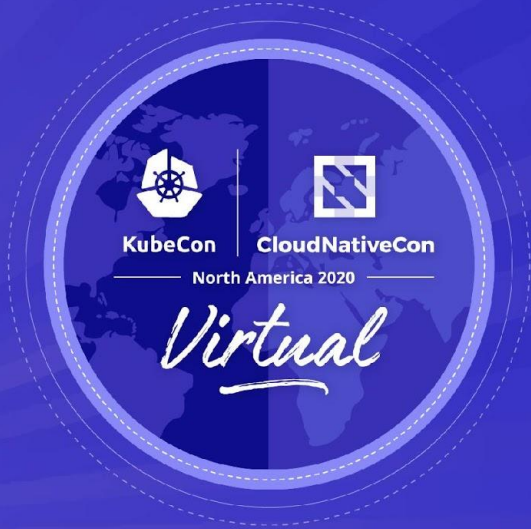
seccomp

what can it do for you?

Justin Cormack

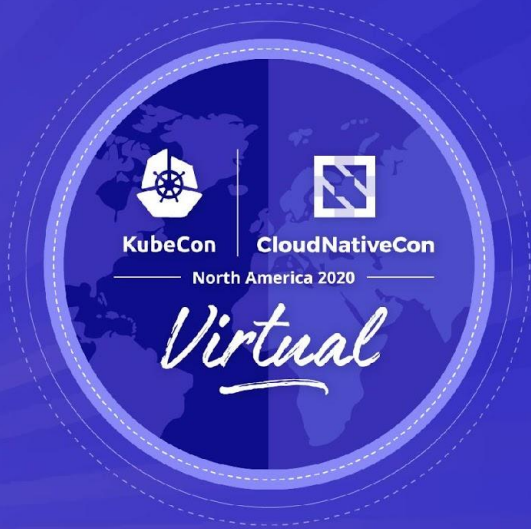


Justin Cormack

- Engineer at Docker
- Based in Cambridge, UK
- Notary maintainer
- CNCF TOC member
- CNCF SIG Security
- @justincormack

seccomp

what is it anyway?



“Secure Computing”

- Originally (2005) seccomp was an extreme sandboxing methods for code just doing compute
- Processes could call `read`, `write` (with existing file), `exit` and `sigreturn` only
- This was rarely used
- In 2013, seccomp BPF was introduced, allowing small BPF programs to be written to decide if syscalls should be allowed, error, be logged or kill the thread or process

In theory

In theory you take a look at what a program is doing

```
execve("/bin/ls", ["ls"], [/* 8 vars */]) = 0
brk(NULL)                                 = 0x55afee658000
access("/etc/ld.so.nohwcap", F_OK)        = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)        = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=25558, ...}) = 0
mmap(NULL, 25558, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fb8914f8000
close(3)                                  = 0
access("/etc/ld.so.nohwcap", F_OK)        = -1 ENOENT (No such file or directory)
open("/lib/x86_64-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0000k\0\0\0\0\0\0"..., 832) = 832
```

... and say yes to each call if it is ok, or return EPERM or ENOSYS if not allowed, or you want to pretend it does not exist

seccomp in practise

- You get very restricted information, eg for
  `open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC)`
  you actually only see
  `open(<pointer>, O_RDONLY|O_CLOEXEC)`
- So you cannot make decisions based on filenames, or other things where a struct is passed in a pointer
- You cannot inspect a file descriptor to know what type it is, for example if it is a network connection or a local file
- You need to allow for weird things like registers not being masked, multiple architectures (32 and 64 bit and halfway) being supported
- You need to know what should be allowed and why without context

In Docker and Kubernetes

- Jessie Frazelle and I did the work for Docker in early 2016
- Enabled by default since then, most people do not disable it
- In Kubernetes, prior to 1.19 you could annotate pods, since then you use `seccompProfile` in `securityContext`
- In Kubernetes you need to specify profiles as filenames, as apparently 800 more lines of Yaml is too much
- Not enabled by default, use `RuntimeDefault` for something like Docker

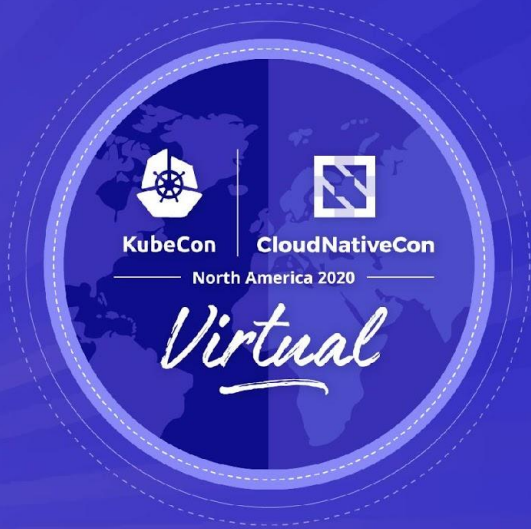
```
spec:
  securityContext:
    seccompProfile:
      type: RuntimeDefault
```

seccomp in OCI practise

- In the container space, seccomp is layered through two more abstraction layers
- First it is called via the Go bindings to libseccomp, which generates BPF code from a simpler description of matching rules
  - however libseccomp can only generate a subset of BPF programs
- The calls to libseccomp are generated from JSON that is defined by the runtime, this allows some runtime configuration in addition, so Docker will have slightly different rules on different architectures, and if you add other privileges
- JSON⇒JSON⇒Go⇒C⇒BPF

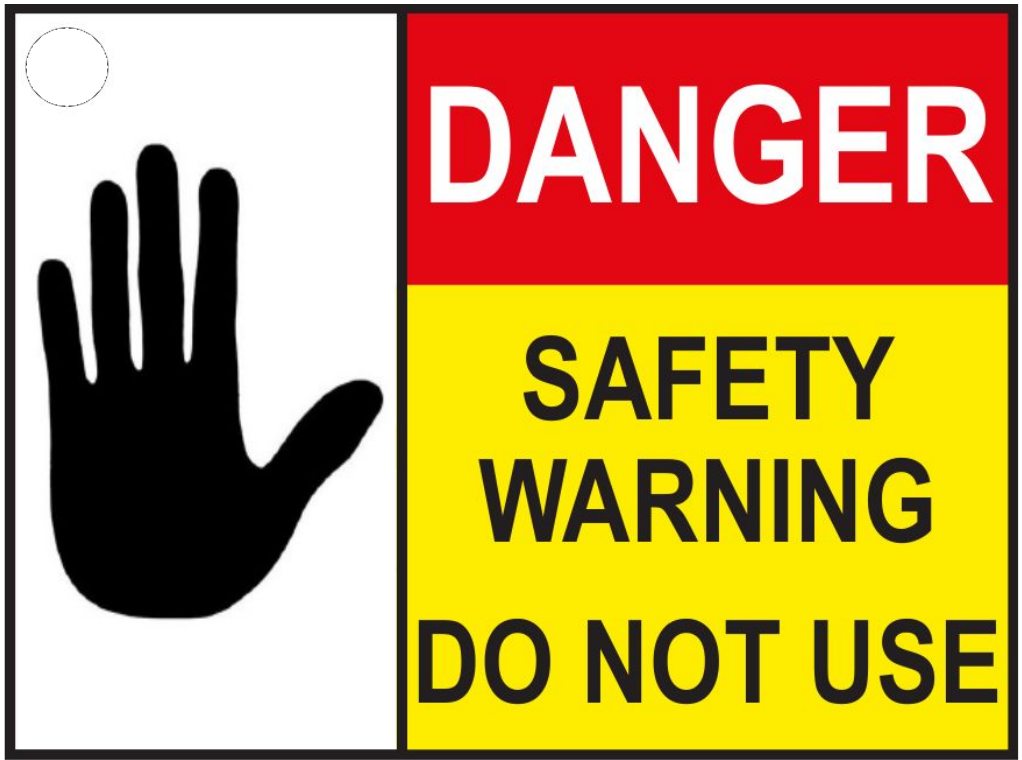
seccomp

why are we doing this?



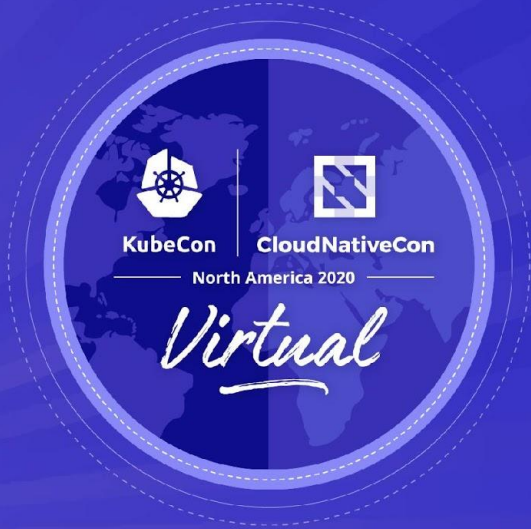
Do not use

- The Linux kernel includes syscalls that are often not considered safe for isolated programs to use
- huge attack surface, CVEs
  `perf_event_open`
  user namespaces
  `bpf`
- disable security features
  `ADDR_NO_RANDOMIZE`
- obsolete
  `sysctl`
- not namespaced
  keyrings, time before 5.6



seccomp

some good outcomes



User namespaces

- “I consider the ability to use CLONE_NEWUSER to acquire CAP_NET_ADMIN over /any/ network namespace and to thus access the network configuration API to be a huge risk. For example, unprivileged users can program iptables. I'll eat my hat if there are no privilege escalations in there.”
  Andy Lutomirski

CVE 2016-3134

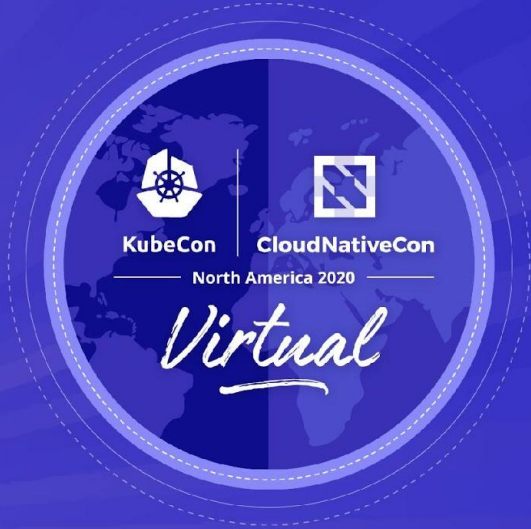
- CVE 2016-3134 “In the `mark_source_chains` function (net/ipv4/netfilter/ip_tables.c) it is possible for a user-supplied `ipt_entry` structure to have a large `next_offset` field. This field is not bounds checked prior to writing a counter value at the supplied offset.”
- Can be exploited if you use `setsockopt(IPT_SO_SET_REPLACE)`
- This normally requires `CAP_NET_ADMIN` which is not granted
- However, in a user namespace, you can create a new network namespace in which you are “local root” and can call these functions
- Blocking user namespace creation by seccomp mitigated these issues

CVE 2020-8835

- “In the Linux kernel 5.5.0 and newer, the bpf verifier (kernel/bpf/verifier.c) did not properly restrict the register bounds for 32-bit operations, leading to out-of-bounds reads and writes in kernel memory.”
- This could be exploited by an unprivileged user with access to the `bpf` syscall
- In general `bpf` is used in the control plane not in end user applications
- It is blocked in the Docker profile unless `CAP_SYS_ADMIN` granted

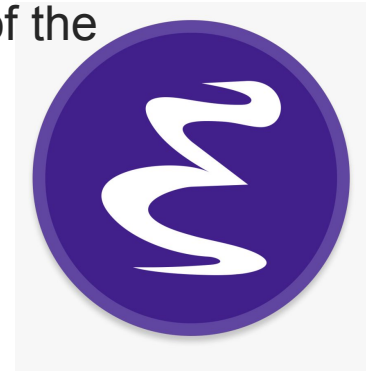
seccomp

some bad outcomes



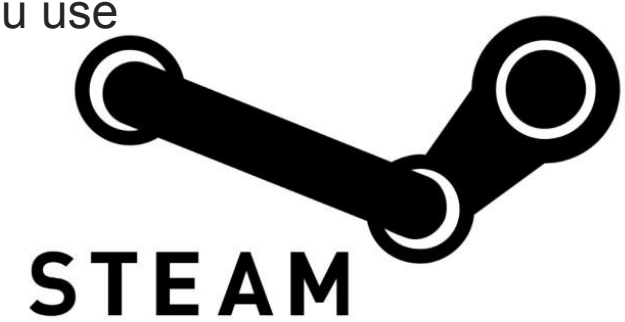
The war on Emacs

- Ah Emacs! I single handedly stopped people running Emacs in a container with seccomp for many years!
- “GNU Emacs' build process depends on the ability of the build-stage binary (temacs) to "dump" itself to a new executable file containing preloaded lisp objects/state in its `.data` segment. This process is highly non-portable even in principle; in practice, the big issue is where malloc allocations end up. They need to all be contiguous just above the `.data`/`.bss` in the original binary so that they can become part of the .data mapping.”
- Required disabling ASLR via `prctl(ADDR_NO_RANDOMIZE)`
- This makes exploiting many other things much easier!
- Eventually Emacs stopped doing this!



Accidentally broke Steam!

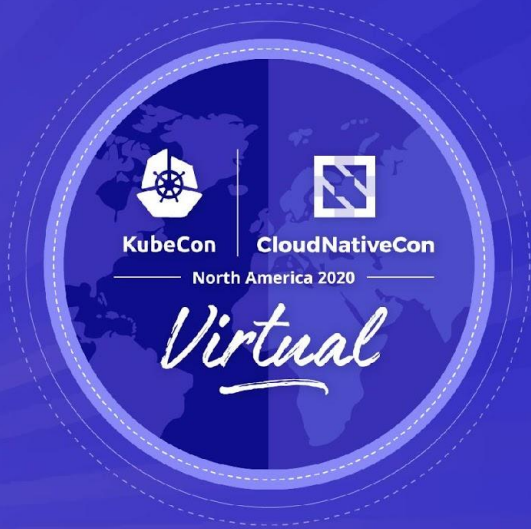
- Steam runs only 32 bit binaries, and is widely used
- Linux has made a lot of changes to 32 bit syscalls
- First one, that broke Steam, was the switch from `socketcall` to separate calls for `socket`, `bind`, `connect` etc...
- Second one more recently was when new calls were added to support 64 bit time on 32 bit systems, to save us from the year 2038
- Illustrates the fragility of the same user code potentially being converted to different syscalls depending what base image you use



Performance

- We use an allow list not a block list, so that new syscalls are blocked
- This list is very long, and not yet processed efficiently
- For IO intensive applications there is a significant performance hit
- Some of this can be fixed by using the existing binary search tree code, or by even more efficient BPF checking code
- The vast majority of users are not affected, and those that are just disable seccomp rather than working on the tedious work of fixing it

seccomp
did not help



CVE 2018-17182

- My favourite Linux CVE, found by Jann Horn, Google Project Zero
- cache invalidation bug in the Linux page fault code
- “However, this optimization is incorrect because it doesn't take into account what happens if a previously single-threaded process creates a new thread immediately after the mm_struct's sequence number has wrapped around to zero.”
- Fixed by changing a counter to 64 bits not 32 bits to avoid overflow
- Only requires `mmap` and `clone` to execute, so seccomp cannot protect
- Actually exploiting it is easier with information leaks however

seccomp

should we be using it like this?

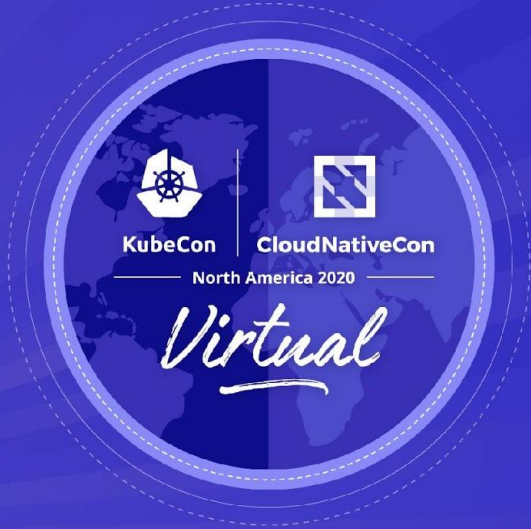
Is this right?

Why is a container platform responsible for the state of Linux kernel security?

- Because we want to have some isolation without virtual machines
- Because our applications do not generally use the whole Linux kernel syscall space, mostly they use a sane (but undefined) subset
- Although seccomp was designed for end user applications to use, not platforms and administrators, it has failed for the vast majority as it is too difficult to use

seccomp

choose your adventure



Don’t use it?

- The status quo is probably that only a few serious companies will use it
- Plus Docker users, which is now mostly a development platform, so makes little sense there now
- Update your kernel weekly
- Higher rate of 0-days

Are small blocklists better?

- I am coming around to the view that a small blocklist is a better approach
- Block `bpf`, user namespaces, `open_perf_event`, ASLR disable etc
- Easier to understand, less likely to break on upgrade
- Easier to customise inline in your yaml with an allow list
- Way less maintenance work and compatibility issues
- Risk of blocklist moving to zero because there is someone who wants everything

Is it better to push to runtime?

- Whose problem is this anyway?
  - user
  - application
  - Kubernetes
  - CRI eg containerd
  - runc
- Currently responsibility is pushed up to the user, but the threat model is not clear in this chain
- Why don’t we have runtimes that provide security guarantees? We are seeing these with VMs, gVisor, but the stack was not designed for this
- Why is the runtime a JSON syscall config?

gVisor

- gVisor looks at the problem by re-implementing much of Linux in a memory safe language (Go), and intercepting the syscalls.
- Heroic effort on the transparent unikernel spectrum
- The most successful in this line of bypassing much of Linux
- Uses seccomp internally but largely makes it unnecessary for users
- Has a performance hit too, and may have compatibility issues

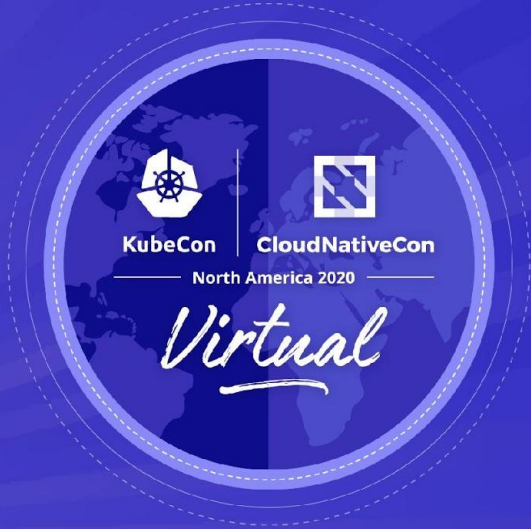
Lambda like?

- AWS Lambda is like a very restricted container runtime
- Uses seccomp (have not probed policy, LMK if you do)
- Uses a Linux kernel with features removed; normal Linux distros are designed with everything in usually as they are general purpose
- No application can run as root
- Restricted runtime API
- We could define an isolated container ABI and runtime more like this, and less like pick anything you want from Linux
- Clear delineation with control plane components that may be more privileged
- The “sandboxed” flag proposal but with a specification

eBPF LSM

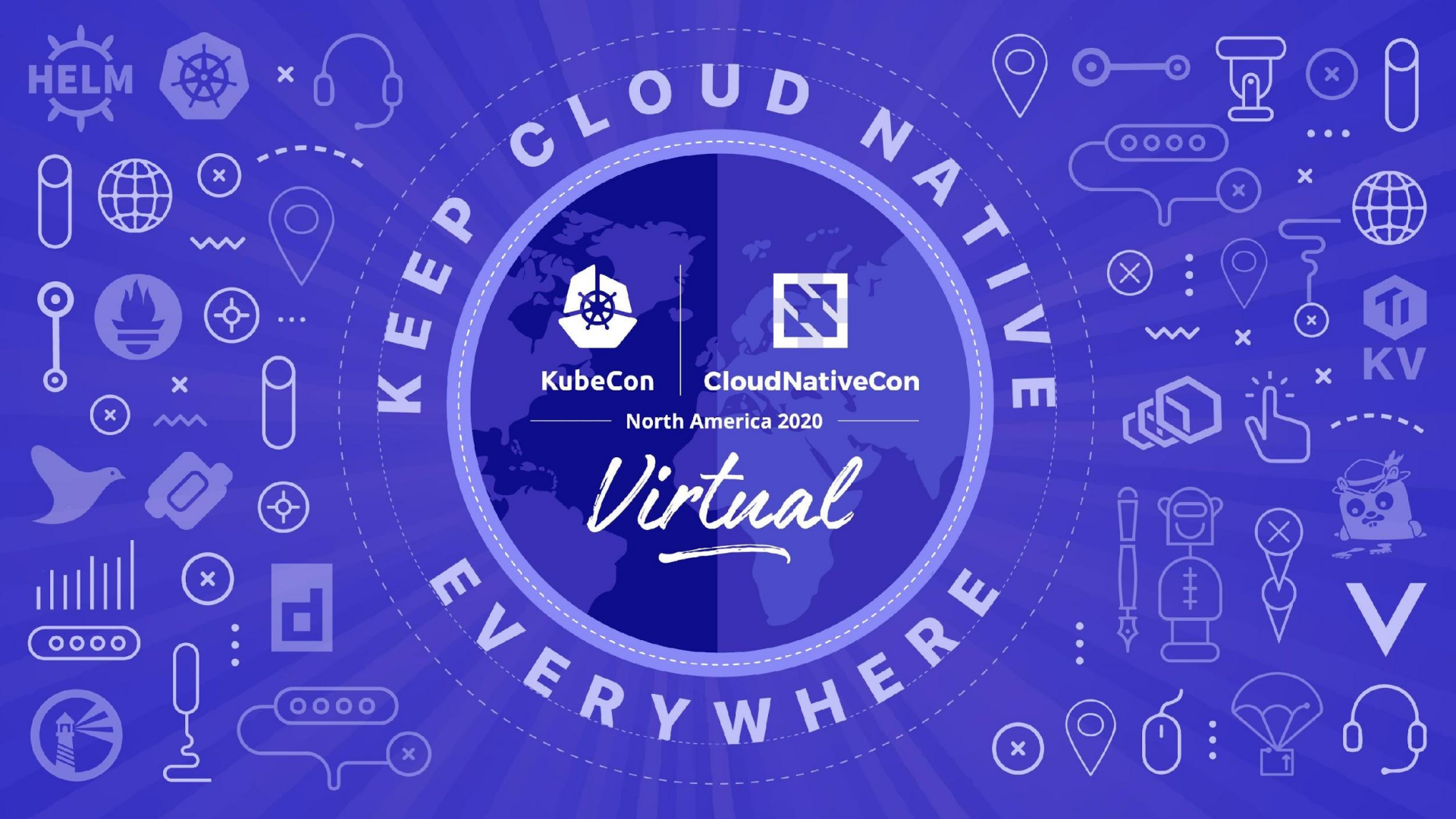
- Has been a long topic of conversation for years
- Merged in Linux 5.7
- Solve the lack of context and lack of programmability issue
- Custom LSM for different applications
- Obsoletes SELinux and AppArmor eventually
- Startup sized problem, or NSA sized?
- Technical solution does not solve human problems

seccomp
what will happen?



Prediction

- Continuing lack of investment in lower levels of stack mean that little will happen
- Majority expect someone else to solve their problems
- Funding that eBPF LSM container security startup is easier than most of the other options
- The serious service providers are already using VMs for containers anyway, with additional layers inside
- Security vendors are not solving this kind of problem
- End users find it difficult to contribute to this sort of problem, due to lack of expertise in these parts of the stack


KEEP CLOUD NATIVE
EVERYWHERE
KubeCon
CloudNativeCon
North America 2020
Virtual
HELM
KV