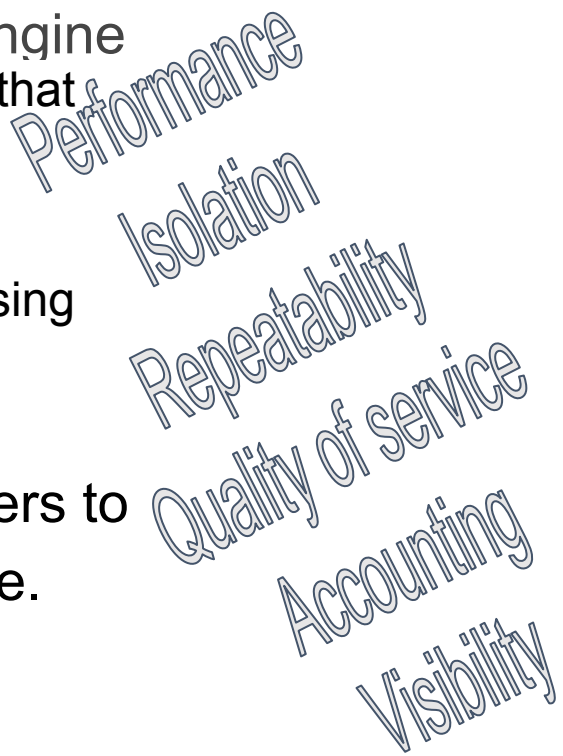
Container Isolation at Scale

(... and introducing gVisor)

Dawn Chen and Zhengyu He

Containers are amazing!

- Year 2013: Docker Inc. released its container engine
  - Million downloads and about 8,000 docker images that year
- Now the technology has really taken off
  - ESG survey shows about 40% of companies are using containers
  - Docker Inc. reports > 29 million downloads
- Google has been developing and using containers to manage our applications for more than a decade.
  - Launch over 4 billion containers per week.



But not contained!

- Security concerns remain
  - ESG survey shows 94% felt that containers negatively affect security
- The last decade has seen a lot of work on isolation mechanisms
  - Namespaces
  - Cgroups
  - Users
  - Capabilities
  - Chroot
  - Seccomp
  - Linux Security Modules (LSM)

Prior to Borg: Run as root

- All devices accessible
- Host filesystem accessible
- All resources consumable
- Network reconfigurable
- Can perform any kernel call
- Can SIGKILL others

```
root  234  /bin/sh
```

Prior to Borg: Run as root

What if anything goes wrong?

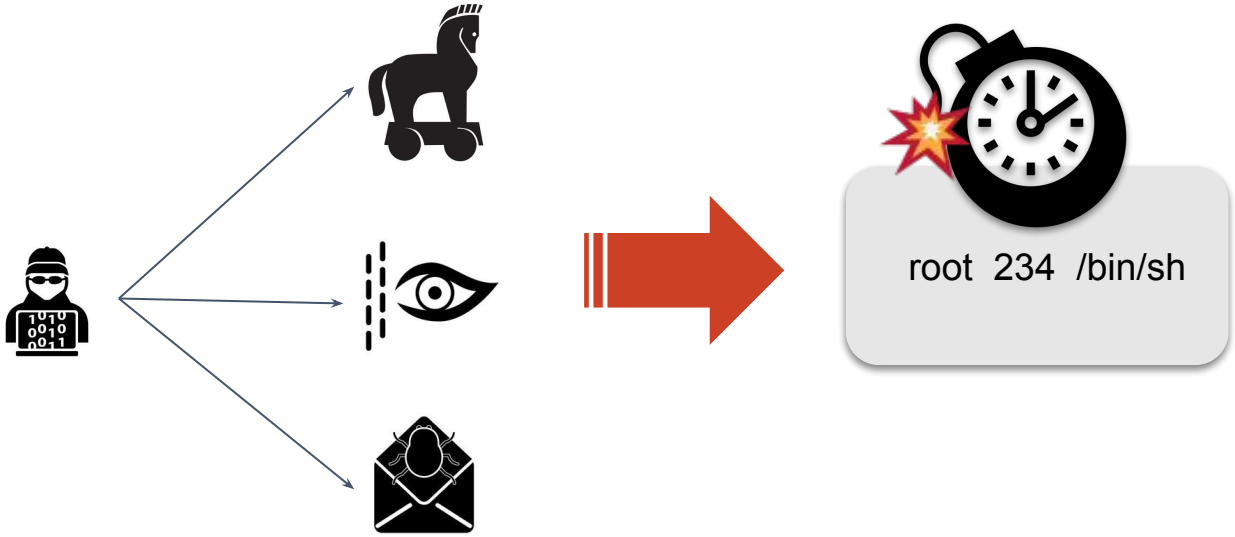
- A bug in a script

  *$ rm -rf $(UNDEFINED_DIR)/**

- Or malicious software?

root 234 /bin/sh

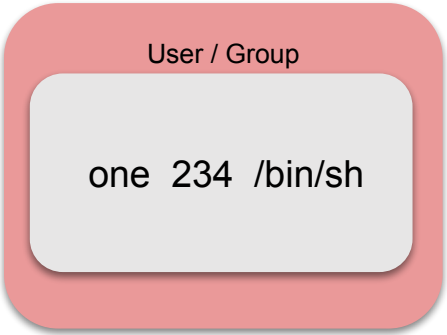
Prior to Borg: Container as root



Nothing shields the system!

Run as unprivileged user

- Limited devices access including network device
- Limited filesystem access
- Permissions of kernel calls are checked before execute
- Limited ability to send signals



But if setuid?

Drop capabilities

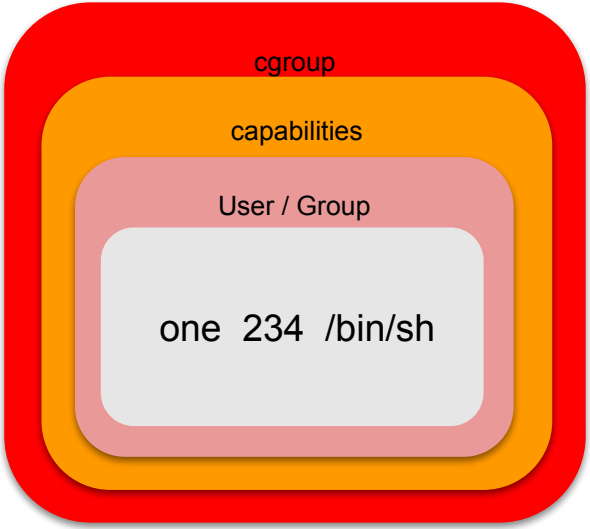
- Examples of dropped capabilities:
  SYS_MODULE, SYS_ADMIN, SYS_TIME,
  SYS_RESOURCE, NET_ADMIN, SYS_LOG, …
- Fewer capabilities, better isolation!

Now ok with privilege isolation, what about resource isolation?

Apply CGroups

- Cgroup limits, accounts for, and isolates the resource usage:
  - cpu - limits access to the CPU
  - cpuacct - accounts cpu usage by cgroup
  - cpuset - assign cores & memory nodes to cgroup
  - devices - control device access by cgroup
  - memory - limits & accounts memory usage

and more

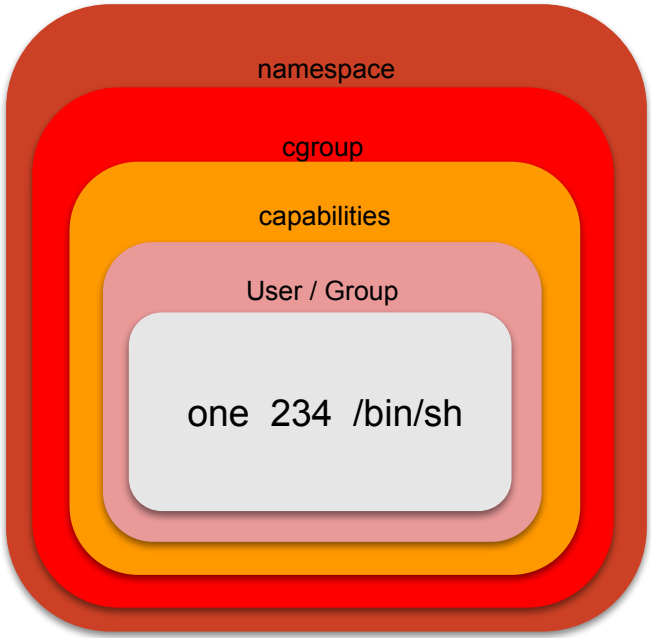


But still can see all processes, network interfaces, mount points on the system!

Apply namespace

- Provide isolation for each namespace type
- Currently support 7 different namespaces:

  Network, PID, mount, user, IPC, UTS, cgroup
- More to come



Is this enough?

Still ...

“Containers do not contain”

--- Dan Walsh, 2014

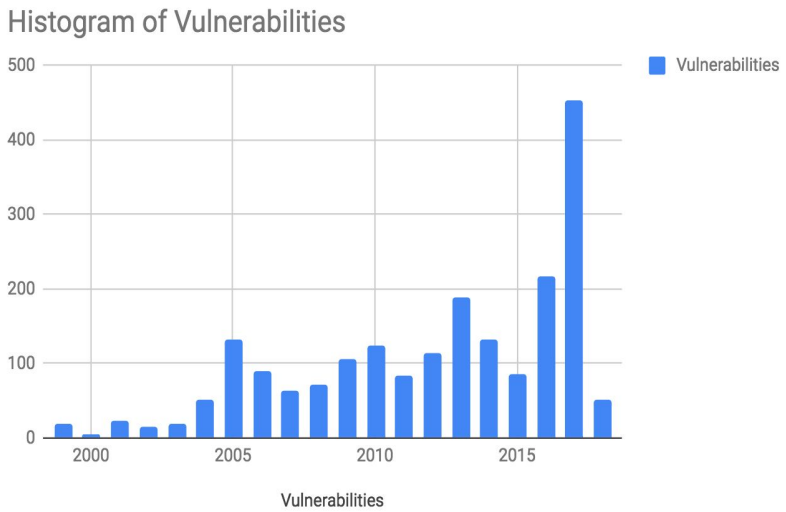
- The kernel supports several alternative ways to configure fine-grained access control per process, using Mandatory Access Control:
  - SELinux
  - AppArmor
- "secure computing mode" - but really we mean **seccomp-bpf**
  - Filter syscalls

Not quite yet ..

- “Each container also gets its own network stack” (from Docker security site).
  - Not really. It just has its own interface, but uses the same linux TCP/IP stack.
  - CVE-2013-4348 A single malformed packet from remote can crash your kernel
- There are more ...
  - CVE-2016-5195 DirtyCOW
  - CVE-2017-5753/5715/5754 Spectre/Meltdown

Why?

- Still sharing the same kernel
- Share same device drivers
- Linux kernel represents a large attack surface.
- CGroup accounting may not be accurate



What is Next?

As a Container Fan, I wish

1. An image I pulled from a random corner of the world should not exploit my Linux box.

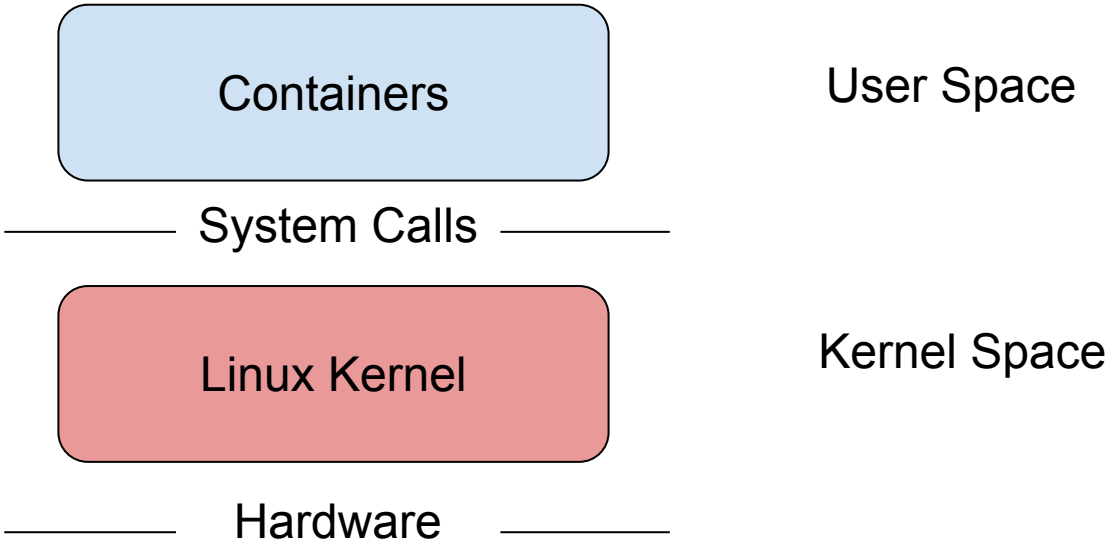
2. Little work or no work required from me.
   - Not overly restricted
   - No modification to the application

3. Feels like a container
   - Fast startup
   - Cheap to run: low memory consumption

As a Security Engineer, I know

- I need more than one **security layer** between a untrusted workload and my ~~Bitcoin wallet~~.
  production job
- So that no single compromise can steal all of my ~~coins~~.
  user data

Rethink Container Isolation

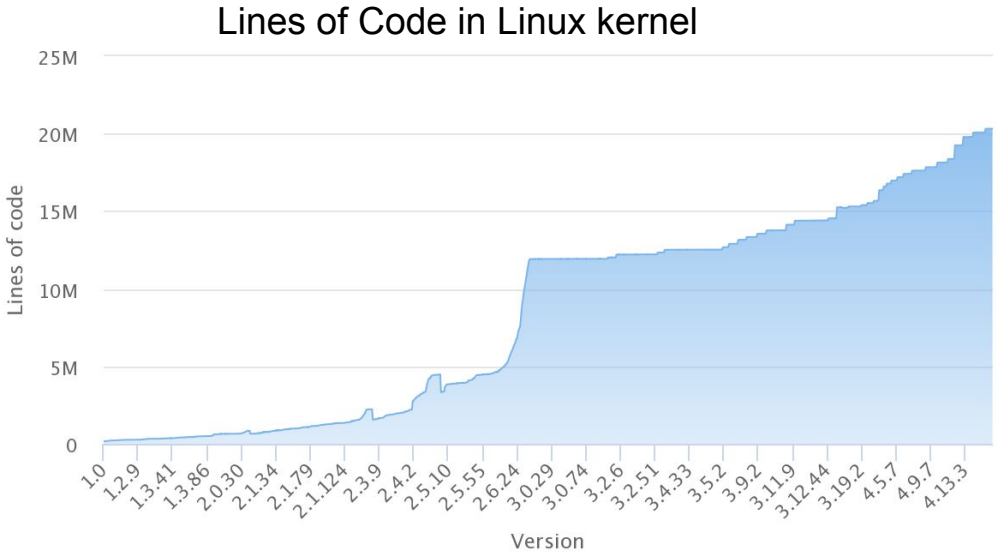


Linux Fun Facts

- 319 native 64-bit syscalls in Linux x86_64

```
grep x64 arch/x86/entry/syscalls/syscall_64.tbl
```

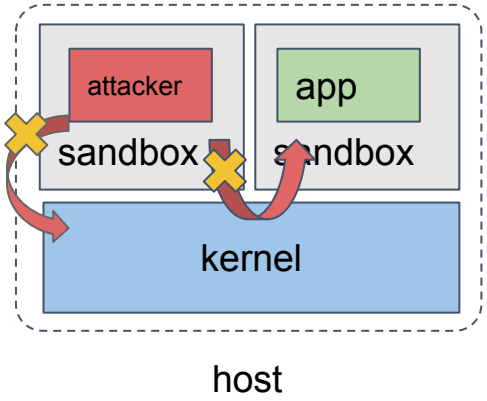
- 2046 CVEs since 1999
  - 257 Privilege escalations



https://www.linuxcounter.net/statistics/kernel

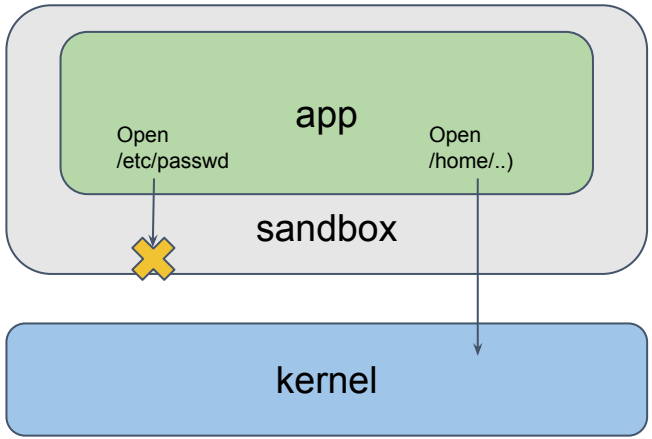
Sandbox

- Sandbox is an effective layer to reduce the attack surface.



Recap: Rule-based Sandbox

- AppArmor, SELinux, Seccomp-bpf



Reduce the attack surface by restricting what the application can access.

Linux Security Modules

- A framework used by AppArmor, SELinux
  - Kernel-module enforcing rules

- http://stopdisablingselinux.com/

```
/usr/sbin/tcpdump {
  #include <abstractions/base>
  #include <abstractions/nameservice>
  #include <abstractions/user-tmp>

  capability net_raw,
  capability setuid,
  capability setgid,
  capability dac_override,
  network raw,
  network packet,

  # for -D
  capability sys_module,
  @{PROC}/bus/usb/ r,
  @{PROC}/bus/usb/** r,

  # for -F and -w
  audit deny @{HOME}/.* mrwkl,
  audit deny @{HOME}/.*/ rw,
  audit deny @{HOME}/.*/** mrwkl,
  audit deny @{HOME}/bin/ rw,
  audit deny @{HOME}/bin/** mrwkl,
  @{HOME}/ r,
  @{HOME}/** rw,

  /usr/sbin/tcpdump r,
}
```

Syscall Filtering

- `ptrace`
  - Checking in userspace. Vulnerable to TOCTOU if multi-threaded.
- Seccomp-bpf
  - In-kernel
  - Multi-threading safe (after TSYNC)
- Alt-syscall
  - Slightly faster (O(1) lookup time)
  - Not as flexible as seccomp-bpf

```
#define VALIDATE_ARCHITECTURE \
    BPF_STMT(BPF_LD+BPF_W+BPF_ABS, arch_nr), \
    BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, ARCH_NR, 1, 0), \
    BPF_STMT(BPF_RET+BPF_K, SECCOMP_RET_KILL)

#define EXAMINE_SYSCALL \
    BPF_STMT(BPF_LD+BPF_W+BPF_ABS, syscall_nr)

#define ALLOW_SYSCALL(name) \
    BPF_JUMP(BPF_JMP+BPF_JEQ+BPF_K, __NR_##name, 0, 1), \
    BPF_STMT(BPF_RET+BPF_K, SECCOMP_RET_ALLOW)

#define KILL_PROCESS \
    BPF_STMT(BPF_RET+BPF_K, SECCOMP_RET_KILL)
```

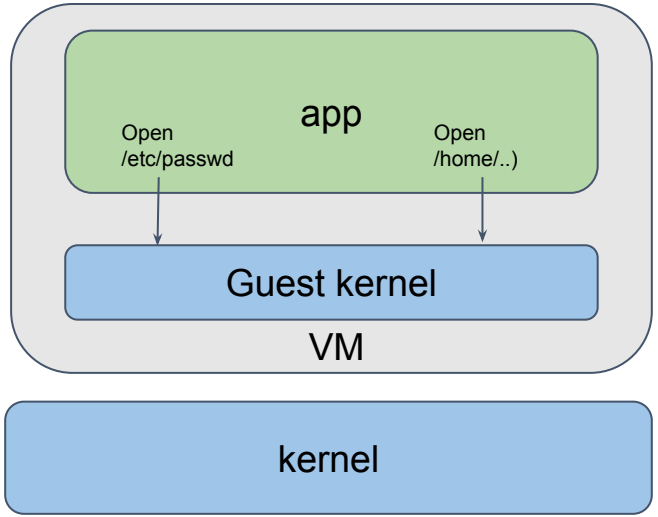
Still not so easy

- Writing the rules are tedious
  - Smart engineers like @jessfraz will automate it.
- The rules are fragile
  - Overfitting or underfitting
  - Friendly reminder: Go users, don't forget to include `epoll_pwait` in your seccomp filters. http://golang.org/cl/92895
  - Not completely secure
  - Spectre/Meltdown

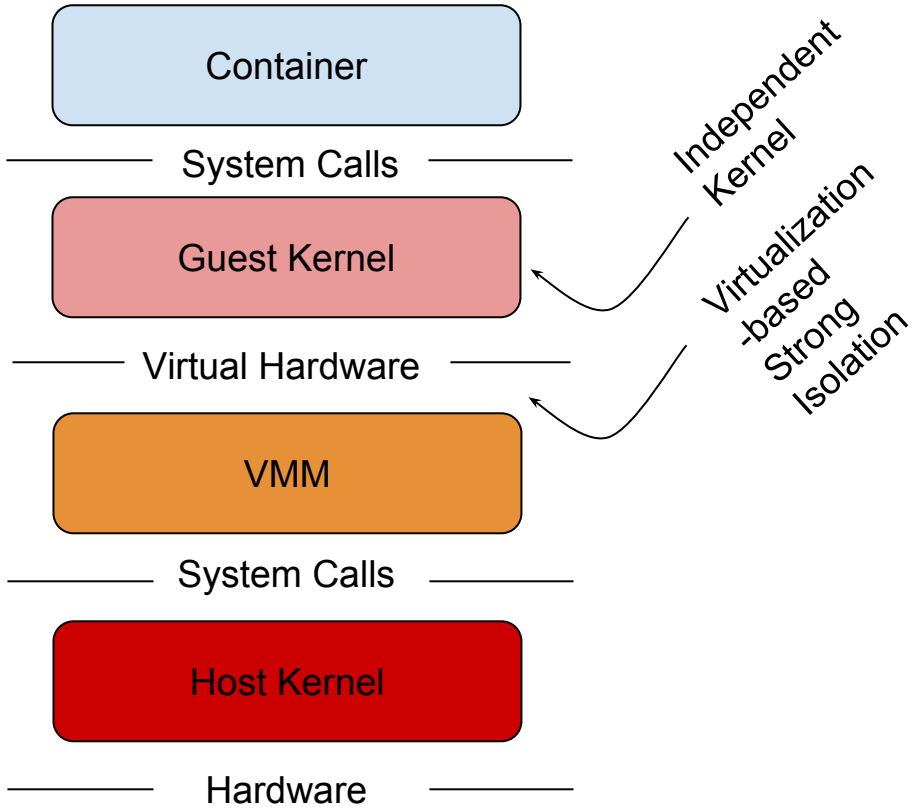
Hypervisor-based

- Universal!
- Strong Isolation

- Heavy weight
  - Extra software (Hypervisor+VMM+Guest Kernel)
- Inflexible resource boundaries
  - Linux needs to know the number of CPUs/Memory at boot



Rethink Containers Isolation Provided by VMs

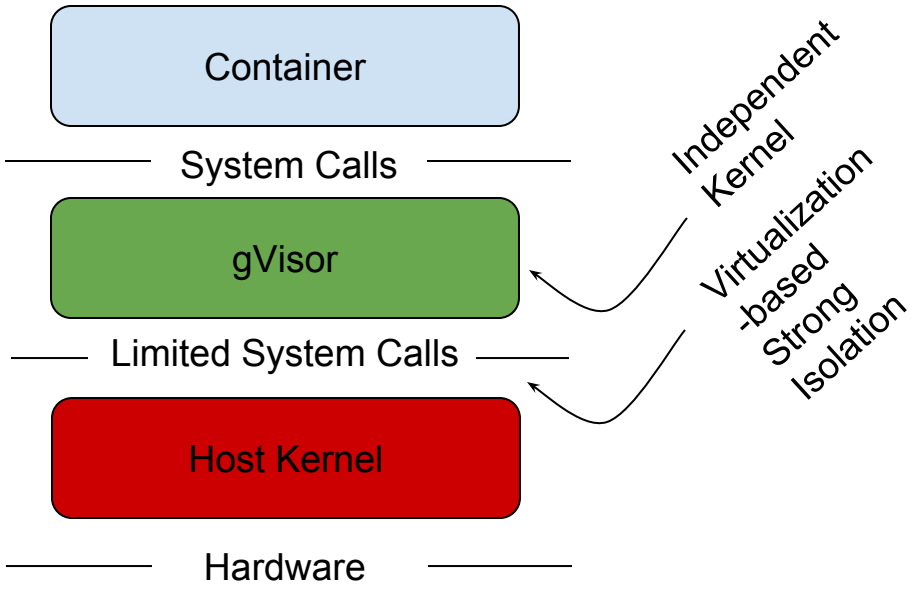


Lesson Learned

- Key Ingredients:
  - Independent Kernel
  - Virtualization hardware is an important defensive layer
    - Clear privilege separation and state encapsulation

- Collaterals:
  - Virtualized hardware interface
    - Inflexible
    - Obscure primitives (I/O ports, interrupts, exceptions)
  - The Linux kernel
    - One-size-fit-all
    - Monolithic (everything in the same address space)

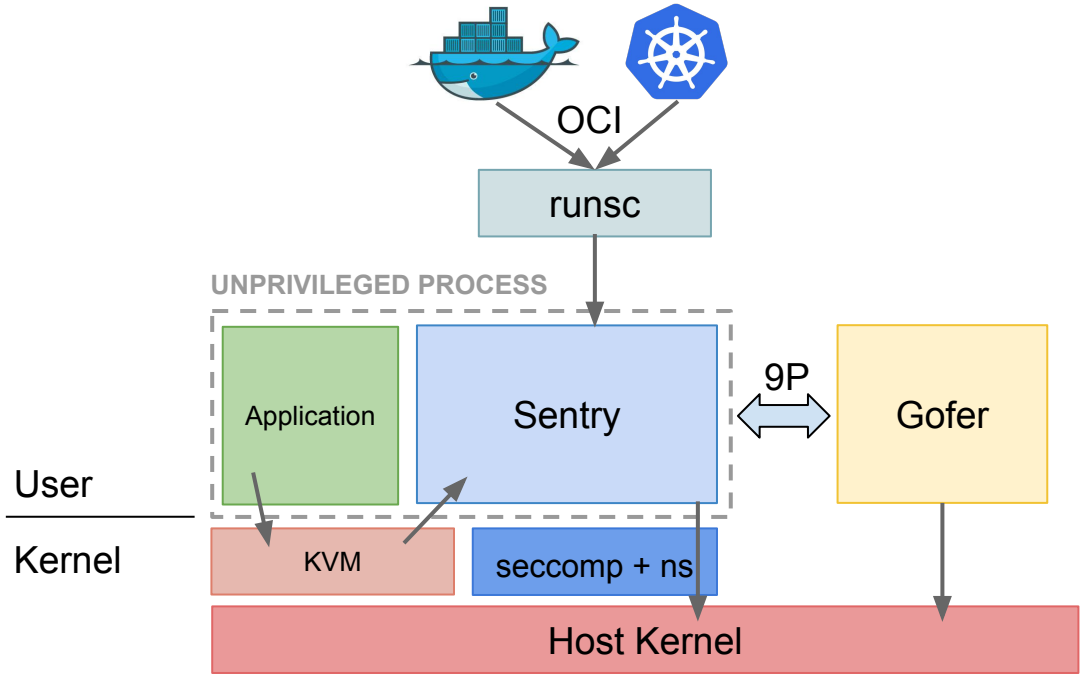
Our Approach -- gVisor



What is it really?

- Sandboxes untrusted applications
- Implements Linux system API in user space
  - 211 syscalls so far
  - Not a port like UML or LKL
  - Not just filters (as opposed to seccomp-bpf)
  - Runs unmodified Linux binaries (as opposed to NaCL)
- Secure by default
  - No filter configuration, AppArmor or SElinux policies
  - One kernel per sandbox
- Written in Go, a memory/type-safe language
- Save/Restore is a first-class citizen

Runsc: An OCI runtime powered by gVisor



Made for Containers

150ms

startup time*

15MB

memory overhead*

- Use as you go: no fixed resource
- Easy to debug
- ….

*collected with /bin/true and /bin/sleep

Cautions

What it IS good for:

- Small containers
- Spin up quickly
- High density

What it’s NOT good for:

- Trusted images
- Syscall heavy workloads
- Direct access to hardware, i.e. passthrough device support

Ramon @ KubeConEU
@rvcdbn

Follow

wondering how easy it would be to implement custom system calls in gVisor - could be a powerful tool for OS research

8:47 AM - 2 May 2018 from Copenhagen, Denmark

Wanna Try?

- Go to: https://github.com/google/gvisor
- 6 commands, then you are good to go

```
$ docker run --runtime=runsc hello-world
```

```
$ docker run --runtime=runsc -p 3306:3306 mysql
```

Want more?

- Talk to us at the gVisor booth.
- Join: https://groups.google.com/forum/#!forum/gvisor-users
- Get involved:
  - https://github.com/google/gvisor
  - Join sig-node for discussion
- Other talks:
  - Secure Pods (Fri, 5/4 11:10 - 11:45)
  - Kubernetes Runtime Security (Fri, 5/4 14:45 - 15:20)

Questions?