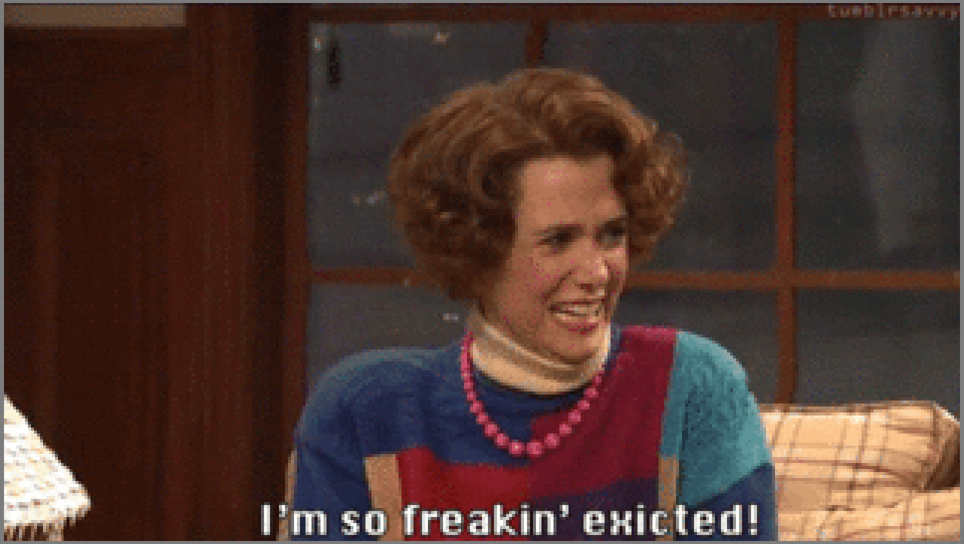
CONTAINING SECURITY

bit.ly/2017-containing_security

Vincent Batts @vbatts

```
$> finger $(whoami)
Login: vbatts                                   Name: Vincent Batts
Directory: /home/vbatts                         Shell: /bin/bash
Such mail.
Plan:
OHMAN
$> id -Gn
devel opencontainers docker appc redhat golang slackware
```

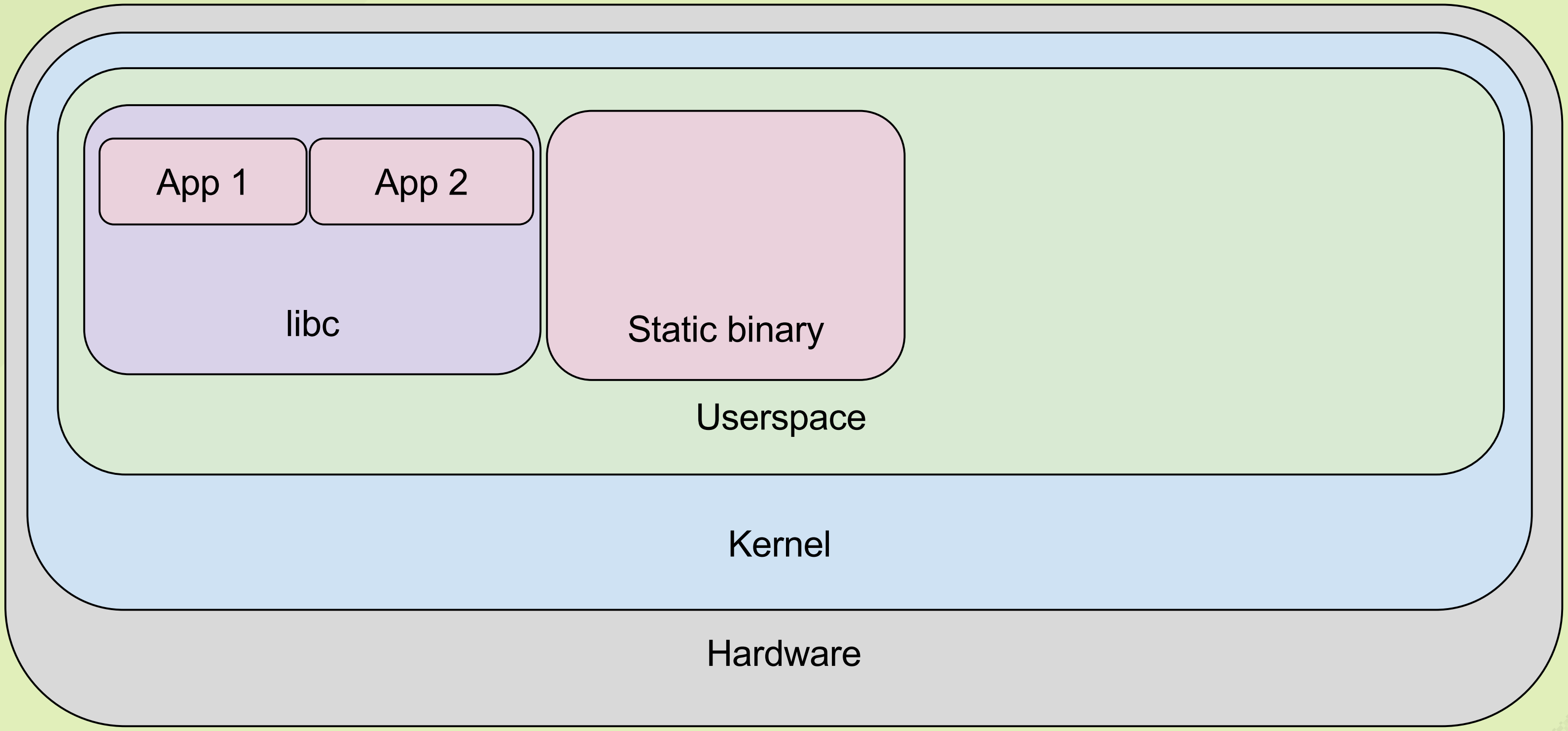


CONTAINERS

(Cite: the internet)

CONTAINERS

(Cite: The Internet)


App 1
App 2
libc
Static binary
Userspace
Kernel
Hardware

Kernel's Guarantee:

DON'T BREAK USERSPACE

Kernel's Guarantee:

DON'T BREAK USERSPACE

But what is there to break?

Kernel's Guarantee:

DON'T BREAK USERSPACE

But what is there to break?

- syscalls (open, read, write, close, exec, fork, mmap, mount, stat, etc.)

Kernel's Guarantee:

DON'T BREAK USERSPACE

But what is there to break?

- syscalls (open, read, write, close, exec, fork, mmap, mount, stat, etc.)
- signals

Kernel's Guarantee:

DON'T BREAK USERSPACE

But what is there to break?

- syscalls (open, read, write, close, exec, fork, mmap, mount, stat, etc.)
- signals
- ioctl's

Kernel's Guarantee:

DON'T BREAK USERSPACE

But what is there to break?

- syscalls (open, read, write, close, exec, fork, mmap, mount, stat, etc.)
- signals
- ioctl's
- prctl's

Kernel's Guarantee:

DON'T BREAK USERSPACE

But what is there to break?

- syscalls (open, read, write, close, exec, fork, mmap, mount, stat, etc.)
- signals
- ioctl's
- prctl's
- fcntl's

Kernel's Guarantee:

DON'T BREAK USERSPACE

But what is there to break?

- syscalls (open, read, write, close, exec, fork, mmap, mount, stat, etc.)
- signals
- ioctl's
- prctl's
- fcntl's
- sysfs

Kernel's Guarantee:

DON'T BREAK USERSPACE

But what is there to break?

- syscalls (open, read, write, close, exec, fork, mmap, mount, stat, etc.)
- signals
- ioctl's
- prctl's
- fcntl's
- sysfs
- procfs

Kernel's Guarantee:

DON'T BREAK USERSPACE

But what is there to break?

- syscalls (open, read, write, close, exec, fork, mmap, mount, stat, etc.)
- signals
- ioctl's
- prctl's
- fcntl's
- sysfs
- procfs
- and more, I'm sure

Kernel's Guarantee:

DON'T BREAK USERSPACE

But what is there to break?

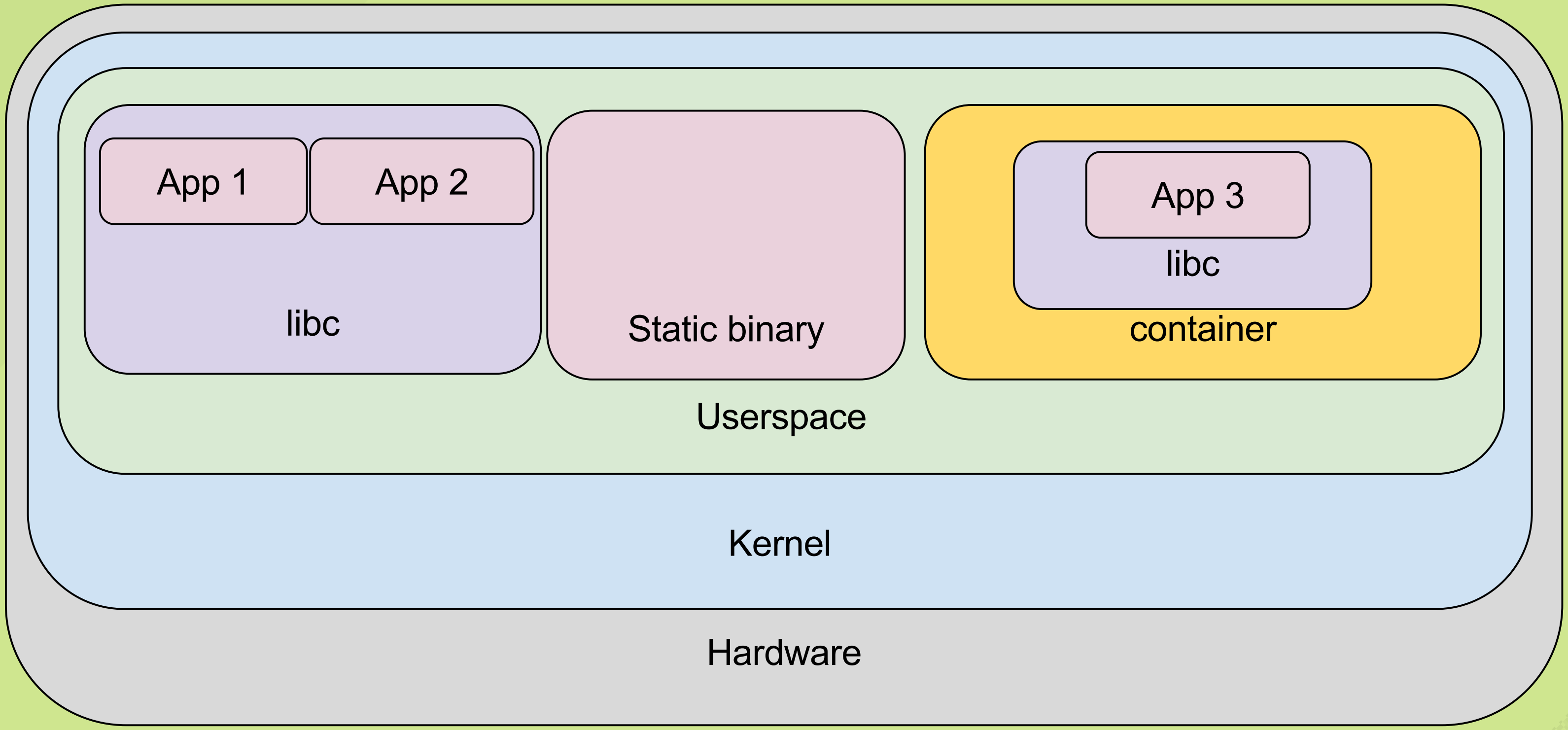
- syscalls (open, read, write, close, exec, fork, mmap, mount, stat, etc.)
- signals
- ioctl's
- prctl's
- fcntl's
- sysfs
- procfs
- and more, I'm sure

It's sprawling surface to deal with

EPERM

EACCES

Context of errors is in kernelspace, not userspace


App 1
App 2
libc
Static binary
App 3
libc
container
Userspace
Kernel
Hardware

CONTAINERS:

CONTAINERS:

Share the host's kernel

CONTAINERS:

Share the host's kernel

Crashes and Exploits alike

CONTAINERS:

- Share the host's kernel
    - Crashes and Exploits alike
- virtualizing by "namespacing" kernel resources and concepts

CONTAINERS:

Share the host's kernel

Crashes and Exploits alike

virtualizing by "namespacing" kernel resources and concepts

Isolation by control groups, syscall filtering, and Linux Security Modules (SELinux, apparmor, etc.)

KERNEL NAMESPACES:

unshare() and namespaces

- mount
- IPC (message queues, semaphores, shm)
- UTS (hostname)
- network
- PID
- cgroup
- user

KERNEL NAMESPACES:

Orthogonal in nature

Varying levels of maturity

Drastically increase complexity and attack surface

KERNEL NAMESPACES:

`User Namespace`

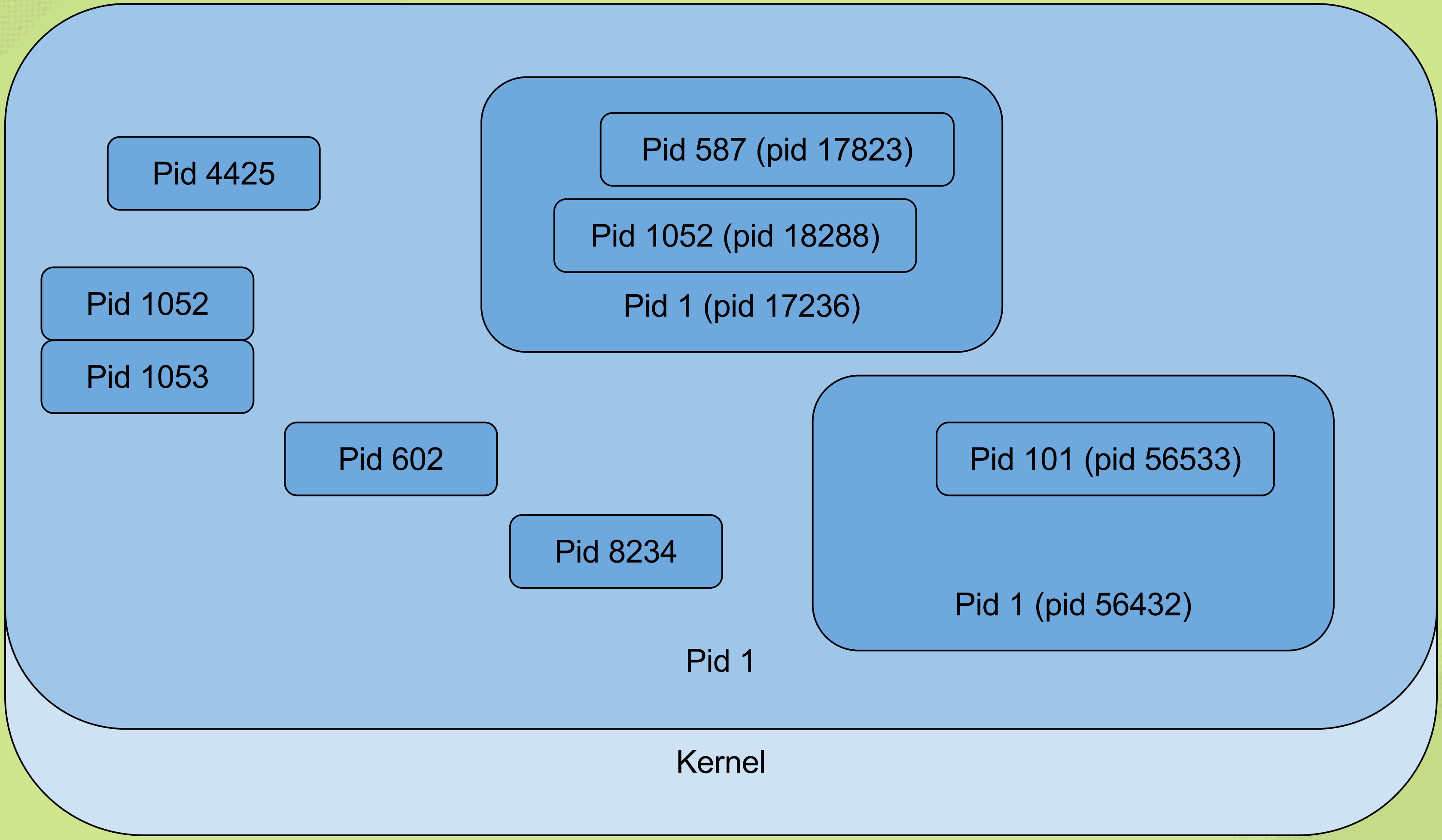
- neat step for isolation
- notable source of root escalations in the kernel
- still no viable vfs solutions (apart from chown'ing)

OpenShift (and others) are opting for
just explicitly running as non-root UID

`runc' can now launch non-root containers directly

Access to Docker daemon means root privilege. Period.

KERNEL NAMESPACES: PID



LSM (Linux Security Modules)

- Kernel Framework
- There are several. Most compare SELinux vs. Apparmor
- (Comprehensive and Complex) vs. (Simple and Narrow)
- (RBAC and MAC) vs. (just MAC)

Capabilities

- capabilities(7)
- Determine an application's capabilities (and syscalls too)
- SystemTap (stap)
- no_new_privs flag

Syscalls

- *wide* surface area
- attempt at syscall reference
- seccomp(2)
- Container runtime configuration

grsecurity

- paid subscription to patches
- breaks support for kernel
- RBAC, like SELinux

Audit

- Linux Audit
- BPF in kernel
  - bpf(2)
  - eBPF Superpowers
  - eBPF overview
- remove `docker' group. Require `sudo'
- Container Runtime Events
- OpenShift events and tracing
- L7 application insights and policies

LOCK-STEP

`Signing`

- simple signing vs. Docker notary
- detached, static vs. isolated service
- your key rotation process vs. its key rotation process
- Determine your requirements and use-cases

CLOUD

(Cite: the internet)

VINCENT BATTS

@VBATTS| VBATTS@REDHAT.COM

THANKS!