# How to make friends and win over your security team

Detecting and fixing security issues from scratch

# Who am I?


- Pushkar Joglekar (he/him)
- Nashik -> Pune -> S.F. Bay Area
- Speaks Marathi, Hindi and English
- Sr. Security Engineer @ VMware Tanzu
- Previously worked @ Visa - Securing large scale container environments as an end user
- Wrote a book with Nigel Poulton
- Spoke at Kubecon NA 2019
- Member of Kubernetes SIG Security and CNCF TAG Security
- Find more about me: https://pushkarj.github.io/
- More active on Twitter than LinkedIn

## Developers

- Ship awesome new features
- Explore cool tech
- Solve people's problems
- Make a dent in the universe

## Customers

- Satisfy their customers
- Rarely goes down or easy to fix (e.g. restart)
- Don't get hacked
- Be fiscally responsible

## Security Team

- Don't stop people from shipping
- Don't hamper developer productivity
- Secure by default
- Risk v/s Economics

# Myths about Kubernetes (+ Containers)

- Secure by Default?
  - Network policies are open
  - Not CIS compliant
  - Vulnerable images are allowed
  - No Encryption of Secrets
  - No seccomp, capabilities, app armor, SElinux enabled*
  - Root containers allowed*
  - Privileged container allowed*
  - Host path mounts allowed*
- Multi-tenant?
  - RBAC does not help with workload isolation
  - Two pods from untrusted tenants share nodes and kernel
  - Containers do not have hypervisor isolation**

*This might change in future k8s releases
** Unless RuntimeClass that supports a CRI with hypervisor isolation is enabled

# Let’s focus on running Kubernetes pods securely

# Run as non-root user

# Where can a user be defined?

- Dockerfile
- Security Context
- Pod Security Policy
- Admission controller as a webhook

**Best Practice:** Run as non-root user i.e. any user with ID that is not zero

# How do I know I am doing the right thing?

# Story

As a developer, when I am building apps as a container:

- I want to know if my app runs as a non-root user:
  - As a standalone container
  - As a pod

# As a container- Part 1

```
docker run -it --entrypoint=whoami nginx
```

# As a container- Part 2

```
cat DOCKERFILE | grep "USER"
```

# As a pod

```
cat pod.yaml | grep runAsUser
```

# Is my image vulnerable?

# Story

As a developer, when I am building apps as a container:

- I want to know:
  - If my image has vulnerabilities ?
  - Which of these vulnerabilities can I fix?
  - Validate if I actually fixed the vulnerabilities

# Installing a scanner locally

Install trivy: https://aquasecurity.github.io/trivy/latest/installation/

** Trivy used as an example, you can pick your own scanner

# Running a scan

```
trivy image <image-name>:<image-tag>
```

# Knowing what can I fix?

```
trivy image --ignore-unfixed <image-name>:<image-tag>
```

# Make what you care about the default!

```
tiiu='trivy image --ignore-unfixed'

tiiu <image-name>:<image-tag>
```

# To cache or not to cache ?

# Image layer caching

- Helps reuse of image layers
- Faster builds
- `pull` if layer not found is the default

# Image layer caching

```
FROM debian
```

- This will get you latest debian image, when you are building for the first time
- Next time, it will not pull even if `latest` image that `latest` tag points to has changed
- This means that any security fixes applied to latest image are missed

# Image layer caching

```
RUN apt -y update && apt -y upgrade
```

- First command *pulls info* about available updates
- Second commands *installs update*
- By default this command will run for the first time
- Output of this layer is then reused for next build
- So next time when new (security) updates are available they are ***not*** installed

# How to pull when you want?

`--pull`

- Ensures that base image is pulled regardless of caching state
- This will allow `FROM debian` to pull the latest available debian image

# How to not use cache when you want?

`--no-cache`

- Ensures other cached layers are not reused
- This will execute `RUN apt -y update && apt -y upgrade` regardless of caching status

# Image Layer Caching - Caveats

- Some images have pinned images where tag is immutable
- In this case `--pull` has no effect
- For release branches: Pinned images with immutable tags e.g. `FROM debian:10.9`
- For feature branches: Pin to mutable tag e.g. `latest`

# Image Layer Caching - Caveats

For release branches: Skip `--no-cache`

- This enables consistent layers for each release build e.g. `fc -> rc1 -> rc2 -> GA`
- Can override to fix a critical vulnerability

For feature branches: Use `--no-cache`

# How do you apply this next week?

# Check if the image of your app run as root user

# Check if image of your app is vulnerable

Check if this is because of layer caching

Fix it, Share it with your security team and wait for their reaction :-)

And don’t forget to tweet @PuDiJoglekar and tell me how they reacted

# Thank you