OWASP

Open Web Application Security Project

UPDATED

How to stop worrying about Application Container Security (v2)

Brian Andrzejewski

Information System Security Architect

Twitter: @DevSecOpsGeer

LinkedIn: https://www.linkedin.com/in/bandrzej

Disclaimers

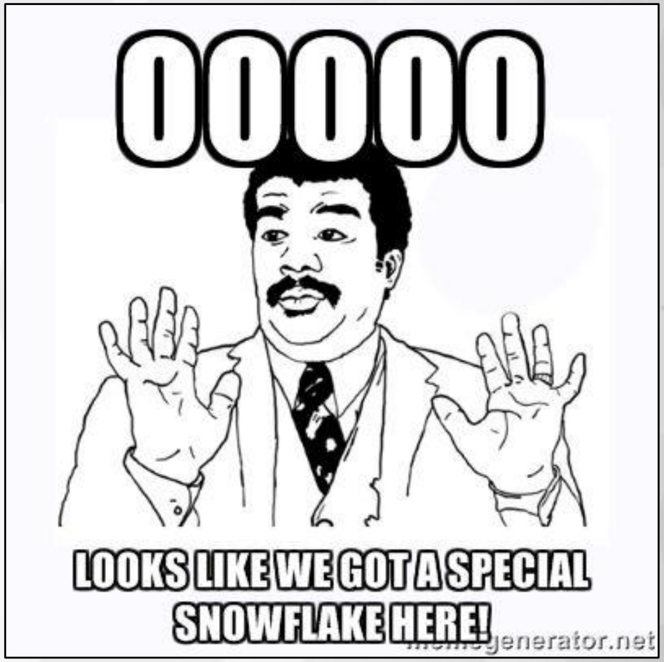
- My personal views and opinions may not represent the position(s) of my employers.
- Mention of any OSS or commercial product names in this talk are not an endorsement.
- Information provided is not public sensitive and based on my 3 years of container security ops.

About Me

- Specialized in AppSec, DevOpsSec, CloudSec, & Vulnerability Assessment
- Prior Help Desk Support, WebDev, SysAdm, Project Manager, Forensic Examiner, & Security Auditor
- Worked in academia, healthcare, risk mgt, contracting, & government

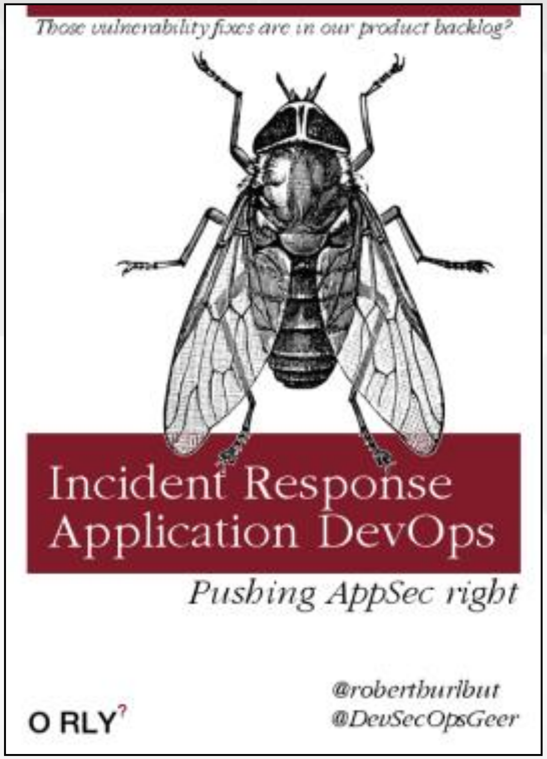
Typical Application Challenges

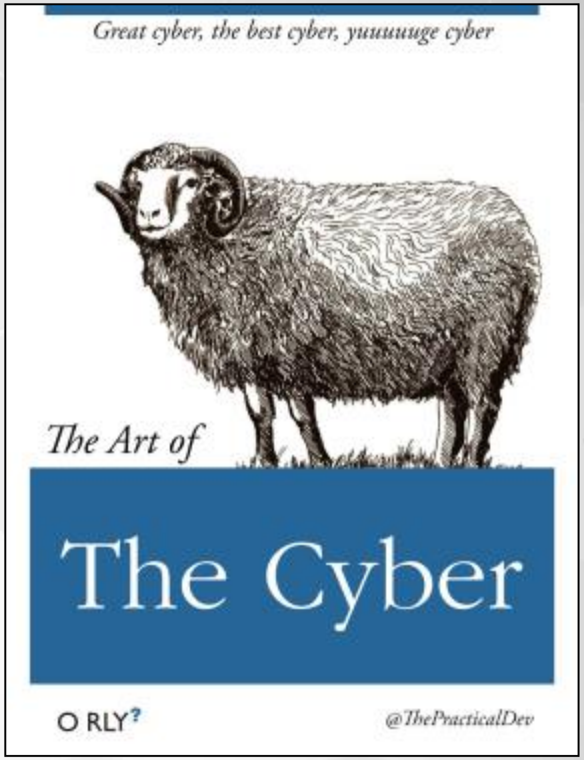
- Large organization
- Brownfield
- Large number of applications
  - Some New
  - Some Old
  - Some Decrepit
  - Internet, Extranet and Intranet facing
  - All different
  - Got micro services too!



OWASP
Open Web Application Security Project

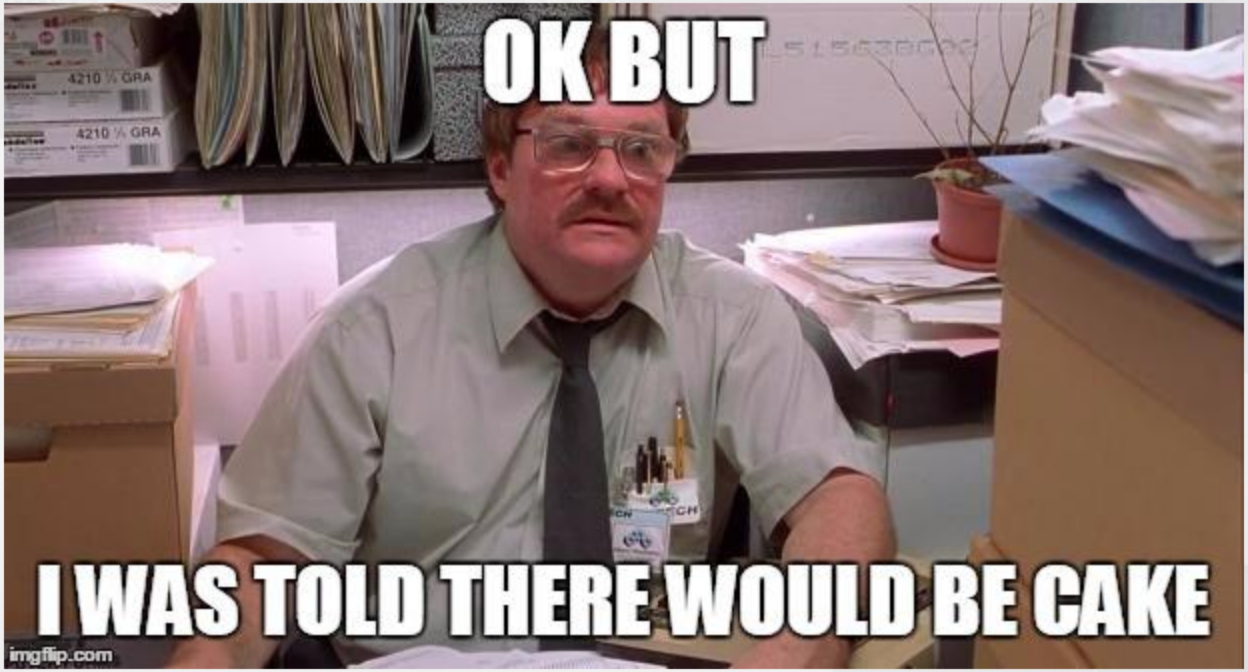
Security Challenges in DevOps Orgs





Security: we are [usually] the last to know… and first to respond.

Benefits for DevOps and Security



OWASP
Open Web Application Security Project

Container Security Benefits – Cake Icing

- Standard, hardened infrastructure on releases
- Pipeline integration moves security left
- Read-only containers = Application Whitelisting
- Continuous (re)deploying from known good
- No humans in production – SSH turned off
- Patching improvements
- Complete record of changes

Container DevOps Benefits – The Cake Layers

- Containers <u>**will run**</u> the same
  - Packaged OS + Dependencies + App run
  - Reduces “***worked on *my* machine***”
  - Portable to deploy across hosts

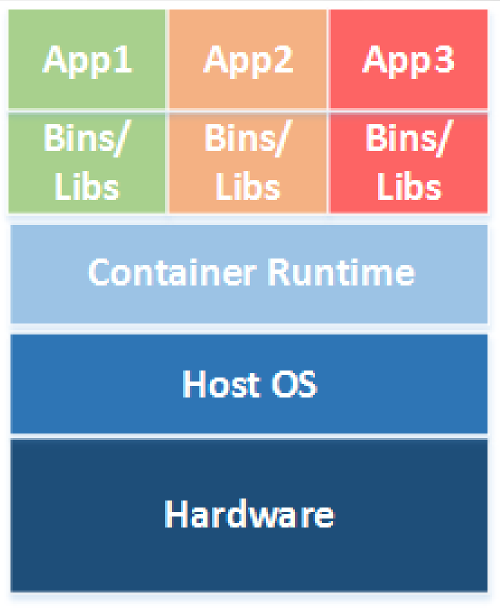
- Produces:
  - Higher Developer Productivity
  - Patches baked before tests in releases
  - More frequent Release schedule
  - Increased Server utilization on hosts

My [Enterprise] Container Journey

- Understanding the basic tech
- My first [trusted] container
- Moving security upstream
- Avoid the container failboat

Understanding the basic tech

- Uses OS level virtualization
- Shares host OS resources + kernel <u>at runtime</u>
- Isolation applies for processes, filesystem, & network via OS kernel
- Images sealed w/ crypto hash
- Typically Copy-on-write (CoW) *layered* file system



*"A construct designed **to package and run** an **application or its` components** running **on a shared Operating System.**"*
- *NIST Pub 800-180 (draft), "*NIST Definition of Microservices, Application Containers and System Virtual Machines"

My first [trusted] container

```
FROM centos7.1.1503

MAINTAINER USCIS <noreply@uscis.dhs.gov>

# Set global environment variables for base container
ENV http_proxy={INTERNET_PROXY}/ \
    https_proxy={INTERNET_PROXY}/ \
    no_proxy={NO_PROXY} \
    NEXUS={NEXUS_SERVER}

# Fix cache issues through proxy: https://sites.google.com/site/kbinstuff/yum-and-proxies
# And patch base container with latest upgrades (upgrade runs update)
RUN echo "http_caching=packages" >> /etc/yum.conf \
    && yum upgrade -y \
    && yum install epel-release wget -y \
    && yum clean all -y

ARG BUILDER_HOST
ARG DOCKER_IMAGE
ARG GIT_REPO
ARG GIT_BRANCH
ARG GIT_HASH
ARG DOCKER_TAG
ARG BUILD_DATE
ARG BASE_SHA
ARG CREATED
ARG VENDOR
ARG VERSION

LABEL com.docker.hub.base.version="${VERSION:-UNKNOWN}" \
      com.docker.hub.base.image="${VENDOR:-UNKNOWN}" \
      com.docker.hub.base.build-date="${CREATED:-UNKNOWN}" \
      com.docker.hub.base.digest="${BASE_SHA:-UNKNOWN}" \
      gov.dhs.uscis.base.image="${DOCKER_IMAGE:-centos7-base}" \
      gov.dhs.uscis.base.builder="${BUILDER_HOST:-UNKNOWN}" \
      gov.dhs.uscis.base.git.repo="${GIT_REPO:-USCIS/dockerfiles}" \
      gov.dhs.uscis.base.git.branch="${GIT_BRANCH:-UNKNOWN}" \
      gov.dhs.uscis.base.git.sha="${GIT_HASH:-UNKNOWN}" \
      gov.dhs.uscis.base.version="${DOCKER_TAG:-0.0.x}" \
      gov.dhs.uscis.base.build-date="${BUILD_DATE:-UNKNOWN}"
```

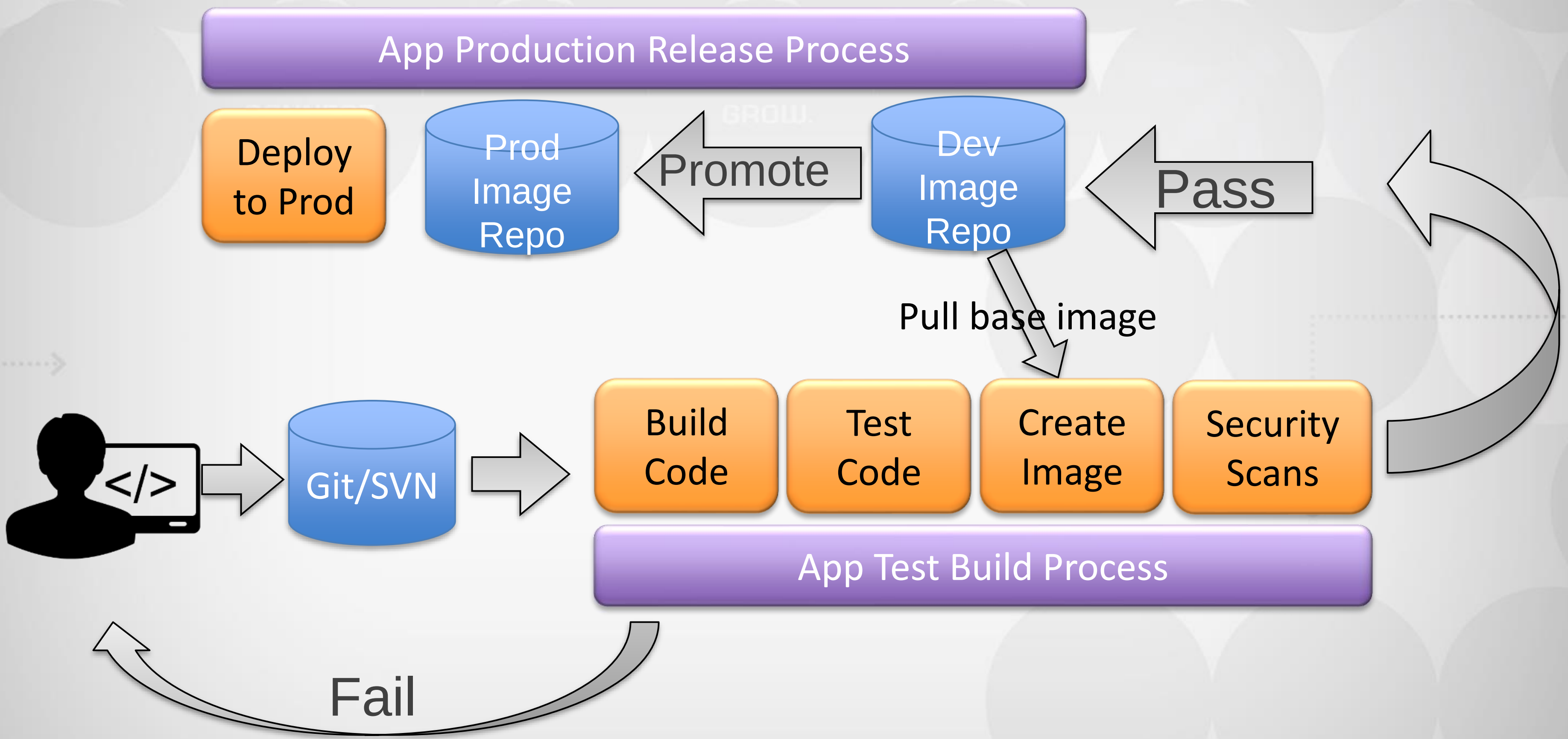
Base source OS

Env app vars injection

OS patching + build

Metadata tagging

Moving Container Security Upstream

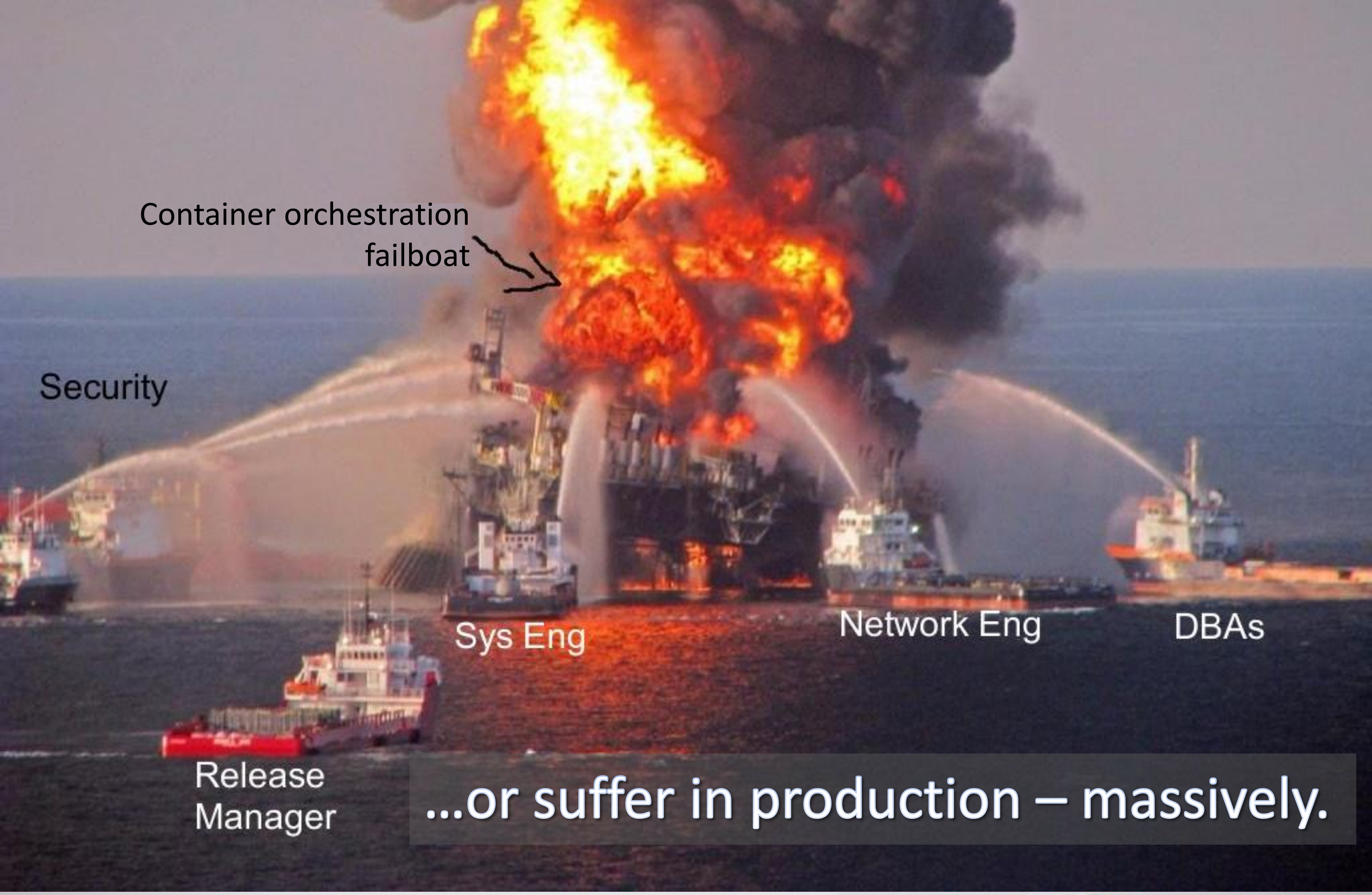


Avoiding the container failboat...

- Running as root (for all things)
- Unbounded CPU + memory runtime
- Writing persistent data to container filesystem
- Unsecured virtual network stack
- Mixing workloads of different threat postures

<u>Break the tech</u> *to learn the tech*

(...in a controlled non-prod environment – of course!)



…or suffer in production – massively.

Learning Secure App Containers

Local Container Development

Container Orchestration

1. Center for Internet Security Benchmarks

- Community consensus driven + CIS PM managed
- Defines Level 1 (general) & Level 2 (sensitive info) processing controls
- Host OS + Container Daemon + Container Image + Container Runtime
- Available for Cloud, OSes, Docker, & Kubernetes

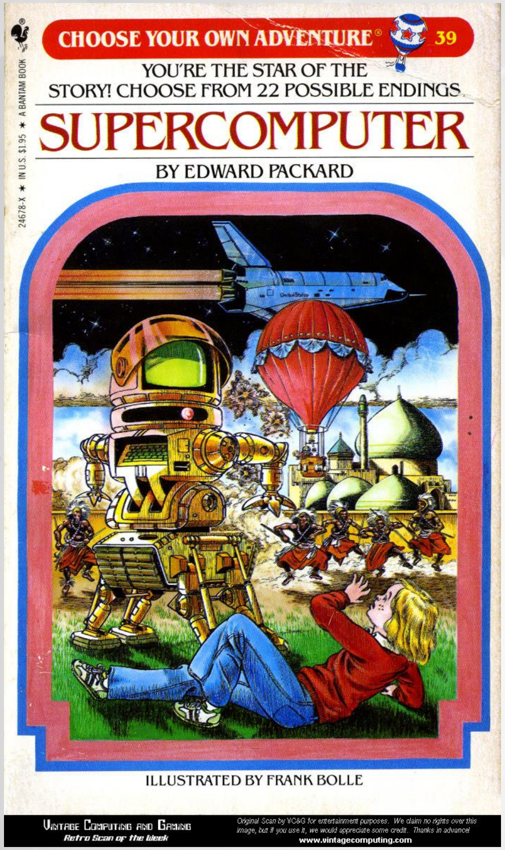
2. Develop threat model for app risk postures

- Processes executing on container and hosts
- Data being processed (intermix on hosts? Sensitive? Access controls?)
- Sources of connections (internal, external, behind proxy? Inputs? Outputs?)

3. Determine expected container app ops

- App logs to SIEM (audit, error, info level)
- Data persistence (host? net share? Data SaaS?)
- Health checks (simple vs. complex)
- Restart vs. destroy on non-responsive containers

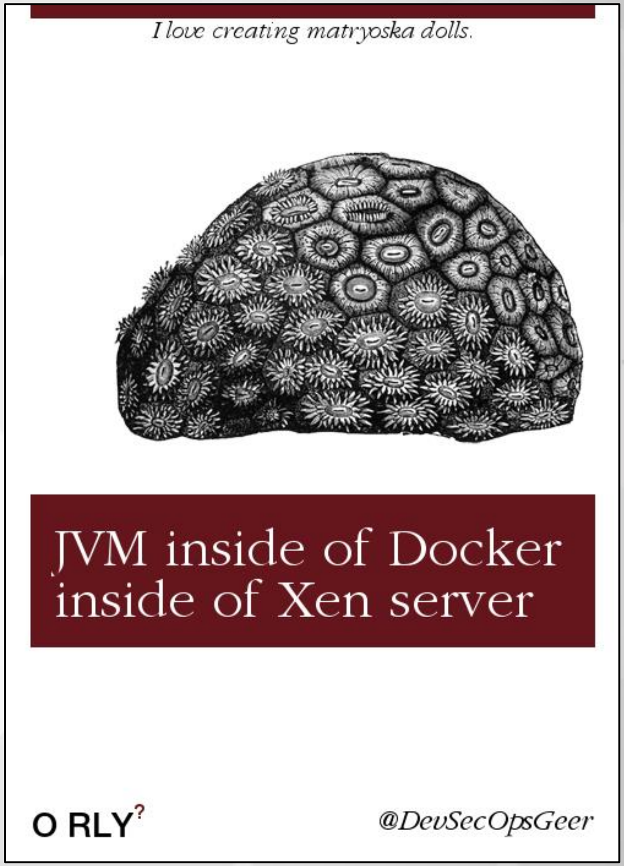
4. Runtime: Choose your own adventure



- Run the stack myself?
- Have a vendor run the stack for me?
- Hybrid model?

My Container Security Maturity Model

- Purposely build security from day 1
- Focus on basic critical items 1st to reduce major vulns
- Mature your #ContainerOps into rest of industry benchmarks
- Optimize and tweak to your organization policies and needs



Container Host Security Management

Maturity	Objectives
1: Initial	• Use a standard out-of-the-box server operating system • Use standalone container daemons on local hosts
2: Managed	• Use of networked container daemons • Use default kernel calls and namespaces • Enforce host and container logging
3: Defined	• Command + control of host daemons • Scaling homogenesis hosts based on orchestration app loads • Establishing logical groups of hosts to process sensitive app info
4: Quantified	• Restricting kernel calls by containers to host • Minimalistic hosts to operate only container daemons
5: Optimizing	• Reducing surface attack areas on hosts (i.e. no SSH access) • Removing container binding to certain host dependencies • Chaos Monkey resiliency when taking hosts out

Container Image Security Management

Maturity	Objectives
1: Initial	• Scan for CVEs in OS, Package Managers, and App Dependencies • Establish series of trusted base images for DevOps use • No root users in container OS image
2: Managed	• Establish internal registries for non-prod and prod use • Build series of base and framework images • Metadata tag releases beyond version number
3: Defined	• Chain app image rebuilds back to base + framework images • Image & compliance scans to break builds and stop runtimes
4: Quantified	• Automated redeployments on new CVE drops from dev to prod • Monitor processes + hashes, network, and kernel interactions • Matching found runtime threats to indicators of compromise (IoCs)
5: Optimizing	• Customized whitelist of kernel namespace and syscalls per app • Exporting runtime threat results to OASIS STIX for kill chain analysis

Container Data & Ops Management

Maturity	Objectives
1: Initial	• Basic CI/CD pipeline processes to build and push releases • Avoid data writes to container file system (except tempfs) • Set CPU and memory runtime min and max limits
2: Managed	• Basic autoscaling containers framework on same hosts • Data writes to managed container volumes on daemon host • Restrict access to “hand jamming” deployments in orchestration
3: Defined	• Enabling read-only containers to reduce attack surface • Data volumes are dynamically managed under orchestration
4: Quantified	• Use mature data management patterns for data persistence • Application secrets are injected at runtime as environment vars
5: Optimizing	• Custom runtime defenses based on application risk posture • Application secrets are accessed “just-in-time” for runtime • Tracking container runtime drift of processes, network, and kernel

OWASP
Open Web Application Security Project

Further Reading

- NIST Special Publication 800-190: Application Container Security Guide (Final) https://csrc.nist.gov/publications/detail/sp/800-190/final
- CIS Security Benchmarks https://www.cisecurity.org/cis-benchmarks/
- NCC Group's "Understanding and Hardening Linux Containers v1.1" https://www.nccgroup.trust/globalassets/our-research/us/whitepapers/2016/april/ncc_group_understanding_hardening_linux_containers-1-1.pdf