Running at Light Speed: Cloud Native Security Patterns

Jack Mannino
OWASP Netherlands
January 2019
Hi, How is Everybody? Good. Great.

CEO @ nVisium since 2009
Doesn’t mix coffee and sugar
Drinks beer the right way - never Budweiser
Mostly codes in Scala + Go
Cloud Native Characteristics

Containerized
Microservices
Dynamically Managed Orchestration
Declarative
Telemetry & Health Reporting
Resiliency
Cloud Native Secure Architecture

- Container Isolation
- Control Plane Hardening
- Network Segmentation
- Encrypted Communications
- Authentication (container & cluster-level)
- Authorization & Access Control
- Secrets Management
- Logging & Monitoring
Who’s Job is it Anyway?

- **DevOps Teams**
  - Applications

- **Platform Team**
  - Shared Services:
    - databases | messaging | clustering & coordination | logging | application monitoring | user management & security | ...
  - Container Services:
    - App & Policy definitions | Container Lifecycle mgmt. | Scheduling | Infrastructure Automation | Service Discovery | Load Balancing | ...

- **IT Ops Team**
  - Infrastructure Services:
    - compute | network | storage
Isolating Containerized Workloads

Control Plane & Core Components

Control Plane manages the cluster’s state and schedules containers.

A privileged attack against a control plane node or pod can have serious consequences.

Managed container orchestration services generally abstract away the control plane for you.
Reconciler Pattern

Fix configuration drift

Event-driven model for updating/patching

Decrease the mean-time-to-fix (MTTF)

Kubernetes controller loops & scheduler handle updates
Spoiler: Containers Aren’t Sandboxes

Shared host resources & kernel

Often relies on rule-based execution policies (Seccomp, SELinux, AppArmor) for isolation
Container Privilege Escalation

CVE-2018-1000400
Kubernetes’ Container Runtime Interface (CRI) was vulnerable to a privilege context switching error due to a capability granting issue.

CVE-2016-3697
runC was vulnerable to privilege escalation by starting a Docker container with an arbitrary UID.
The Gateway Drug

FROM golang:1.10.2
MAINTAINER Jack Mannino <jack@nvisium.com>

USER root

ENV password s3curitah1

RUN apt-get update && apt-get install -y apt-transport-https

RUN mkdir /app
ADD . /app/
WORKDIR /app
RUN go build main -o main.
CMD ["/app/main"]

Container Engine Process Models

- **rkt** 1.0+
- **Docker** < 1.11.0
- **Docker** 1.11.0+

https://coreos.com/rkt/docs/latest/rkt-vs-docker-process-model.png
Container Isolation Models

Via cgroups & namespaces
  Docker, Rkt, LXC

User-space kernels
  gVisor

Hypervisor/VM
  Kata Containers, Clear Containers
## How They Stack Up

<table>
<thead>
<tr>
<th>Available Container Security Features, Requirements and Defaults</th>
<th>LXC 2.0</th>
<th>Docker 1.11</th>
<th>CoreOS Rkt 1.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Feature</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>User Namespaces</td>
<td>Default</td>
<td>Optional</td>
<td>Experimental</td>
</tr>
<tr>
<td>Root Capability Dropping</td>
<td>Weak Defaults</td>
<td>Strong Defaults</td>
<td>Weak Defaults</td>
</tr>
<tr>
<td>Procfs and Sysfs Limits</td>
<td>Default</td>
<td>Default</td>
<td>Weak Defaults</td>
</tr>
<tr>
<td>Cgroup Defaults</td>
<td>Default</td>
<td>Default</td>
<td>Weak Defaults</td>
</tr>
<tr>
<td>Seccomp Filtering</td>
<td>Weak Defaults</td>
<td>Strong Defaults</td>
<td>Optional</td>
</tr>
<tr>
<td>Custom Seccomp Filters</td>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>Bridge Networking</td>
<td>Default</td>
<td>Default</td>
<td>Default</td>
</tr>
<tr>
<td>Hypervisor Isolation</td>
<td>Coming Soon</td>
<td>Coming Soon</td>
<td>Optional</td>
</tr>
<tr>
<td>MAC: AppArmor</td>
<td>Strong Defaults</td>
<td>Strong Defaults</td>
<td>Not Possible</td>
</tr>
<tr>
<td>MAC: SELinux</td>
<td>Optional</td>
<td>Optional</td>
<td>Optional</td>
</tr>
<tr>
<td>No New Privileges</td>
<td>Not Possible</td>
<td>Optional</td>
<td>Not Possible</td>
</tr>
<tr>
<td>Container Image Signing</td>
<td>Default</td>
<td>Strong Defaults</td>
<td>Default</td>
</tr>
<tr>
<td>Root Interation Optional</td>
<td>True</td>
<td>False</td>
<td>Mostly False</td>
</tr>
</tbody>
</table>

Just Use the Defaults != Turn It Off

seccomp=unconfined for a container in a kubernetes pod? Or ...
https://stackoverflow.com/.../seccomp-unconfined-for-a-container-in-a-kubernetes-po...
Apr 3, 2016 - I'm running kubernetes with docker 1.10 and I want to run a container with --security-opt seccomp=unconfined. I understand from ...

Set seccomp to unconfined in docker-compose - Stack Overflow
Sep 4, 2017 - I need to be able fork a process. As i understand it i need to set the security-opt. I have tried doing this with docker command and it works fine.

c - warning: Error disabling address space randomization ...
https://stackoverflow.com/.../warning-error-disabling-address-space-randomization-op...
3 answers
Oct 11, 2017 - If you're using Docker, you probably need the --security-opt seccomp=unconfined option (as well as enabling prace): docker run ...
Control Groups & Namespaces

By UID, GID, and PID

Isolation for what you can use (cgroups) and what you can see (namespaces)

Not all objects are namespaced (time, keyctl), however default seccomp profiles blocks these syscalls
What Am I Shipping?

Base Image Management

- Focus on keeping the attack surface small
- Use base images that ship with minimal installed packages and dependencies
- Use version tags vs. image:latest
- Use images that support security kernel features (Seccomp, AppArmor, SELinux)

```bash
$ grep CONFIG_SECCOMP= /boot/config-
$ (uname -r)
$ cat /sys/module/apparmor/parameters/enabled
```
Build Integrity & Attestation

- Ensures integrity of the images and publisher attestation
- Sign to validate pipeline phases
- Example – Docker Content Trust & Notary, GCP’s Binary Authorization
- Consume only trusted content for tagged builds
Seccomp
Linux kernel feature that reduces the attack surface by filtering dangerous syscalls

Applied via metadata annotations at the pod or container level

Docker provides a reasonably secure default profile to leverage (security quick win!)
AppArmor

Mandatory access control (MAC) for Linux.

Reduces the attack surface by whitelisting capabilities, restricting network access, file permissions, and more.

Similar to Seccomp, container runtimes provide default profiles with quick wins (especially Docker and LXC).
Restricting Capabilities

- Circa 2003, root privileges were broken into a subset of capabilities.
- This feature enables us to reduce the damage a compromised root account can do.
- Docker default profile allows 14 of 40+ capabilities.

Open Container Initiative (OCI) spec restricts this even further:

- AUDIT_WRITE
- KILL
- NET_BIND_SERVICE

Docker Default Capabilities

- CHOWN
- DAC_OVERRIDE
- FOWNER
- FSETID
- KILL
- SETGID
- SETUID
- SETPCAP
- NET_BIND_SERVICE
- NET_RAW
- SYS_CHROOT
- MKNOD
- AUDIT_WRITE
- SETFCAP
Limiting Privileges

More often than not, your container does not need root!!!

Limit access to underlying host resources (network, storage, or IPC)

Example – Ping command requires CAP_NET_RAW

```
docker run -d --cap-drop=all --cap-add=net_raw my-image
```

```
securityContext:
  allowPrivilegeEscalation: false
capabilities:
drop:
  - ALL
add:
  - "NET_RAW"
runAsNonRoot: true
runAsUser: 1000
```
User Namespaces

Supported by modern kernels and leading container tech

Remaps UID from the container to an unprivileged high-number UID on the host

Enable

```
ExperimentalHostUserNamespaceDefaultingGate
```

dockerd -usersns-remap="someuser:someuser"
Rootless Containers
Unprivileged user that can’t ask for more privileges
Cannot install packages
Rootless containers are built via filesystem snapshots
runC and others support rootless, but upstream support has been limited
Upstream Orchestration Support

Cool story, how do I actually use this?

Kubernetes 1.12 supports the procMount security context attribute

Allows paths in the container’s /proc to not be masked

type ContainerSecurityContextAccessor interface {
  Capabilities() *api.Capabilities
  Privileged() *bool
  ProcMount() api.ProcMountType
  SELinuxOptions() *api.SELinuxOptions
  RunAsUser() *int64
  RunAsNonRoot() *bool
  ReadOnlyRootFilesystem() *bool
  AllowPrivilegeEscalation() *bool
}
No New Privileges

Introduced in Linux 3.5, uses the `no_new_privs` kernel flag

Breaks setuid and setgid

Docker `--no-new-privileges`
K8s via the `allowPrivilegeEscalation` security context constraint
Authentication

- Authentication occurs at several layers
  - Authenticating API users.
  - Authenticating nodes to the cluster.
  - Authenticating services to each other.
  - Webhook endpoints should authenticate requests.
- Kubernetes provides several authenticator options, each with their own strengths and tradeoffs.
- *By default, Kubernetes uses a self-signed certificate infrastructure.*
Implementation Flaw - Account Reuse

By default, K8s uses the namespace default service account if you don’t define one for your pod.

Elevate privileges against other services in the same namespace or potentially at the cluster level.
Run Commands via K8s API

Even if network traffic is blocked, K8s API lets us execute shell commands.

This is allowed across shared accounts or users with **create** privileges for **pods/exec**

https://k8s/api/v1/namespaces/owned1/pods/pod-xyz-xyz/exec?command=/bin/bash&stdin=true&stderr=true&stdout=true&tty=true
Fixing the Problem
Always use a unique service account per pod!

```
kubectl create serviceaccount s1 --namespace="prod"
```
This can be set at the orchestration level

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  serviceAccountName: s1
```
Don’t Share Anything From the Host

✓ Mounts & Volumes
✓ Host namespaces
✓ Host Network
✓ Host PID
Authorization

• After we’ve authenticated a subject, we need to limit the resources they can view or manipulate.

• Kubernetes provides two approaches to authorizing api-server cluster actions for users and services.
  • Role-Based Access Control (RBAC)
  • Attribute-Based Access Control (ABAC)

• Kubelet API uses the Node Authorizer to control access.

• Webhook endpoints can also be authorized independently.
Role-Based Access Control

- Our goal is to restrict what each user and service can do against a running cluster.
- Roles can be granted at the cluster-level or namespace-level.
  - `ClusterRoleBinding` – applies to all namespaces across the cluster
  - `RoleBinding` – applies only within a single namespace
- Kubernetes also enables default roles for common role types and for system services.
- RBAC policies are cumulative; Nothing is taken away, only given.
Create Roles & Bindings

kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: production
  name: read-pods
rules:
- apiGroups: [""] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]

Create a role

kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: production
subjects:
- kind: ServiceAccount
  name: joe-dev # Name is case sensitive
roleRef:
  kind: Role #this must be Role or ClusterRole
  name: read-pods # name of the Role or ClusterRole
  apiGroup: rbac.authorization.k8s.io

Bind the role to an explicit namespace
Controller Pattern

Used to read an object’s specification and mutate its status

*Enforce security at cluster admission boundaries*

Pluggable architecture

Kubernetes implements a controller for most objects
Admission Controllers

Kubernetes provides hooks to check & fix containers at runtime.

PodSecurityPolicy enforces a set of security constraints at the cluster level for pods being admitted.

Enable the `PodSecurityPolicy` admission controller via `kube-apiserver: --admission-controlPodSecurityPolicy`.
Designing a PodSecurityPolicy

One or more policies can be applied per cluster

When multiple policies are defined, K8s selects a policy in the following order:

1. Policies that validate the pod without altering it are used first.
2. If it is a new pod, the policy is selected alphabetically.
3. The policy is rejected if it’s a pod update and mutation is required.
Designing a PodSecurityPolicy

apiVersion: extensions/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restrictive-pod-security-policy
  annotations:
    seccomp.security.alpha.kubernetes.io/defaultProfileName: docker/default
    apparmor.security.beta.kubernetes.io/allowedProfileNames: 'runtime/default'
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: 'docker/default'
    apparmor.security.beta.kubernetes.io/defaultProfileName: 'runtime/default'

spec:
  privileged: false
  allowPrivilegeEscalation: false
  requiredDropCapabilities: - ALL
  volumes:
  - 'configMap'
  - 'emptyDir'
  - 'projected'
  - 'secret'
  - 'downwardAPI'
  - 'persistentVolumeClaim'
  hostNetwork: false
  hostIPC: false
  hostPID: false
  runAsUser:
    rule: MustRunAsNonRoot
  seLinux:
    rule: RunAsAny
  supplementalGroups:
    rule: MustRunAs'
    ranges:
      - # Forbid adding the root group.
        min: 1
        max: 65535
  fsGroup:
    rule: MustRunAs'
    ranges:
      - # Forbid adding the root group.
        min: 1
        max: 65535
  readOnlyRootFilesystem: true

PodSecurityPolicy components
- AppArmor
- Capabilities
- Host Namespaces
- Privilege Escalation
- Seccomp
- SELinux
- Users and Groups
- Volumes and Filesystems
Apply a PodSecurityPolicy

To ensure that a pod can access a PodSecurityPolicy, it must be granted RBAC access.

```yaml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: authenticated-users-psp
rules:
- apiGroups: ['policy']
  resources: ['podsecuritypolicies']
  verbs: ['use']
  resourceNames:
    - psp-1
    - psp-2
```
Sidecar Pattern

Decomposition pattern

Extends a container’s capabilities & security boundaries

Introduces complexity & tighter coupling

Generally requires centralized
Ambassador Pattern

Out-of-process proxy

Deployed as a sidecar or daemon

*Update legacy apps without changes*
Avoid when features can’t be generalized & require deeper integration with the client application.

**Key Boundary**
Consider whether to use a single shared instance or an instance for each client.
Service Mesh Pattern

Mesh of sidecars with centralized management

Popular implementations
- Envoy
- Istio
- Consul

Lift security controls out of containers

Automated Injection
Secrets Management

**Docker**

docker run --it --e "DBUSER=dbuser" --e "DBPASSWD=dbpasswd"
mydbimage

echo <secret> | docker secret create some-secret

**Kubernetes**

kubectl create secret generic db-user-pw --from-file=./username.txt --from-file=./password.txt

kubectl create -f ./secret.yaml
**Nothing is Perfect**

---

**Overview**

- **Namespace:** default

**Details**

- **Name:** jack-pass
- **Namespace:** default
- **Creation time:** 2017-10-19T18:36

**Data**

- **password.txt:** jack555
- **username.txt:** admin
Beware of Plain Text Storage

Prior to 1.7, secrets were stored in plain text at-rest.

As of v1.7+, k8s can encrypt your secrets in etcd.

Not perfect at all, either.

```yaml
kind: EncryptionConfig
apiVersion: v1
resources:
  - resources:
    - secrets
  providers:
    - aescbc:
      keys:
        - name: key1
          secret: YOURKEYHERE
      - identity: {}
```
Dynamic Secrets

Example – Retrieve & Mount a Secret

```bash
command:
  - "sh"
  - "-c"
  - >
    X_VAULT_TOKEN=$(cat /etc/vault/token);
    VAULT_LEASE_ID=$(cat /etc/app/creds.json | jq -j '.lease_id');
    while true; do
      sleep 3600;
    done

lifecycle:
  preStop:
    exec:
      command:
        - "sh"
        - "-c"
        - >
          X_VAULT_TOKEN=$(cat /etc/vault/token);
          VAULT_LEASE_ID=$(cat /etc/app/creds.json | jq -j '.lease_id');

volumeMounts:
  - name: app-creds
    mountPath: /etc/app
  - name: vault-token
    mountPath: /etc/vault
```
Conclusion

Think about security early and anticipate future growth

Focus on logical and organizational structure and codify it into your environment

Enable your engineers to move fast, but protect them from themselves

Apply security controls at the layers that make the most sense
Keep in Touch

Jack Mannino

Twitter @jack_mannino

Linkedin - https://www.linkedin.com/in/jackmannino

Email - Jack@nvisium.com