09. – 12.12.2019
Frankfurt am Main

Johannes Schnatterer, Cloudogu GmbH

Kubernetes-Security: 3 Dinge, die jeder Entwickler wissen sollte

#ittage
@jschnatterer

@haveibeenpwned

https://haveibeenpwned.com/PwnedWebsites

126
500px
7K7K
8fit
Atlas
BTC
EPIC
Elance
Forbes
flvs
KM.RU
mate1
NETLOG
NETSHOES
NETPROPHEX
SIAE
SHEIN
SEPHORA
PLEX
piZap
TESCO
VOID.TO
vtech
V-Tight Gel
WANELO
Bell
Canon
CASH CRITE
Chegg
DISQUS
EyeEm
imgur
LOOKBOOK
LOUNGE BOARD
LITTLE MONSTERS
SONY
TRAI
tumblr.
whitepages
WHMCS
AERSERV
Bitly
CLUB
CLUB
CLUB
CLUB
Dropbox
HAUTELOOK
JustDate
MALL.CZ
PP.
gym
stockX
SUMO
Xat
XKCD
XSplit
Yandex
yatra
YOUKU
younow
zomato
zoosk

Plenty of security options

securityContext runAsNonRoot runAsUser privileged procMount
allowPrivilegeEscalation readOnlyRootFilesystem
PodSecurityPolicy RBAC NetworkPolicy
seccomp Linux Capabilities AppArmor SELinux
Falco Open Policy Agent gVisor Kata Containers
Nabla Containers Service Mesh KubeSec KubeBench

3 Things Every Developer Should Know About K8s Security

0. Role Base Access Control (RBAC)

https://memegenerator.net/instance/83566913/homer-simpson-boring

- RBAC active by default since K8s 1.6
- ... but not if you migrated!

- Try

```
curl --cacert /var/run/secrets/kubernetes.io/serviceaccount/ca.crt \
-H "Authorization: Bearer $(cat /var/run/secrets/kubernetes.io/serviceaccount/token)"
https://${KUBERNETES_SERVICE_HOST}/api/v1/secrets
```

- If not needed, disable access to K8s API

```
automountServiceAccountToken: false
```

Demo

- legacy-authz
- RBAC

1. Network Policies (netpol)

A "firewall" for communication between pods.

- Applied to pods
  - within namespace
  - via labels
- Ingress / egress
  - to/from pods (in namespaces) or CIDRs (egress only)
  - for specific ports (optional)
- Enforced by the CNI Plugin (e.g. Calico)
- ⚠️ No Network Policies: All traffic allowed

📌 Helpful to get started

- https://github.com/ahmetb/kubernetes-network-policy-recipes
- Securing Cluster Networking with Network Policies - Ahmet Balkan
  https://www.youtube.com/watch?v=3gGpMmYeEO8
- Interactively describes what a netpol does:

```
kubectl describe netpol <name>
```

Recommendation: Whitelist ingress traffic

In every namespace except `kube-system`:

- Deny ingress between pods,
- then whitelist all allowed routes.

Advanced: ingress to `kube-system`

⚠️ Might stop the apps in your cluster from working

Don't forget to:

- Allow external access to ingress controller
- Allow access to kube-dns/core-dns to every namespace

Advanced: egress

- Verbose solution:
  - Deny egress between pods,
  - then whitelist all allowed routes,
  - repeating all ingress rules. 🙁
- More pragmatic solution:
  - Allow only egress within the cluster,
  - then whitelist pods that need access to internet.

🚧 Net pol pitfalls

- Whitelisting monitoring tools (e.g. Prometheus)
- Restart might be necessary (e.g. Prometheus)
- No labels on namespaces by default
- `egress` more recent than `ingress` rules and less sophisticated
- Policies might not be supported by CNI Plugin.

  Testing!

  🌐 https://www.inovex.de/blog/test-kubernetes-network-policies/

More Features?

- Proprietary extensions of CNI Plugin (e.g. cilium or calico)
- Service Meshes: similar features, also work with multiple clusters

  ➜ different strengths, support each other

  🌐 https://istio.io/blog/2017/0.1-using-network-policy/

Demo

- nosqlclient
- web-console

🎁 Wrap-Up: Network Policies

My recommendations:

- Ingress whitelisting in non-`kube-system` namespaces
- Use with care
  - whitelisting in `kube-system`
  - `egress` whitelisting for cluster-external traffic

2. Security Context

Defines security parameter per pod/container ➜ container runtime

📌 Secure Pods - Tim Allclair

🎥 https://www.youtube.com/watch?v=GLwmJh-j3rs

Recommendations per Container

```yaml
apiVersion: v1
kind: Pod
metadata:
  annotations:
    seccomp.security.alpha.kubernetes.io/pod: runtime/default
spec:
  containers:
  - name: restricted
    securityContext:
      runAsNonRoot: true
      runAsUser: 100000
      runAsGroup: 100000
      readOnlyRootFilesystem: true
      allowPrivilegeEscalation: false
      capabilities:
        drop:
          - ALL
  enableServiceLinks: false
```

Recommendation per Container in Detail

Enable seccomp

- Enables e.g. docker's seccomp default profile that block 44/~300 Syscalls
- Has mitigated Kernel vulns in past and might in future 🔮 🌐 https://docs.docker.com/engine/security/non-events/
- See also k8s security audit: 🌐 https://www.cncf.io/blog/2019/08/06/open-sourcing-the-kubernetes-security-audit/

Run as unprivileged user

- `runAsNonRoot: true`

  Container is not started when the user is root
- `runAsUser` and `runAsGroup` > 10000
  - Reduces risk to run as user existing on host
  - In case of container escape UID/GID does not have privileges on host
- Mitigates vuln in `runc` (used by Docker among others)

  https://kubernetes.io/blog/2019/02/11/runc-and-cve-2019-5736/

No Privilege escalation

- Container can't increase privileges
- E.g. `sudo`, `setuid`, Kernel vulnerabilities

Read-only root file system

- Starts container without read-write layer
- Writing only allowed in volumes
- Config or code within the container cannot be manipulated
- Perk: More efficient (no CoW)

Drop Capabilities

- Drops even the default caps:
  https://github.com/moby/moby/blob/3152f94/oci/caps/defaults.go
- Mitigates `CapNetRaw` attack - DNS Spoofing on Kubernetes Clusters
  https://blog.aquasec.com/dns-spoofing-kubernetes-clusters

Bonus: No Services in Environment

- By default: Each K8s service written to each container's env vars
  ➜ Docker Link legacy, no longer needed
- But convenient info for attacker where to go next

Security context pitfalls

Read-only root file system

Application might need temp folder to write to

- Run image locally using docker, access app
  - Run automated e2e/integration tests
- Review container's read-write layer via

```
docker diff <containerName>
```

- Mount folders as `emptyDir` volumes in pod

Drop Capabilities

Some images require capabilities

- Find out needed Caps locally:

```
docker run --rm --cap-drop ALL <image>
# Check error
docker run --rm --cap-drop ALL --cap-add CAP_CHOWN <image>
# Keep adding caps until no more error
```

- Add necessary caps to k8s resource
- Alternative: Find image with same app that does not require caps, e.g. `nginxinc/nginx-unprivileged`

Run as unprivileged user

- Non-root verification only supports numeric user. 🙄
  - `runAsUser: 100000` in `securityContext` of pod or
  - `USER 100000` in `Dockerfile` of image.
- Some official images run as root by default.
  - Find a **trusted** image that does not run as root

    e.g. for mongo or postgres:

    https://hub.docker.com/r/bitnami/
  - Derive from the original image and create your own non-root image

    e.g. nginx: https://github.com/schnatterer/nginx-unpriv

- UID 100000 might not have permissions. Solutions:
  - Init Container sets permissions for PVCs
  - Permissions in image ➜ `chmod`/`chown` in `Dockerfile`
- Application requires user for UID in `/etc/passwd`
  - New image that contains a user for UID e.g. `100000` or
  - Create `/etc/passwd` in init container and mount into app container
- `runAsGroup` - beta from K8s 1.14. Before that defaults to GID 0 ☹️

  🌐 https://github.com/kubernetes/enhancements/issues/213

Tools

Find out if your cluster adheres to these and other good security practices:

- controlplaneio/kubesec - managable amount of checks
- Shopify/kubeaudit
  - a whole lot of checks,
  - even deny all ingress and egress NetPols and AppArmor Annotations

➜ Be prepared for a lot of findings
➜ Create your own good practices

Demo

Security Context & PSP
Namespace 'wild-west'
nginx
nginxinc/nginx-unprivileged
docker-sudo

🎁 Wrap-Up: Security Context

My recommendations:

- Start with least privilege
- Only differ if there's absolutely no other way

3. Pod Security Policies (PSP)

- enforce security context cluster-wide
- additional options for blocking pods trying to
  - enter node's Linux namespaces (net, PID, etc.)
  - mounting docker socket,
  - binding ports to nodes,
  - starting `privileged` containers
  - etc.
- more effort than security context and different syntax 🙁

➜ Still highly recommended!

Recommendation

https://github.com/cloudogu/k8s-security-demos/blob/master/4-pod-security-policies/demo/01-psp-restrictive.yaml

Too much ground to cover for 45 min!

Summary

- Enable RBAC
- Don't allow arbitrary connections between pods, e.g. via NetPols
- Start with least privilege for your containers
  - using either securityContext or
  - PodSecurityPolicy

What for?

- Increase security
- Reduce risk of data breach
- Don't end up on @haveibeenpwned

Johannes Schnatterer

Cloudogu GmbH

https://cloudogu.com/schulungen

K8s Security series on JavaSPEKTRUM starting 05/2019

See also https://cloudogu.com/blog

@jschnatterer

@cloudogu

Demo Source: https://github.com/cloudogu/k8s-security-demos