

Speaker notes

- about 400 breaches

```
tmpFile=$(mktemp)
curl -sS https://haveibeenpwned.com/PwnedWebsites > ${tmpFile}

cat ${tmpFile}
| grep PwnedLogos
| sed 's//'
| xargs wget --no-clobber
```

Yields duplicate Email.pngs

```
cat ${tmpFile}
| grep PwnedLogos
| grep -vE 'List.png|Email.png'
| sort
| sed -r 's//'
> 01b-pawned-logos.md
```


**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)

- 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

Speaker notes

- `curl -k https://$KUBERNETES_SERVICE_HOST/api/v1/namespaces/default/secrets/web-console -H "Authorization: Bearer $(cat /var/run/secrets/kubernetes.io/serviceaccount/token)"`
- `k create rolebinding web-console --clusterrole admin --serviceaccount default:web-console`
- [Demo Script](#)
- [plantUml src](#)

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

Speaker notes

- Allow external access to ingress controller
(otherwise no more external access on any cluster resource)
- Allow access to kube-dns/core-dns to every namespace
(otherwise no more service discovery by name)

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/>

Speaker notes

- Matching both pods and namespace needs k8s 1.11+
- Restart might be necessary for the netpol to become effective
- In order to match namespaces, labels need to be added to the namespaces, e.g.

```
kubectl label namespace/kube-system namespace=kube-system
```

- On GKE: "at least 2 nodes of type n1-standard-1" are required
- Restricting kube - system might be more of a challenge (DNS, ingress controller)

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/>

Speaker notes

- no option for cluster-wide policies
- whitelisting egress for domain names instead of CIDRs
- filtering on L7 (e.g. HTTP or gRPC)
- netpols will not work in multi-cloud / cluster-federation scenarios

Possible solutions:

- Proprietary extensions of CNI Plugin (e.g. cilium or calico)
- Service Meshes: similar features, also work with multiple clusters;
operate on L7, NetPol on L3/4
→ different strengths, support each other
□ <https://istio.io/blog/2017/0.1-using-network-policy/>

□ Demo

- nosqlclient
- web-console

Speaker notes

- `curl --output /tmp/mongo.tgz https://downloads.mongodb.org/linux/mongodb-shell-linux-x86_64-3.4.18.tgz && tar xf /tmp/mongo.tgz -C /tmp`
- `/tmp/mongodb-linux-x86_64-3.4.18/bin/mongo users --host mongodb.production.svc.cluster.local --eval 'db.users.find().pretty()'`
- - Limited time: Only show ingress whitelisting → Offtopic: MongoDB recommendation → not mongo image but `bitnami/mongo` (helm chart)
- [Demo Script](#)
- [plantUml src](#)
- [plantUml src with prometheus](#)

□ 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

```
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
```


Speaker notes

There is also a securityContext on pod level, but not all of those settings cannot be applied there.

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/>

Speaker notes

- seccomp
 - Switching off in docker would be security misconfiguration. In K8s it's [explicitly deactivated](#) :-o But [will be activated in one of the next versions](#)
 - Has been thoroughly tested by docker on all Dockerfiles on GitHub - see [Jessica Frazzelle](#)

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/>

Speaker notes

- `runAsNonRoot` for nginx image: Error: container has `runAsNonRoot` and image will run as root
 - For custom images: Best Practice run as USER
 - For OTS images this might not be possible
 - For NGINX you could build your own image that does not run as root

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 CapNet tRaw 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: 1000000` in `securityContext` of pod or
 - `USER 1000000` 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>

Speaker notes

- `runAsGroup` was alpha from 1.10, which is deactivated by default

Tools

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

- [controlplaneio/kubesec](#) - manageable 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

Speaker notes

- Results differ between tools.
- The checks are opinionated, just like the recommendations show here.
- Scrutinize, prioritize and be pragmatic when fixing.

□ Demo

□ Wrap-Up: Security Context

My recommendations:

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

Speaker notes

BTW - Security Context settings can be enforced cluster-wide via Pod Security Policies

3. Pod Security Policies

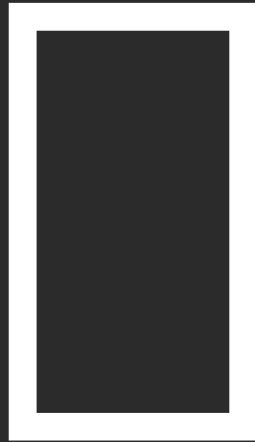
(PSP)

- 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?

What for?

- Increase security

What for?

- Increase security
- Reduce risk of data breach

What for?

- Increase security
- Reduce risk of data breach
- Don't end up on [@haveibeenpwned](#)

Johannes Schnatterer

Cloulogu GmbH

□ <https://cloulogu.com/schulungen>

K8s Security series on JavaSPEKTRUM starting 05/2019

See also □ <https://cloulogu.com/blog>

[@jschnatterer](#)

[@cloulogu](#)

Demo Source: <https://github.com/cloulogu/k8s-security-demos>

Speaker notes

- JavaSPEKTRUM [05/2019: \(27.09.2019\)](#)