

Kubernetes Security and Hardening

Cyber Protection Developer's Conference
Sofia 2019

Orlin Vasilev



kubernetes

\$ whoami - Orlix



Orlin Vasilev

- Software Infrastructure Engineer at VMware
- Meetup Organizer - Kubernetes & CNCF Bulgaria
- father of two
- snowboarder/DIY person/home BBQ pitmaster

twitter: @OrlinVasilev

GitHub: @OrlinVasilev



Agenda



- CNCF and CNCF Bulgaria
- Container Security
- Kubernetes Security
- Demo – kube-bench
- Q&A

Goals of this talk



Kubernetes Community

Raise awareness of high-risk attacks possible in many installs

Demonstrate few attacks

Provide hardening methods

What is needed



Basic Understanding of:
K8s components and concepts
Pods
Services
Docker
networking...

CNCF - Who “Manages” Kubernetes?



**CLOUD NATIVE
COMPUTING FOUNDATION**

The CNCF is a child entity of the Linux Foundation and operates as a vendor neutral governance group.



kubernetes

CNCF - “few” other projects



Graduated



kubernetes
Orchestration



Prometheus
Monitoring



envoy

Service Proxy

Incubating



OPENTRACING

Distributed Tracing
API



fluentd

Logging



Remote
Procedure Call



Container Runtime



rkt

Container Runtime



JAEGER

Distributed Tracing



Security



Vitess

Storage



CoreDNS

Service Discovery



Messaging



LINKERD

Service Mesh



HARBOR

Registry



Package
Management



CNI

Networking API



ROOK

Storage



etcd

Key/Value
Store



Software Upd
Spec



kubernetes

CNCF Bulgaria



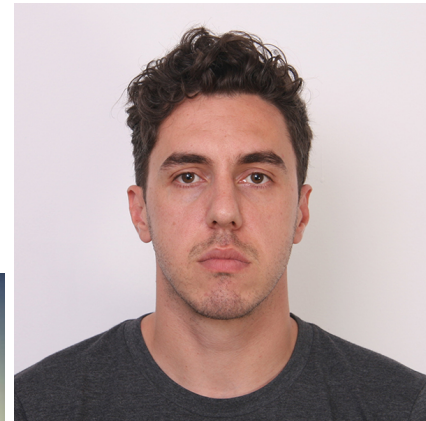
- 7 meetups
- ~550 members
- Meetup: <https://goo.gl/16bF3X>
- YouTube: <https://goo.gl/88yH4n>
- CFP: <https://goo.gl/TfVEMc>



Vladimir Dimov



Spas Atanasov



kubernetes

Possible Attack Surfaces



Underlying Infrastructure - OS , Metadata API

Kube-apiserver

Container - Malicious Image, Compromised Container

Dashboard Access

Escape the container

Direct Etcd Access

Metrics Scraping - read-only ports

Application Tampering

Kubelet

Launch too many pods / high consume CPU/RAM/Disk

Service Account Tokens

Docker daemon security(or containerd, rkt, CRI-O ...)



kubernetes

Container Security



Container Runtime - Least Privileges

Base Image

Image Builder/Maintainer

Image Scanning

Image Signing

TIP: do not run as root in container

```
RUN useradd -r -u 1001 -g appuser appuser  
USER appuser
```



kubernetes

Few Kubernetes Security Aspects



- Kubernetes security:
 - Properly configured RBACs
 - Secrets
 - Pod Security Policy
 - Network Policy
 - Admission Controllers
 - etc...

Kubernetes RBAC



- Enabled in kube-api : `—authorization-mode=RBACRole` and `ClusterRole`
- `RoleBinding` and `ClusterRoleBinding`
- `ServiceAccounts` and `User Accounts`

Tip:

use kube-apiserver audit logs to monitor activities

Role and RoleBinding



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

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: User
  name: jane # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role #this must be Role or ClusterRole
  name: pod-reader #must match name
  apiGroup: rbac.authorization.k8s.io
```



Kubernetes Secrets



- A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key.
- base64ed
- consumed within a pod as volumeMounts
- secrets used to keep docker login information for private registry
- use secrets instead of writing sensitive data in the containers

```
apiVersion: v1
data:
  .dockerconfigjson: eyJhdXRocyI6eyJk...(long base64)
kind: Secret
metadata:
  name: regcred
  namespace: secconf
type: kubernetes.io/dockerconfigjson
```



PodSecurityPolicy



- Enforced by the PodSecurityPolicy admission controller enabled on kube-api
- PSP is an ClusterLevel resource
- Set of conditions which allows a pod to be run
- some examples
 - [privileged](#)
 - [runAsUser](#)
 - [hostPID](#)
 - [volumes](#)

Tips:

- create PSP before enabling PSP admission controller
- PSP are applied alphabetically

```
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: privileged
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
spec:
  privileged: true
  allowPrivilegeEscalation: true
  allowedCapabilities:
  - '*'
  volumes:
  - '*'
  hostNetwork: true
  hostPorts:
  - min: 0
    max: 65535
  hostIPC: true
  hostPID: true
  runAsUser:
    rule: 'RunAsAny'
  selinux:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'
```

Usage via RBAC with ClusterRoles and ClusterRoleBindings to ServiceAccount or User

Network Policy



A network policy is a specification of how groups of pods are allowed to communicate with each other and other network endpoints.

Uses labels to select pods and define rules.

NetworkPolicy needs network plugin which supports it.(calico/weave..)

By default, if no policies exist in a namespace, then all ingress and egress traffic is allowed to and from pods in that namespace.

NetworkPolicy example



```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - ipBlock:
            cidr: 172.17.0.0/16
            except:
              - 172.17.1.0/24
```

```
- namespaceSelector:
    matchLabels:
      project: myproject
- podSelector:
    matchLabels:
      role: frontend
ports:
  - protocol: TCP
    port: 6379
egress:
  - to:
      - ipBlock:
          cidr: 10.0.0.0/24
    ports:
      - protocol: TCP
        port: 5978
```



CIS Kubernetes Benchmark



- 200+ pages of best practices and tests
<https://www.cisecurity.org/benchmark/kubernetes/>
- Tests for workers, masters and federated nodes
- **kube-bench** (Aqua Security)
 - 1265 stars on GitHub
 - ~30 contributors
 - <https://github.com/aquasecurity/kube-bench>



more tools



- **sonobuoy**(heptio)
 - 1141 stars on GitHub
 - 33 contributors
 - <https://github.com/heptio/sonobuoy>
- **kubeaudit** (shopify)
 - 306 stars on GitHub
 - 16 contributors
 - <https://github.com/Shopify/kubeaudit>
- **k8guard**
 - 122 stars on GitHub
 - 8 contributors
 - <https://github.com/k8guard/k8guard-start-from-here>



Demo

Demo 1 - scraping unsecured metrics

Demo 2 - misconfigured taints for master and exploiting the lack of PSP

Demo 3 - NetworkPolicy



aliases:

k = kubectl

kdp = kubectl describe pods

kep = kubectl edit pod

kgn = kubectl get nodes —show-lables -o wide

lp = kubectl get pods

lpa = kubectl get pods —all-namespaces

kshell = kubectl exec -it \$\$ bash

kn = kubens





Q&A

Resources and goodreads



- <https://github.com/kelseyhightower/kubernetes-the-hard-way>
- <https://github.com/hardening-kubernetes/from-scratch>
- <https://github.com/cncf/presentations/tree/master/>
- <https://goo.gl/TNRxtd>

General Guidance

1. Verify that all your security settings properly enforce the policy
2. Use the latest stable K8s version possible to gain the latest security capabilities and fixes
3. Audit the OS, container runtime, and K8s configuration using CIS Benchmarking and other tools like [kube-auto-analyzer](#) and [kube-bench](#)
4. Log **everything** to a location **outside** the cluster

Image Security

1. Use private registries, and restrict public registry usage
2. Scan all images for security vulnerabilities continuously. E.g [CoreOS Clair](#) or [Atomic Scan](#)
3. Decide which types/severity of issues should prevent deployments
4. Maintain standard base images and ensure that all workloads use them
5. **Do NOT run containers as the root user**

K8s Components Security

1. API Server *authorization-mode=Node,RBAC*
2. Ensure all services are protected by TLS
3. Ensure *kubelet* protects its API via *authorization-mode=Webhook*
4. Ensure the *kube-dashboard* uses a restrictive *RBAC* role policy and v1.7+
5. Closely monitor all *RBAC* policy failures
6. Remove default *ServiceAccount* permissions

Network Security

1. Filter access to the cloud provider metadata APIs/ URL, and Limit IAM permissions
2. Use a CNI network plugin that filters ingress/ egress pod network traffic
 - a. Properly label all pods
 - b. Isolate all workloads from each other
 - c. Prevent workloads from egressing to the Internet, the Pod IP space, the Node IP subnets, and/or other internal networks
 - d. Restrict all traffic coming into the kube-system namespace except kube-dns
3. Consider a Service Mesh!

Hardening Tips (Continued)

Workload Containment and Security

1. Namespaces per tenant
2. Default network “deny” inbound on all namespaces
3. Assign CPU/RAM *limits* to all containers
4. Set *automountServiceAccountToken: false* on pods where possible
5. Use a *PodSecurityPolicy* to enforce container restrictions and to protect the node
6. Implement container-aware malicious activity / behavioral detection

Misc Security

1. Collect logs from all containers, especially the RBAC access/deny logs
2. Encrypt the contents of *etcd*, and run *etcd* on dedicated nodes
3. Separate Cloud accounts/VPCs/projects/resource groups
4. Separate clusters for dev/test and production environments
5. Separate node pools for different tenants