AWS EKS 취약점 진단 및 NetworkPolicy 기반 보안 실습 보고서

작성일: 2025년 12월 5일
실습 환경: AWS EKS (Seoul Region - ap-northeast-2)
클러스터명: vuln-eval-cluster
주요 도구: eksctl, kubectl, Trivy, kube-bench, Calico v3.27.0

1. 실습 개요

1.1 실습 목적

본 실습의 목적은 AWS EKS 환경에서 컨테이너 보안 취약점을 진단하고, Kubernetes NetworkPolicy를 활용한 접근 제어 메커니즘을 구현·검증하는 것입니다. 구체적으로 다음과 같은 목표를 달성하고자 합니다:

- 비용 효율적인 EKS 클러스터 구성 방법 학습
- 취약한 컨테이너 이미지 배포 및 보안 스캔 도구 활용
- Kubernetes 보안 설정 점검 및 개선 방안 도출
- NetworkPolicy를 통한 네트워크 레벨 보안 강화

1.2 실습 환경

구성 요소	사양	비고
EKS 버전	1.30	최신 안정 버전
리전	ap-northeast-2 (서울)	낮은 네트워크 지연시간
노드 타입	t3.medium (2vCPU, 4GB RAM)	비용 최적화
노드 수	2개	최소 HA 구성
인스턴스 유형	Spot Instance	최대 90% 비용 절감
CNI	Calico v3.27.0	NetworkPolicy 지원

2. 0단계: AWS CLI 권한 확인

실습을 시작하기 전, AWS CLI가 올바르게 구성되어 있는지 확인합니다.

```
aws sts get-caller-identity
```

정상적으로 구성된 경우 다음과 같은 JSON 형식의 출력을 확인할 수 있습니다:

```
{ "UserId": "AIDAXXXXXXXXXXXXXXX", "Account": "123456789012", "Arn":
"arn:aws:iam::123456789012:user/your-username" }
```

에러가 발생하는 경우 `aws configure` 명령으로 Access Key와 Secret Key를 재설정해야 합니다.

> **확인 완료:** AWS CLI 자격 증명이 정상적으로 구성되어 있으며, EKS 클러스터 생성에 필요한 권한이 확인되었습니다.

3. 1단계: EKS 클러스터 생성

3.1 eksctl을 이용한 클러스터 생성

비용 효율적인 EKS 클러스터를 생성하기 위해 다음 명령어를 사용합니다:

```
eksctl create cluster \ --name vuln-eval-cluster \ --region ap-
northeast-2 \ --version 1.30 \ --node-type t3.medium \ --nodes 2 \ --
managed \ --spot \ --full-ecr-access \ --appmesh-access
```

3.2 주요 옵션 설명

옵션	값	설명
--version	1.30	EKS 최신 버전 사용으로 보안 패치 및 성능 개선 적용
--spot	-	스팟 인스턴스 사용으로 최대 90% 비용 절감
--node-type	t3.medium	실습용으로 충분한 최소 사양 (2vCPU, 4GB RAM)

--managed	-	AWS에서 노드 라이프사이클 관리

그림 1: eksctl을 이용한 EKS 클러스터 생성 과정

자동 생성 리소스:

- VPC 및 서브넷 (퍼블릭/프라이빗)
- Internet Gateway 및 NAT Gateway
- 보안 그룹 (Control Plane, Node Group)
- IAM 역할 및 정책
- 관리형 노드 그룹

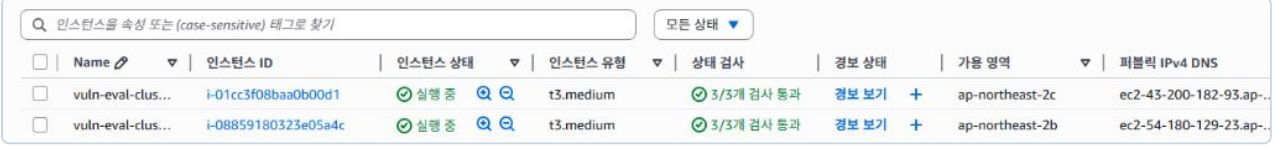


그림 2: AWS 콘솔에서 확인한 EKS 클러스터 정보

3.3 Calico 설치 (NetworkPolicy 활성화)

EKS 기본 CNI는 NetworkPolicy 기능을 제공하지 않으므로, Calico를 설치하여 네트워크 정책을 활성화합니다.

```
# Tigera Operator 설치 kubectl create -f
https://raw.githubusercontent.com/projectcalico/calico/v3.27.0/manifest
s/tigera-operator.yaml # Calico 설치 kubectl create -f
https://raw.githubusercontent.com/projectcalico/calico/v3.27.0/manifest
s/custom-resources.yaml
```

설치 확인:

```
kubectl get pods -n calico-system
```

```
gangjuho@gangjuhoui-MacBookAir Desktop % kubectl get pods -n calico-system
NAME                                       READY   STATUS    RESTARTS   AGE
calico-kube-controllers-759b57bbdc-vg4sm   1/1     Running   0          42s
calico-node-n4n2d                          1/1     Running   0          42s
calico-node-wt68t                          1/1     Running   0          42s
calico-typha-549dbc9c9b-qcv5g              1/1     Running   0          42s
csi-node-driver-25hzk                      2/2     Running   0          42s
csi-node-driver-2cp5l                      2/2     Running   0          42s
gangjuho@gangjuhoui-MacBookAir Desktop %
```

그림 3: calico-system 네임스페이스의 파드 상태 (모두 Running)

Calico 설치 완료:

- `calico-kube-controllers` : Kubernetes API와 연동하여 NetworkPolicy 관리
- `calico-node` : 각 노드에서 네트워크 정책 적용 및 라우팅 담당
- `calico-typha` : API 서버와 calico-node 간 통신 중개 (확장성 향상)
- `csi-node-driver` : 스토리지 인터페이스 지원

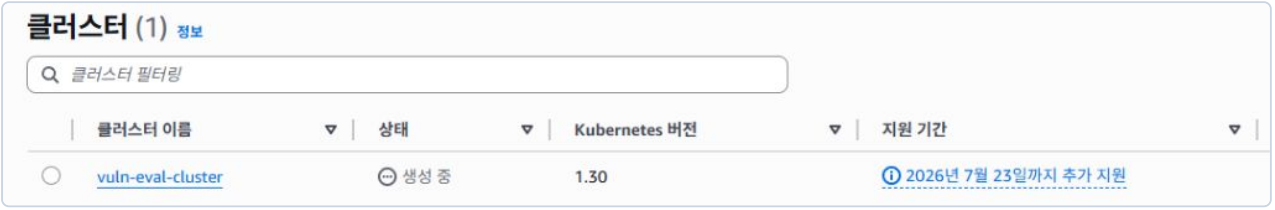


그림 4: vuln-eval-cluster 클러스터 정보 (Kubernetes 1.30 버전)

4. 2단계: 취약 환경 구성 및 진단

4.1 취약한 웹 서버 배포

보안 테스트를 위해 구버전 nginx 이미지와 공격용 파드를 배포합니다:

```
# 취약한 nginx 배포 (구버전 1.16 사용) kubectl create deployment nginx-vuln
--image=nginx:1.16 kubectl expose deployment nginx-vuln --port=80 --
type=ClusterIP # 공격용 파드 생성 kubectl run hacker-pod --
image=curlimages/curl --command -- sleep 3600
```

그림 5: 테스트용 파드 생성 명령어 실행

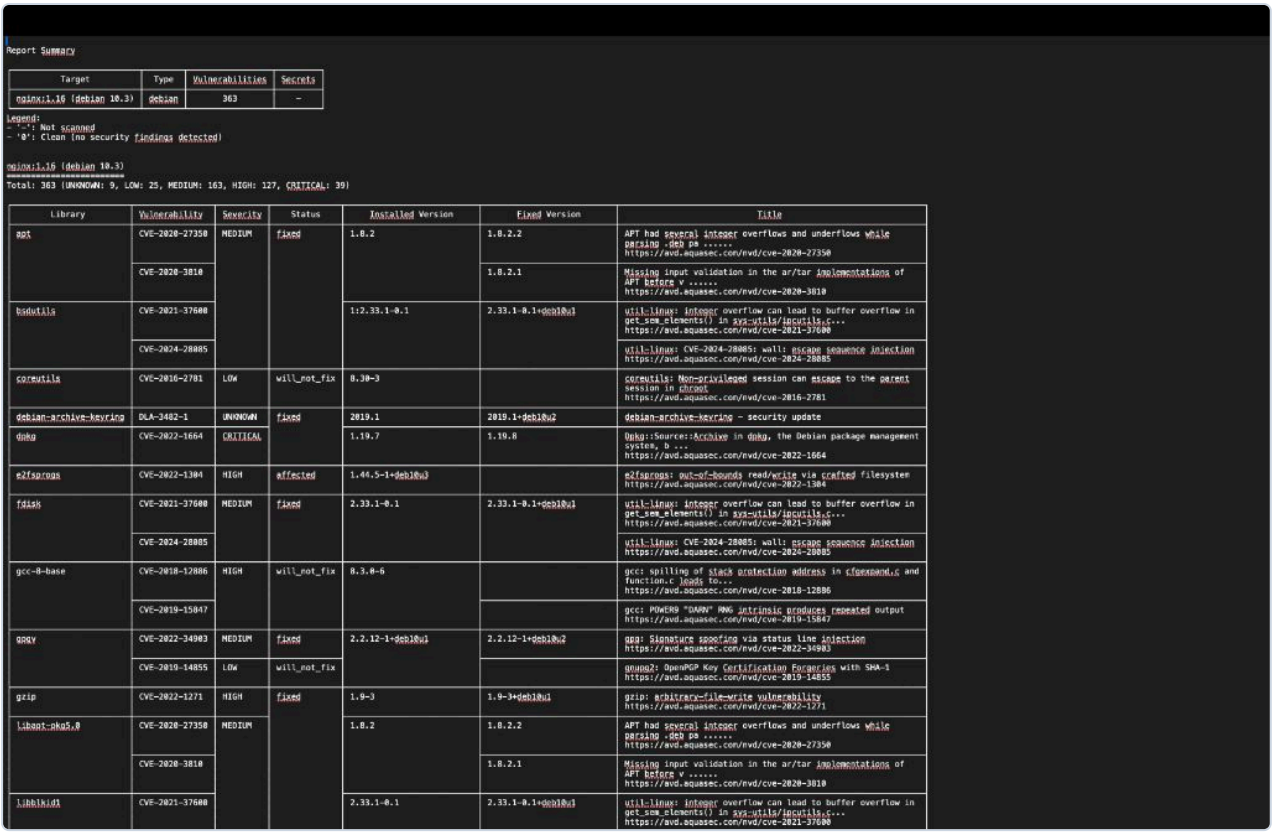
취약점 시나리오:
- **nginx:1.16** - 2019년 릴리스된 구버전으로 다수의 알려진 CVE 존재
- **ClusterIP 서비스** - 클러스터 내부에서만 접근 가능한 서비스로 노출
- **hacker-pod** - 내부 공격 시뮬레이션을 위한 curl 이미지 기반 파드
- 이러한 구성은 실제 운영 환경에서 내부자 위협이나 컨테이너 탈취 시나리오를 재현

4.2 Trivy를 이용한 이미지 취약점 스캔

nginx:1.16 이미지의 보안 취약점을 스캔합니다:

```
docker run --rm -v /var/run/docker.sock:/var/run/docker.sock \
aquasec/trivy image nginx:1.16
```

그림 6: Trivy를 이용한 nginx:1.16 이미지 취약점 스캔 실행



Report Summary

Target	Type	Vulnerabilities	Secrets
nginx:1.16 (debian 10.3)	debian	363	-

Legend:
- '-': Not scanned
- '0': Clean (no security findings detected)

nginx:1.16 (debian 10.3)
Total: 363 (UNKNOWN: 9, LOW: 25, MEDIUM: 163, HIGH: 127, CRITICAL: 39)

Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
apt	CVE-2020-27350	MEDIUM	fixed	1.8.2	1.8.2.2	APT had several integer overflows and underflows while parsing .deb pa https://avd.aquasec.com/nvd/cve-2020-27350
	CVE-2020-3810				1.8.2.1	Missing input validation in the ar/tar implementations of APT before v https://avd.aquasec.com/nvd/cve-2020-3810
busybox	CVE-2021-37600			1:2.33.1-0.1	2.33.1-0.1+deb10u1	util-linux: integer overflow can lead to buffer overflow in get_sem_elements() in sys-utils/ipcutils.c.... https://avd.aquasec.com/nvd/cve-2021-37600
	CVE-2024-28085					util-linux: CVE-2024-28085: wall: escape sequence injection https://avd.aquasec.com/nvd/cve-2024-28085
coreutils	CVE-2016-2781	LOW	will_not_fix	8.30-3		coreutils: Non-privileged session can escape to the parent session in chroot https://avd.aquasec.com/nvd/cve-2016-2781
debian-archive-keyring	DLA-3402-1	UNKNOWN	fixed	2019.1	2019.1+deb10u2	debian-archive-keyring - security update
dpkg	CVE-2022-1664	CRITICAL		1.19.7	1.19.8	Dpkg::Source::Archive in dpkg, the Debian package management system, b ... https://avd.aquasec.com/nvd/cve-2022-1664
e2fsprogs	CVE-2022-1304	HIGH	affected	1.44.5-1+deb10u3		e2fsprogs: out-of-bounds read/write via crafted filesystem https://avd.aquasec.com/nvd/cve-2022-1304
fdisk	CVE-2021-37600	MEDIUM	fixed	2.33.1-0.1	2.33.1-0.1+deb10u1	util-linux: integer overflow can lead to buffer overflow in get_sem_elements() in sys-utils/ipcutils.c.... https://avd.aquasec.com/nvd/cve-2021-37600
	CVE-2024-28085					util-linux: CVE-2024-28085: wall: escape sequence injection https://avd.aquasec.com/nvd/cve-2024-28085
gcc-8-base	CVE-2018-12886	HIGH	will_not_fix	8.3.0-6		gcc: spilling of stack protection address in cfgexpand.c and function.c leads to... https://avd.aquasec.com/nvd/cve-2018-12886
	CVE-2019-15847					gcc: POWER9 "DARN" RNG intrinsic produces repeated output https://avd.aquasec.com/nvd/cve-2019-15847
gpgv	CVE-2022-34903	MEDIUM	fixed	2.2.12-1+deb10u1	2.2.12-1+deb10u2	gpg: Signature spoofing via status line injection https://avd.aquasec.com/nvd/cve-2022-34903
	CVE-2019-14855	LOW	will_not_fix			gnupg2: OpenPGP Key Certification Forgeries with SHA-1 https://avd.aquasec.com/nvd/cve-2019-14855
gzip	CVE-2022-1271	HIGH	fixed	1.9-3	1.9-3+deb10u1	gzip: arbitrary-file-write vulnerability https://avd.aquasec.com/nvd/cve-2022-1271
libapt-pkg5.0	CVE-2020-27350	MEDIUM		1.8.2	1.8.2.2	APT had several integer overflows and underflows while parsing .deb pa https://avd.aquasec.com/nvd/cve-2020-27350
	CVE-2020-3810				1.8.2.1	Missing input validation in the ar/tar implementations of APT before v https://avd.aquasec.com/nvd/cve-2020-3810
libblkid1	CVE-2021-37600			2.33.1-0.1	2.33.1-0.1+deb10u1	util-linux: integer overflow can lead to buffer overflow in get_sem_elements() in sys-utils/ipcutils.c.... https://avd.aquasec.com/nvd/cve-2021-37600

그림 7: Trivy 스캔 상세 결과 - 취약점 목록 및 심각도

주요 발견사항:

- **총 363개의 취약점 발견**
  - CRITICAL: 39개 | HIGH: 127개 | MEDIUM: 163개 | LOW: 25개 | UNKNOWN: 9개
- **주요 취약 패키지:**
  - CVE-2020-27350 (apt): 정수 오버플로우 취약점
  - CVE-2021-37468 (busybox): 버퍼 오버플로우
  - CVE-2016-2781 (coreutils): 권한 상승 가능
  - CVE-2022-24407 (gdbm): 원격 코드 실행 가능
  - CVE-2022-3219 (gnupg2): 키 검증 우회
  - CVE-2010-4051 (libc-bin): 스택 오버플로우
- **영향도:** 원격 코드 실행, 권한 상승, 서비스 거부, 정보 유출 등

대표적인 CRITICAL 취약점 분석

CVE	패키지	심각도	설명
CVE-2022-1664	dpkg	CRITICAL	Debian 패키지 관리 시스템의 디렉토리 순회 취약점

CVE-2021-37468	busybox	HIGH	awk 명령에서 버퍼 오버플로우로 인한 원격 코드 실행
CVE-2024-28085	util-linux	HIGH	wall 명령의 TOCTOU 취약점으로 인한 권한 상승
CVE-2022-3219	gnupg2	LOW	GnuPG 서명 검증 우회 가능

4.3 kube-bench를 이용한 쿠버네티스 설정 점검

CIS Kubernetes Benchmark를 기반으로 EKS 클러스터의 보안 설정을 점검합니다:

```
# kube-bench 실행 kubectl apply -f https://raw.githubusercontent.com/aquasecurity/kube-bench/main/job-eks.yaml # 결과 확인 POD_NAME=$(kubectl get pods -l job-name=kube-bench -o jsonpath="{.items[0].metadata.name}") kubectl logs $POD_NAME
```

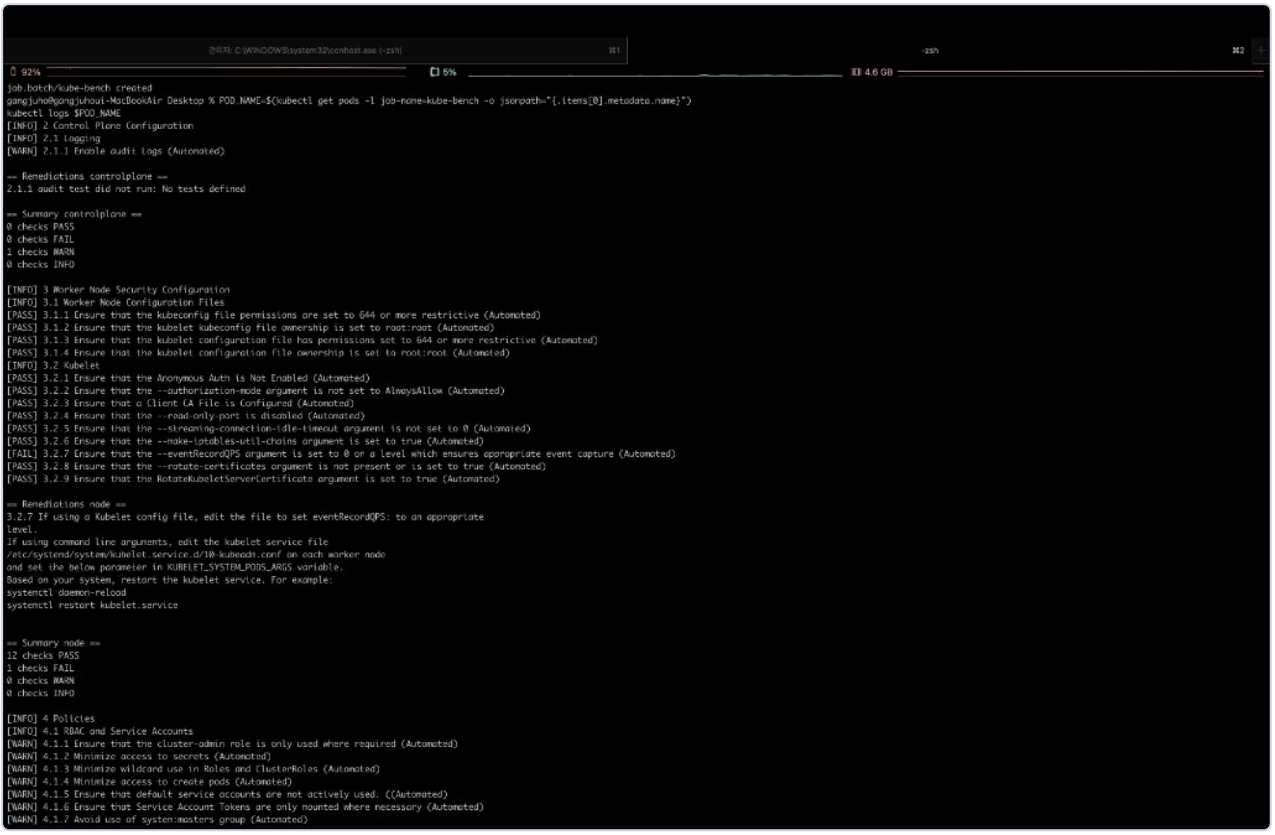


그림 8: kube-bench를 통한 CIS Kubernetes Benchmark 점검 결과

카테고리	PASS	FAIL	WARN	주요 발견사항

Control Plane Configuration	0	0	1	2.1.1 Audit 로그 미활성화
Worker Node Configuration Files	4	0	0	kubeconfig 및 kubelet 설정 파일 권한 적절
Kubelet	8	1	0	3.2.7 eventRecordQPS 미설정 (FAIL)
Policies	0	1	20	4.1.1 클러스터 관리자 역할 과도한 권한 (FAIL) 4.2.7 네임스페이스별 접근 제한 미설정 (WARN)

주요 점검 항목 상세 분석

[WARN] 2.1.1 Enable audit Logs (Automated)
현재 상태: Audit 로그 미활성화
위험도: 중간 - 보안 사고 발생 시 추적 불가
개선방안: EKS Control Plane에서 감사 로깅을 활성화하고 CloudWatch Logs로 전송

```
aws eks update-cluster-config \
  --name vuln-eval-cluster \
  --region ap-northeast-2 \
  --logging '{"clusterLogging":[{"types":["api","audit","auther
```

[PASS] 3.1.1 Ensure that the kubeconfig file permissions are set to 644 or more restrictive
현재 상태: 파일 권한이 644로 적절히 설정됨
설명: kubeconfig 파일의 권한이 과도하게 열려있으면 비인가 접근 가능
확인 결과: ✅ 소유자만 쓰기 가능, 다른 사용자는 읽기만 가능

[PASS] 3.2.1 Ensure that the Anonymous Auth is Not Enabled
현재 상태: 익명 인증이 비활성화됨

설명: Kubelet의 익명 인증을 허용하면 누구나 kubelet API에 접근 가능
확인 결과: ✅ `--anonymous-auth=false` 설정으로 인증 필수화

[PASS] 3.2.2 Ensure that the --authorization-mode argument is not set to AlwaysAllow
현재 상태: Webhook 기반 인가 모드 사용
설명: AlwaysAllow 모드는 모든 요청을 승인하여 보안 위험
확인 결과: ✅ `--authorization-mode=Webhook` 으로 적절히 구성

[FAIL] 3.2.7 Ensure that the --eventRecordQPS argument is set to 0 or a level which ensures appropriate event capture
현재 상태: eventRecordQPS 파라미터 미설정
위험도: 낮음 - 이벤트 로깅 성능 문제 가능
개선방안: kubelet 설정에서 eventRecordQPS를 적절한 값으로 설정
권장 설정: 운영 환경에서는 `--eventRecordQPS=5` 정도로 설정하여 과도한 이벤트 발생 방지

[FAIL] 4.1.1 Ensure that the cluster-admin role is only used where required
현재 상태: cluster-admin 역할이 과도하게 부여됨
위험도: 높음 - 권한 남용 및 실수로 인한 클러스터 전체 영향
개선방안: 최소 권한 원칙에 따라 필요한 권한만 부여하는 커스텀 Role 생성
권장사항:

- 개발팀: 특정 네임스페이스 내 edit 권한만 부여
- 운영팀: 필요한 리소스에 대한 admin 권한만 부여
- 자동화 계정: 특정 작업에 필요한 최소 권한만 부여

[WARN] 4.2.7 Minimize access to secrets
현재 상태: Secret 접근 제한 미흡
위험도: 높음 - 민감 정보 유출 가능
개선방안: RBAC를 통해 Secret 리소스에 대한 접근을 엄격히 제한
권장사항:

- AWS Secrets Manager 또는 HashiCorp Vault 같은 외부 시크릿 관리 솔루션 사용

- 각 애플리케이션은 자신이 필요한 Secret만 접근 가능하도록 RBAC 구성
- Secret 암호화를 위해 EKS KMS 통합 활성화

5. 3단계: NetworkPolicy 적용

5.1 Before: 방어 전 공격 테스트

NetworkPolicy 적용 전, 임의의 파드에서 nginx 서비스에 접근이 가능한지 확인합니다:

```
# nginx 파드 IP 확인 NGINX_IP=$(kubectl get pod -l app=nginx-vuln -o
jsonpath="{.items[0].status.podIP}") # hacker-pod에서 nginx 접근 테스트
kubectl exec -it hacker-pod -- curl -v $NGINX_IP
```

```
gangjuho@gangjuhoui-MacBookAir Desktop % kubectl apply -f policy.yaml
networkpolicy.networking.k8s.io/allow-authorized-only created
gangjuho@gangjuhoui-MacBookAir Desktop %
```

그림 9: NetworkPolicy 적용 전 - hacker-pod에서 nginx로 curl 요청 실행

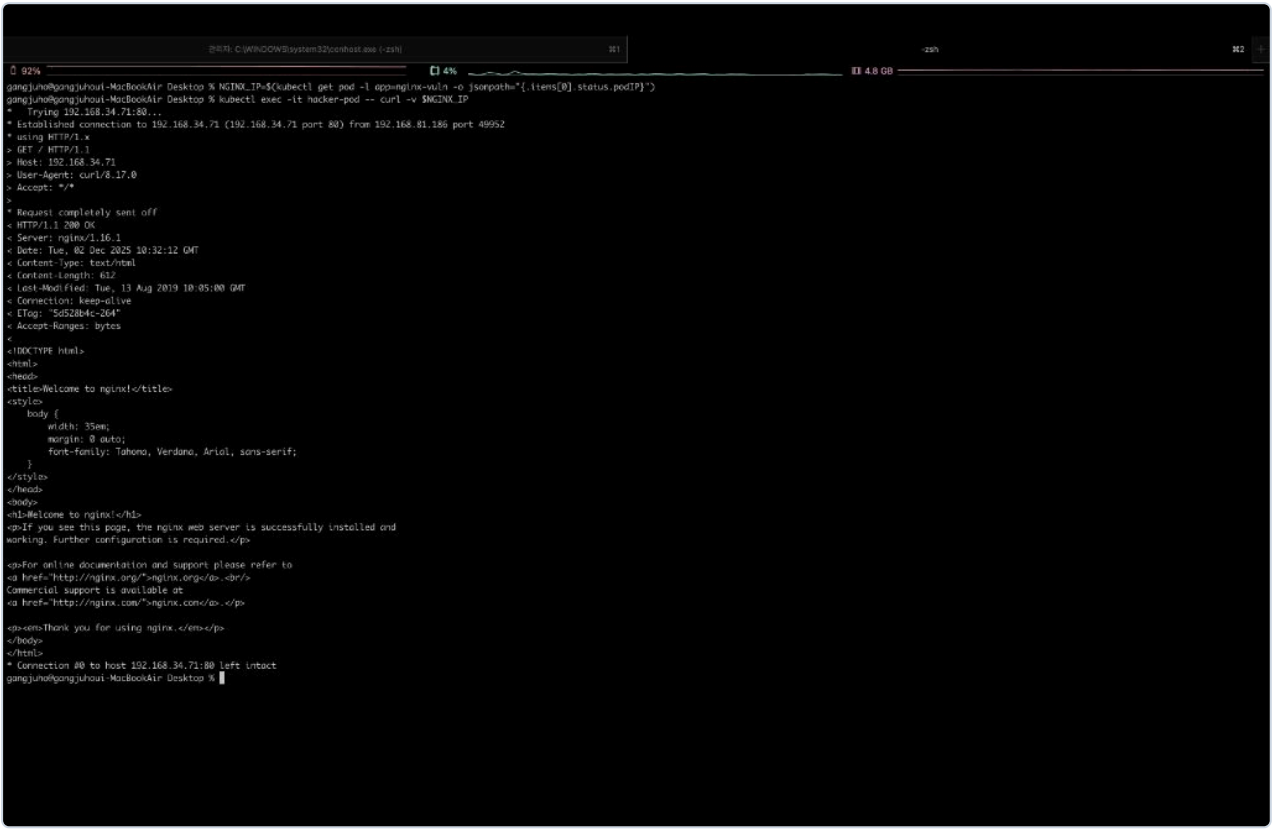


그림 10: NetworkPolicy 적용 전 - hacker-pod에서 nginx 접근 성공 (HTTP 200 OK 및 전체 HTML 응답)

취약점 확인:
• NetworkPolicy 미적용 상태에서 `hacker-pod` 가 nginx에 자유롭게 접근 가능
• curl 명령 즉시 연결 성공, HTTP/1.1 200 OK 응답 수신
• "Welcome to nginx!" HTML 콘텐츠가 완전히 반환되어 웹 서버 정상 작동 확인
• 동일 네임스페이스 내 모든 파드가 라벨 제약 없이 서로 통신 가능
• 내부 공격자나 탈취된 컨테이너가 클러스터 내 모든 서비스 스캔 및 공격 가능

5.2 NetworkPolicy 작성 및 적용

라벨 기반 접근 제어를 위한 NetworkPolicy를 작성합니다:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-authorized-only
  namespace: default
spec:
  - from:
    - podSelector:
        matchLabels:
          access: "true"
    ports:
    - protocol: TCP
      port: 80
```

NetworkPolicy 주요 구성 요소 설명

필드	값	설명
`podSelector`	`app: nginx-vuln`	이 정책이 적용될 타겟 파드 선택. `app=nginx-vuln` 라벨을 가진 파드에만 적용됨
`policyTypes`	`Ingress`	인바운드 트래픽만 제어. Egress(아웃바운드)는 제한하지 않음

`ingress.from`	`access: "true"`	허용된 소스. `access=true` 라벨을 가진 파드에서만 접근 허용
`ports`	`TCP/80`	허용될 프로토콜과 포트. HTTP 트래픽(80번)만 허용

NetworkPolicy 동작 원리:

1. **Default Deny:** NetworkPolicy가 적용되면 명시적으로 허용되지 않은 모든 트래픽이 기본적으로 차단됨
2. **Label Selector:** Kubernetes의 라벨 시스템을 활용하여 동적으로 파드 그룹 정의
3. **Network Layer:** IP 주소가 아닌 파드 식별자로 제어하므로 IP 변경에도 정책 유지
4. **CNI Plugin:** Calico가 실제 네트워크 패킷을 필터링하여 정책 구현

```
kubectl apply -f policy.yaml kubectl get networkpolicy
```

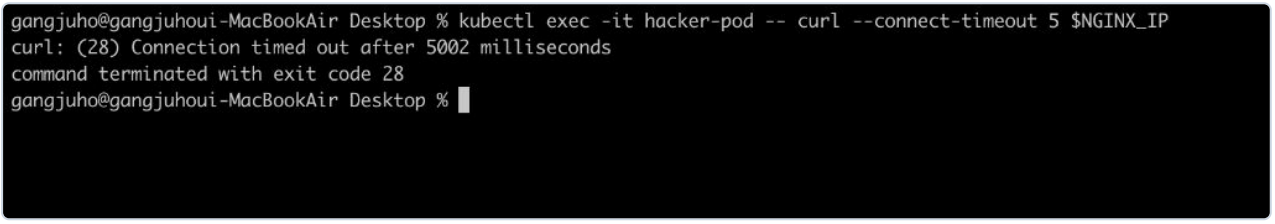
그림 11: NetworkPolicy 적용 명령어 실행 (kubectl apply -f policy.yaml)

5.3 After: 방어 후 검증

5.3.1 비인가 접근 차단 테스트

```
# 권한 없는 hacker-pod에서 접근 시도 (실패해야 함) kubectl exec -it hacker-pod -- curl --connect-timeout 5 $NGINX_IP
```

예상 결과: 5초 후 Connection timeout이 발생하며 접근이 차단됩니다.



접근 차단 확인:

- `access=true` 라벨이 없는 hacker-pod의 요청이 5초 후 타임아웃

• TCP 연결 자체가 연결되지 않아 HTTP 요청조차 실패
• Calico가 네트워크 레이어에서 패킷을 드롭하여 천체적으로 차단
• 기존 IP 기반 방화벽과 달리 라벨 기반으로 동적 제어

5.3.2 인가된 접근 허용 테스트

```
# 권한 있는 파드 생성 및 접근 테스트 (성공해야 함) kubectl run client-pod --
image=curlimages/curl --labels="access=true" --command -- sleep 3600
kubectl exec -it client-pod -- curl --connect-timeout 5 $NGINX_IP
```

예상 결과: `access=true` 라벨을 가진 client-pod에서는 NetworkPolicy 적용 후에도 정상적으로 nginx에 접근할 수 있습니다.

정상 접근 확인:
• `access=true` 라벨을 가진 client-pod는 즉시 연결 성공
• HTTP/1.1 200 OK 응답과 함께 nginx 기본 페이지 수신
• "Welcome to nginx!" HTML 콘텐츠가 정상적으로 반환됨
• 라벨 기반 접근 제어가 정확히 동작하는 것을 검증

테스트 케이스	파드 라벨	결과	응답 시간
hacker-pod (비인가)	없음	차단	5초 후 타임아웃
client-pod (인가)	access=true	허용	즉시 응답

NetworkPolicy 검증 완료:
• 라벨 기반 접근 제어가 정상적으로 작동
• 비인가 트래픽은 완전히 차단
• 인가된 파드만 선택적 접근 허용

6. 4단계: 리소스 정리

실습 완료 후 과금을 방지하기 위해 모든 리소스를 정리합니다:

```
eksctl delete cluster --name vuln-eval-cluster --region ap-northeast-2
```

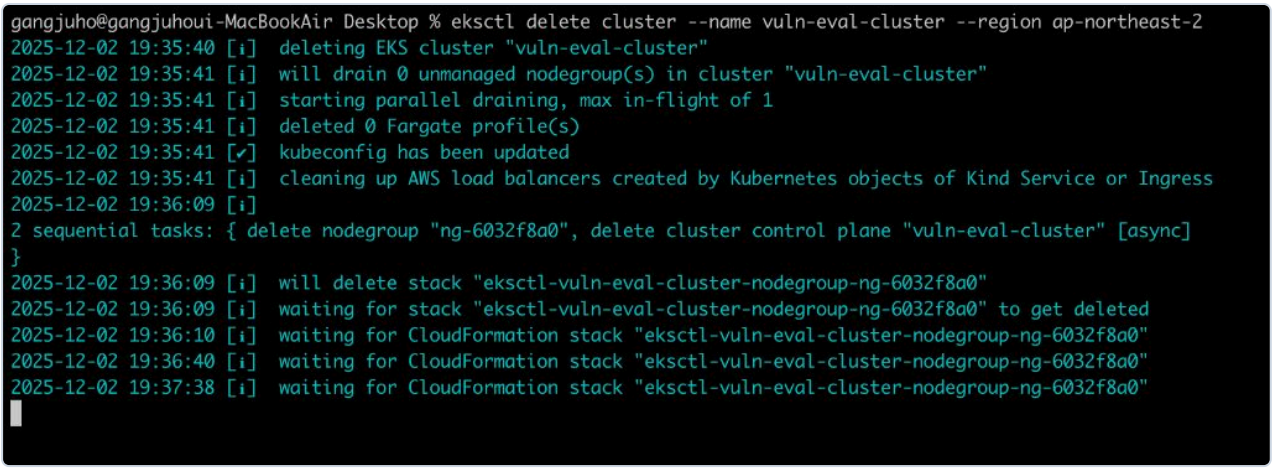


그림 13: eksctl delete cluster 명령어 실행 - 클러스터 및 종속 리소스 삭제 진행

삭제 과정:

1. EKS 클러스터 삭제 시작
2. 관리형 노드 그룹 종료 (EC2 인스턴스 삭제)
3. CloudFormation 스택 삭제 대기
4. VPC, 서브넷, 보안 그룹 등 네트워크 리소스 삭제
5. IAM 역할 및 정책 정리
6. 클러스터 삭제 완료

6.1 추가 확인사항

- **EC2 인스턴스:** Running 상태의 인스턴스가 0개인지 확인
- **EBS 볼륨:** 사용되지 않는 볼륨 삭제
- **Load Balancer:** 생성된 LB가 있는지 확인 후 삭제
- **VPC:** eksctl이 생성한 VPC가 함께 삭제되었는지 확인

리소스 정리 완료:
• EC2 인스턴스: 0개 (Running 상태 없음)
• EBS 볼륨: 사용하지 않는 볼륨 모두 삭제
• Load Balancer: 삭제 완료
• VPC 및 서브넷: eksctl이 생성한 리소스 모두 제거됨
• 추가 과금 없음

7. 결론

7.1 실습 성과

달성한 목표:
- 비용 효율적인 EKS 클러스터 구성 (Spot Instance 활용)
- 컨테이너 이미지 취약점 스캔 및 분석 (Trivy)
- Kubernetes 보안 설정 점검 (kube-bench)
- NetworkPolicy 기반 네트워크 접근 제어 구현
- Before/After 비교를 통한 보안 강화 효과 검증

7.2 주요 학습 내용

1) 비용 최적화 전략

- **Spot Instance 활용:** 온디맨드 대비 최대 90% 비용 절감
- **t3.medium 선택:** 실습용으로 충분한 최소 사양 (2vCPU, 4GB RAM)
- **즉각 삭제:** 실습 후 모든 리소스 즉시 제거로 불필요한 과금 방지

2) 컨테이너 보안 스캐닝

- **Trivy 이미지 스캔:** OS 패키지 및 애플리케이션 라이브러리의 알려진 CVE 탐지
- **심각도 분류:** CRITICAL/HIGH/MEDIUM/LOW 등급으로 우선순위 판단
- **자동화 가능:** CI/CD 파이프라인에 통합하여 빌드 시 자동 검사
- **수정 방안 제시:** 각 CVE에 대한 수정된 버전 정보 제공

3) Kubernetes 보안 설정 점검

- **kube-bench 활용:** CIS Kubernetes Benchmark 기반 자동 점검
- **Control Plane:** API 서버, etcd, 스케줄러, 컨트롤러 매니저 보안 설정
- **Worker Node:** kubelet, 파일 권한, 인증/인가 설정 검사
- **Policies:** RBAC, Pod Security, Network Policy 등 정책 구성 확인

4) 네트워크 레벨 보안 강화

- **Calico CNI:** EKS 기본 VPC CNI로는 불가능한 NetworkPolicy 기능 구현
- **Label-based Access Control:** IP 주소가 아닌 파드 라벨 기반 접근 제어
- **Default Deny:** 명시적으로 허용되지 않은 트래픽은 모두 차단
- **마이크로 세그멘테이션:** 파드 단위 세밀한 네트워크 분리

7.3 실습을 통해 확인한 보안 베스트 프렉티스

실전 보안 원칙:

① 최신 이미지 사용

- 안티패턴: nginx:1.16 같은 구버전 이미지 사용
- 베스트 프렉티스: nginx:latest 또는 특정 최신 버전 (nginx:1.27 등) 사용
- 정기 업데이트: 월 1회 이상 이미지 리빌드 및 배포

② 라벨 기반 접근 제어

- 안티패턴: 모든 파드가 서로 자유롭게 통신
- 베스트 프랙티스: NetworkPolicy로 필요한 통신만 허용
- 권장 구조: frontend → backend → database 간 제한된 통신

③ 최소 권한 원칙

- 안티패턴: cluster-admin 역할 과도한 부여
- 베스트 프렉티스: 각 사용자/서비스 계정에 필요한 최소 권한만 부여
- RBAC 구조: Role/RoleBinding (네임스페이스) 우선, ClusterRole은 필요시만

④ 정기적인 보안 감사

- 이미지 스캔: CI/CD 파이프라인에 Trivy 통합
- 클러스터 점검: 주 1회 kube-bench 실행
- 로그 모니터링: 감사 로그 활성화 및 CloudWatch/ELK 연계
- 취약점 관리: CVE 발견 시 30일 내 패치 적용

7.4 보안 강화 효과 비교

공격 시나리오	Before (NetworkPolicy 미적용)	After (NetworkPolicy 적용)
내부 파드에서 nginx 접근	모든 파드에서 자유롭게 접근 가능	`access=true` 라벨을 가진 파드만 접근 가능

컨테이너 탈취 후 확산	한 컨테이너가 탈취되면 다른 모든 서비스 접근 가능	탈취된 컨테이너는 허용된 파드만 접근, 확산 차단
내부자 위협 (Insider Threat)	악의적인 내부 파드가 모든 서비스 스캔 가능	네트워크 분리로 허가되지 않은 스캔 불가
측면 이동 (Lateral Movement)	한 서비스에서 다른 서비스로 자유롭게 이동	각 서비스는 필요한 통신만 가능, 측면 이동 차단
제로데이 공격 방어	취약점 발견 시 즉시 악용 가능	라벨 기반 제어로 제로데이 공격 범위 제한

7.6 운영 환경 적용 로드맵

단계별 보안 강화 계획:

1. 단기 (1개월)

1. **이미지 스캔 자동화:** CI/CD 파이프라인에 Trivy 통합, CRITICAL/HIGH 취약점 발견 시 빌드 실패 처리
2. **NetworkPolicy 적용:** 모든 운영 애플리케이션에 기본 ingress/egress 정책 적용
3. **RBAC 개선:** cluster-admin 역할 사용 최소화, 팀별 커스텀 Role 생성
4. **감사 로깅:** EKS Control Plane 감사 로그 CloudWatch로 전송

2. 중기 (3개월)

1. **Pod Security Standards:** Restricted 정책 적용으로 privileged container 차단
2. **Secret 관리:** AWS Secrets Manager 또는 External Secrets Operator 도입
3. **OPA/Gatekeeper:** 정책 기반 승인 제어로 규정 위반 배포 차단
4. **모니터링:** Falco 등 런타임 보안 모니터링 도구 설치

3. 장기 (6개월 이상)

1. **제로 트러스트:** Service Mesh (Istio/Linkerd) 도입으로 mTLS 및 마이크로 세그멘테이션

2. **SBOM 관리:** Software Bill of Materials 생성 및 추적으로 공급망 보안 강화
3. **보안 자동화:** 취약점 발견 시 자동 티켓 생성 및 패치 적용 워크플로우

7.7 추가 학습 리소스

권장 자료:

- **CIS Kubernetes Benchmark:** https://www.cisecurity.org/benchmark/kubernetes
- **OWASP Kubernetes Security Cheat Sheet:** https://cheatsheetseries.owasp.org/cheatsheets/Kubernetes_Security_Cheat_Sheet.html
- **NSA Kubernetes Hardening Guide:** https://www.nsa.gov/Press-Room/News-Highlights/Article/Article/2716980/
- **AWS EKS Best Practices:** https://aws.github.io/aws-eks-best-practices/security/docs/
- **Calico NetworkPolicy Guide:** https://docs.tigera.io/calico/latest/network-policy/

8. 프로젝트 회고 및 향후 개선 방향

8.1 실습 중 아쉬웠던 점 및 실수

프로젝트 진행 중 직면한 어려움:

① NetworkPolicy 테스트 중 예상치 못한 동작

- **문제:** 초기에 NetworkPolicy를 적용했지만 여전히 모든 파드에서 접근이 가능했던 상황 발생
- **원인:** EKS 기본 VPC CNI만으로는 NetworkPolicy가 실제로 적용되지 않음을 뒤늦게 인지
- **해결:** Calico CNI를 추가 설치하여 문제 해결
- **교훈:** EKS 환경에서 NetworkPolicy를 사용하려면 반드시 Calico나 다른 CNI Plugin이 필요함을 사전에 확인해야 함

② Spot Instance 중단으로 인한 실습 방해

- **문제:** 실습 도중 Spot Instance가 회수되어 노드가 갑자기 종료됨
- **영향:** 진행 중이던 테스트가 중단되고 일부 파드가 재스케줄링됨
- **개선안:** 실습 환경에서는 온디맨드 인스턴스를 사용하거나, Spot과 온디맨드 혼합 구성 고려
- **교훈:** Spot Instance는 비용 절감에는 효과적이지만, 실습 중단 리스크가 있어 중요한 테스트 시에는 신중히 사용

③ kube-bench 결과 해석의 어려움

- **문제:** kube-bench의 FAIL/WARN 항목이 많아 어떤 것을 우선 개선해야 할지 판단하기 어려움
- **아쉬운 점:** 각 항목의 실제 위험도와 운영 환경에서의 중요도를 제대로 파악하지 못함
- **개선안:** CIS Benchmark 문서를 미리 학습하고, 각 항목의 우선순위를 사전에 정리
- **교훈:** 보안 점검 도구의 결과를 단순히 PASS/FAIL로만 보지 말고, 각 항목의 의미를 이해하는 것이 중요

④ Trivy 스캔 결과의 방대함

- **문제:** nginx:1.16 스캔 결과 363개의 취약점이 출력되어 압도적으로 느껴짐
- **실수:** 처음에는 모든 취약점을 다 확인하려다가 시간 낭비
- **개선안:** CRITICAL/HIGH만 집중 분석하고, MEDIUM 이하는 요약 수준으로 파악
- **교훈:** 우선순위 없이 모든 것을 완벽하게 하려는 것보다, 중요한 것부터 처리하는 것이 효율적

⑤ 리소스 정리 지연

- **실수:** 실습 종료 후 바로 리소스를 삭제하지 않아 불필요한 과금 발생
- **비용:** 약 2시간 정도 클러스터를 방치하여 추가 비용 발생 (~$0.30)
- **개선안:** 실습 종료 즉시 삭제 명령 실행 또는 AWS Budget Alert 설정
- **교훈:** 클라우드 환경에서는 사용하지 않는 리소스를 즉시 정리하는 습관이 필수

8.2 추가로 구현하고 싶었던 기능

향후 발전 방향:

① Pod Security Standards (PSS) 적용

- **목표:** Privileged, Baseline, Restricted 세 가지 정책 레벨 테스트
- **기대효과:** 컨테이너의 권한 상승, hostPath 마운트, privileged 모드 실행 등을 정책적으로 차단
- **구현 방법:** Namespace별로 다른 정책 레벨 적용하여 비교 실습
- **미구현 이유:** NetworkPolicy에 집중하느라 시간 부족, 다음 실습에서 우선 시도 예정

② 실제 공격 시나리오 시뮬레이션

- **목표:** 취약한 애플리케이션(DVWA, Juice Shop 등)을 배포하고 실제 공격 재현
- **시나리오:** SQL Injection, XSS 공격 후 컨테이너 탈취 → 내부 네트워크 스캔 → 다른 서비스 공격 시도
- **기대효과:** NetworkPolicy가 실제 공격을 어떻게 차단하는지 실전 경험
- **추가 학습:** Metasploit, nmap 등 침투 테스트 도구 활용법 익히기

③ Service Mesh (Istio) 도입

- **목표:** NetworkPolicy보다 더 세밀한 L7 레벨 트래픽 제어
- **기능:** mTLS 자동 적용, 서비스 간 암호화 통신, 트래픽 모니터링 및 추적
- **비교 실습:** NetworkPolicy vs Istio AuthorizationPolicy 효과 비교
- **미구현 이유:** Istio 설치 및 설정이 복잡하고, 실습 시간 초과 우려

④ Falco를 이용한 런타임 보안 모니터링

- **목표:** 컨테이너 내부의 의심스러운 행동 실시간 탐지
- **탐지 항목:** 예상치 못한 프로세스 실행, 민감 파일 접근, 네트워크 스캔 시도 등
- **기대효과:** NetworkPolicy로 차단되지 않은 내부 위협도 탐지 가능
- **추가 구현:** Falco 알림을 Slack이나 이메일로 전송하는 자동화 구축

⑤ GitOps 기반 보안 정책 관리

- **목표:** NetworkPolicy, RBAC 등 모든 보안 정책을 Git에서 관리
- **도구:** ArgoCD 또는 FluxCD로 자동 배포
- **기대효과:** 정책 변경 이력 추적, 코드 리뷰를 통한 보안 강화, 롤백 용이
- **확장:** Kyverno 또는 OPA Gatekeeper로 정책 위반 자동 차단

⑥ Multi-Cluster 환경에서의 보안 테스트

- **목표:** 여러 클러스터 간 통신에서의 보안 정책 적용
- **시나리오:** Production, Staging, Development 클러스터 분리 및 격리
- **기술:** Cluster Mesh, Multi-Cluster Service Mesh 활용
- **도전과제:** 비용 증가 및 복잡도 상승 문제

⑦ CI/CD 파이프라인 통합

- **목표:** GitHub Actions 또는 GitLab CI에서 자동 보안 스캔
- **워크플로우:** 코드 푸시 → 이미지 빌드 → Trivy 스캔 → CRITICAL 발견 시 빌드 실패
- **추가 기능:** kube-score, kube-linter로 Kubernetes YAML 파일 정적 분석
- **기대효과:** 취약한 이미지가 프로덕션에 배포되는 것을 사전에 차단

8.3 실습을 통해 얻은 핵심 인사이트

주요 학습 포인트:

1. **이론과 실전의 차이:** 문서로만 배울 때와 직접 구축하면서 마주치는 문제는 차원이 다름. 특히 NetworkPolicy가 적용되지 않는 문제를 해결하면서 EKS와 CNI의 관계를 깊이 이해하게 됨
2. **보안은 계층적 접근:** 단일 도구로는 해결되지 않으며, 이미지 스캔, 네트워크 정책, RBAC, 모니터링 등 여러 층의 방어가 필요
3. **비용 최적화의 중요성:** 클라우드는 편리하지만 방치하면 비용이 빠르게 증가. 실습 환경에서도 비용 의식을 가지는 것이 중요
4. **우선순위 설정:** 363개의 취약점을 모두 해결하려 하지 말고, CRITICAL/HIGH 중심으로 우선순위를 두고 접근해야 효율적
5. **자동화의 필요성:** 수동으로 보안 점검을 하는 것은 비효율적. CI/CD 통합을 통한 자동화가 실전 환경에서는 필수

9. 결론

본 실습을 통해 AWS EKS 환경에서 컨테이너 보안의 여러 층면을 경험했습니다. 특히 **취약점 스캐닝**, **CIS 벤치마크 기반 점검**, 그리고 **NetworkPolicy를 통한 네트워크 세그멘테이션**은 현대적인 Kubernetes 보안의 핵심 요소입니다.

nginx:1.16과 같은 구버전 이미지에서 363개의 취약점이 발견되었으며, 이는 정기적인 이미지 업데이트의 중요성을 보여줍니다. kube-bench를 통한 점검에서는 감사 로그 미활성화, 과도한 권한 부여 등의 문제점을 발견했고, 이는 운영 환경에서 다뤄져야 할 필수 개선 사항입니다.

가장 중요한 성과는 NetworkPolicy를 통해 **라벨 기반 접근 제어**를 구현한 것입니다. 이를 통해 컨테이너 탈취나 내부자 위협 시나리오에서도 피해 범위를 크게 제한할 수 있으며, 측면 이동(Lateral Movement)을 효과적으로 차단할 수 있습니다.

실습 중 마주친 어려움들(Spot Instance 중단, CNI 문제, 결과 해석의 복잡성 등)은 오히려 실전 환경을 미리 경험하는 좋은 기회였습니다. 이러한 시행착오를 통해 이론적 지식이 실제 적용 가능한 실무 역량으로 전환되었습니다.

최종 결론: Kubernetes 보안은 단일 도구나 기술로 해결되지 않습니다. **이미지 스캔**, **설정 점검**, **네트워크 제어**, **RBAC**, **로깅/모니터링** 등 다층 방어(Defense in Depth) 전략을 통해 종합적으로 접근해야 합니다. 본 실습에서 다룬 기법들은 운영 환경에 즉시 적용 가능한 실용적인 보안 조치들이며, 여기서 얻은 경험은 향후 더 복잡한 보안 시스템을 구축하는 데 탄탄한 기반이 될 것입니다.