챕터5 쿠버네티스 보안

Created 2021년 12월 7일 오전 2:28

Tags 비어 있음

시큐리티 컨텍스트

https://kubernetes.io/docs/tasks/configure-pod-container/security-context/

컨테이너 권한에 제한을 걸어둔 파드로 실행

```
apiVersion: v1 kind: Pod metadata: name: security-context-demo spec: securityC
ontext: runAsUser: 1000 runAsGroup: 3000 fsGroup: 2000 volumes: - name: sec-ct
x-vol emptyDir: {} containers: - name: sec-ctx-demo image: busybox command: [
"sh", "-c", "sleep 1h" ] volumeMounts: - name: sec-ctx-vol mountPath: /data/de
mo securityContext: allowPrivilegeEscalation: false
```

파드 생성 후 id와 ps 확인

```
kubectl apply -f https://k8s.io/examples/pods/security/security-context.yaml k
ubectl exec -it security-context-demo -- sh id ps
```

디렉토리 정보확인

```
cd /data ls -l
```

내가 생성한 파일 정보 확인

```
cd demo echo hello > testfile ls -l
```

네트워크와 시스템 시간 관련 커널 권한을 가진 파드 생성

```
apiVersion: v1 kind: Pod metadata: name: security-context-demo-4 spec: contain
ers: - name: sec-ctx-4 image: gcr.io/google-samples/node-hello:1.0 securityCon
text: capabilities: add: ["NET_ADMIN", "SYS_TIME"]
```

일반 컨테이너 생성

```
kubectl run -it --rm --image=gcr.io/google-samples/node-hello:1.0 node sh
```

데이트 명령을 사용해 싱크를 맞춰보자. → 실패

```
date +%T -s "12:00:00"
```

커널 권한을 가진 컨테이너 생성

```
kubectl apply -f https://k8s.io/examples/pods/security/security-context-4.yaml
kubectl exec -it security-context-demo-4 -- sh
```

데이트 명령을 사용해 싱크를 맞춰보자. → 성공

```
date +%T -s "12:00:00"
```

해당 노드에 가서 ntp 동기화를 비활성화

```
timedatectl set-ntp false
```

해당 노드에 ntp 동기화 활성화

```
timedatectl set-ntp yes
```

네트워크 폴리시 테스트하기

https://cloud.google.com/kubernetes-engine/docs/tutorials/network-policy

보호할 새 파드를 구성한다.

```
kubectl run hello-web --labels app=hello \ --image=us-docker.pkg.dev/google-sa
mples/containers/gke/hello-app:1.0 --port 8080 --expose
```

파드를 보호하는 정책을 설정한다.

```
kubectl apply -f https://raw.githubusercontent.com/GoogleCloudPlatform/kuberne
tes-engine-samples/61c260c6e208e54dc7cb586fa77bea9b2bc10f81/network-policies/h
ello-allow-from-foo.yaml
```

네트워크 정책이 정상 동작하는지 확인해보자. CNI → 통신 잘됨

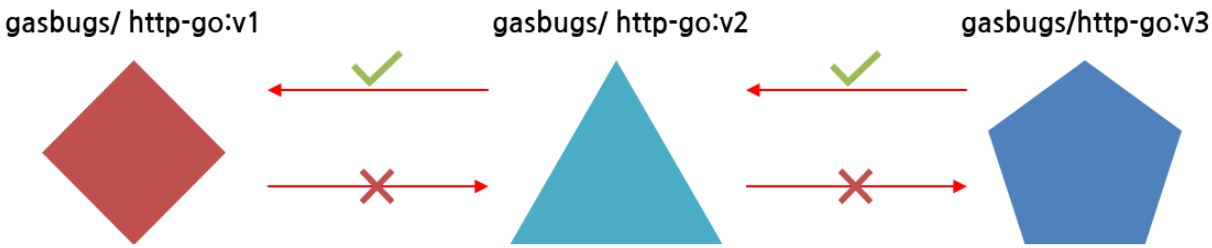
```
kubectl run -l app=foo --image=alpine --restart=Never --rm -i -t test-1 / # wg
et -qO- --timeout=2 http://hello-web:8080 Hello, world! Version: 1.0.0 Hostnam
e: hello-web
```

다른 레이블을 사용하는 경우에는 어떻게 동작하는지 확인해보자. → 통신 안됨

```
kubectl run -l app=other --image=alpine --restart=Never --rm -i -t test-2 / #
wget -qO- --timeout=2 http://hello-web:8080 wget: download timed out
```

네트워크 폴리시 연습문제

다음과 같이 통신이 가능하도록 구성하라(v 표시는 통신 가능, x 표시는 통신 불가).



▼ network-policy-exercise.yaml

```
cat <<EOF | kubectl apply -f - # network-policy-exercise.yaml apiVersion:
v1 kind: Pod metadata: name: http-go-v1 labels: app: http-go-v1 spec: cont
ainers: - name: http-go image: gasbugs/http-go:v1 ports: - containerPort:
8080 --- apiVersion: v1 kind: Pod metadata: name: http-go-v2 labels: app:
http-go-v2 spec: containers: - name: http-go image: gasbugs/http-go:v2 por
ts: - containerPort: 8080 --- apiVersion: v1 kind: Pod metadata: name: htt
p-go-v3 labels: app: http-go-v3 spec: containers: - name: http-go image: g
asbugs/http-go:v3 ports: - containerPort: 8080 --- apiVersion: v1 kind: Se
rvice metadata: name: http-go-v1 spec: selector: app: http-go-v1 ports: -
protocol: TCP port: 80 targetPort: 8080 --- apiVersion: v1 kind: Service m
etadata: name: http-go-v2 spec: selector: app: http-go-v2 ports: - protoco
l: TCP port: 80 targetPort: 8080 --- apiVersion: v1 kind: Service metadat
a: name: http-go-v3 spec: selector: app: http-go-v3 ports: - protocol: TCP
port: 80 targetPort: 8080 EOF
```

▼ 풀이

```
cat <<EOF | kubectl apply -f - # http-go-v1 apiVersion: networking.k8s.io/
v1 kind: NetworkPolicy metadata: name: http-go-v1-network-policy namespace
: default spec: podSelector: matchLabels: app: http-go-v1 policyTypes: - I
ngress ingress: - from: - podSelector: matchLabels: app: http-go-v2 ports:
- protocol: TCP port: 8080 --- # http-go-v2 apiVersion: networking.k8s.io/
v1 kind: NetworkPolicy metadata: name: http-go-v2-network-policy namespace
: default spec: podSelector: matchLabels: app: http-go-v2 policyTypes: - I
ngress ingress: - from: - podSelector: matchLabels: app: http-go-v3 ports:
- protocol: TCP port: 8080 --- # http-go-v3 apiVersion: networking.k8s.io/
v1 kind: NetworkPolicy metadata: name: http-go-v3-network-policy namespace
: default spec: podSelector: matchLabels: app: http-go-v3 policyTypes: - I
ngress EOF
```

쿠버네티스 감사(Audit) 기능 활성화

감사 정책 파일의 예시

```
vim /etc/kubernetes/audit-policy.yaml
```

```
apiVersion: audit.k8s.io/v1 # This is required. kind: Policy # RequestReceived 단계의 모든 요청에 대해 감사 이벤트를 생성하지 말아야 한다. omitStages: - "RequestReceived" rules: # RequestResponse 수준에서 포드 변경 사항 기록 - level: RequestResponse resources: - group: "" # 리소스 "포드"는 RBAC 정책과 일치하는 포드의 하위 리소스에 대한 요청과 일치하지 않습니다. resources: ["pods"] # 메타데이터 수준에서 "pods/log", "pods/status"를 기록합니다. - level: Metadata resources: - group: "" resources: ["pods/log", "pods/status"] # "controller-leader"라는 configmap에 요청을 기록하지 마십시오. - level: None resources: - group: "" resources: ["configmaps"] resourceNames: ["controller-leader"] # 엔드포인트 또는 서비스에서 "system:kube-proxy"에 의한 감시 요청을 기록하지 마십시오. - level: None users: ["system:kube-proxy"] verbs: ["watch"] resources: - group: "" # core API group resources: ["endpoints", "services"] # 리소스가 아닌 특정 URL 경로에 인증된 요청을 기록하지 마세요. - level: None userGroups: ["system:authenticated"] nonResourceURLs: - "/api*" # Wildcard matching. - "/version" # kube-system에서 configmap 변경 사항의 요청 본문을 기록합니다. - level: Request resources: - group: "" # core API group resources: ["configmaps"] # 이 규칙은 "kube-system" 네임스페이스의 리소스에만 적용됩니다. # 빈 문자열 ""을 사용하여 네임스페이스가 없는 리소스를 선택할 수 있습니다. namespaces: ["kube-system"] # 메타데이터 수준에서 다른 모든 네임스페이스의 configmap 및 비밀 변경 사항을 기록합니다. - level: Metadata resources: - group: "" # core API group resources: ["secrets", "configmaps"] # 요청 수준에서 코어 및 확장의 다른 모든 리소스를 기록합니다. - level: Request resources: - group: "" # core API group - group: "extensions" # Version of group should NOT be included. # 메타데이터 수준에서 다른 모든 요청을 기록하는 포괄 규칙입니다. - level: Metadata # Long-running requests like watches that fall under this rule will not # generate an audit event in RequestReceived. omitStages: - "RequestReceived"
```

kube-apiserver.yaml 파일에서 .spec.containers[0].command를 찾아 다음 인자를 추가

```
# /etc/kubernetes/manifests/kube-apiserver.yaml ... - --audit-policy-file=/etc/kubernetes/audit-policy.yaml - --audit-log-path=/var/log/audit.log ...
```

.spec.container[0].volumeMounts와 .spec.volumes에 다음 내용 추가

```
... volumeMounts: - mountPath: /etc/kubernetes/audit-policy.yaml name: audit readOnly: true - mountPath: /var/log/audit.log name: audit-log readOnly: false ... volumes: - name: audit hostPath: path: /etc/kubernetes/audit-policy.yaml type: File - name: audit-log hostPath: path: /var/log/audit.log type: FileOrCreate ...
```

다음 명령을 사용해 큐브시스템 확인

```
$ kubectl get pod -n kube-system
```

마스터 노드에서 /var/log/audit.log 로그 확인

```
tail -f /var/log/audit.log
```

json viewer

http://jsonviewer.stack.hu/

Trivy를 활용한 컨테이너 취약점 진단

도커 컨테이너를 활용한 Trivy 설치

```
docker run --rm -v trivy-cache:/root/.cache/ \ -v /var/run/docker.sock:/var/ru
n/docker.sock aquasec/trivy:latest
```

Trivy에 image 명령을 사용하면 도커 이미지의 취약점을 진단

```
docker run --rm -v trivy-cache:/root/.cache/ \ -v /var/run/docker.sock:/var/ru
n/docker.sock \ aquasec/trivy:latest \ image gasbugs/http-go
```

kube-bench를 활용한 쿠버네티스 보안 점검

kubectl이 실행한 가능한 환경에서 kube-bench를 설치

```
# 릴리즈 파일 다운로드 wget https://github.com/aquasecurity/kube-bench/releases/dow
nload/v0.6.3/kube-bench_0.6.3_linux_amd64.tar.gz tar -xf kube-bench_0.6.3_linu
x_amd64.tar.gz sudo mv kube-bench /usr/bin/ # 설치 완료 # 깃헙 프로젝트 다운로드 git c
lone https://github.com/aquasecurity/kube-bench cd kube-bench
```

마스터 노드 점검하기

```
kube-bench --config-dir `pwd`/cfg --config `pwd`/cfg/config.yaml run --targets
=master
```

워커 노드 점검하기

```
kube-bench --config-dir `pwd`/cfg --config `pwd`/cfg/config.yaml run --targets
=node
```

falco를 활용한 쿠버네티스 컨테이너 보안 모니터링

falco 설치하기

```
curl -s https://falco.org/repo/falcosecurity-3672BA8F.asc | apt-key add - echo
"deb https://download.falco.org/packages/deb stable main" | tee -a /etc/apt/so
urces.list.d/falcosecurity.list apt-get update -y apt-get -y install linux-hea
ders-$(uname -r) apt-get install -y falco
```

python 이미지 컨테이너를 구성하고 패키지 매니저를 통해 vim을 설치

```
kubectl run py --image=python:3.7 -- sleep infinity kubectl exec py -- apt upd
ate kubectl exec py -- apt install -y vim
```

syslog에 falco를 grep하면 관련 로그 확인 가능

```
cat /var/log/syslog | grep falco
```

vim /var/falco/falco.yaml

```
rules_file: - /etc/falco/falco_rules.yaml - /etc/falco/falco_rules.local.yaml
# 사용자 정의 룰 - /etc/falco/k8s_audit_rules.yaml - /etc/falco/rules.d
```