SK infosec

EQST INSIGHT

클라우드 보안 가이드(컨테이너 보안)

-Docker, Kubernetes

클라우드 보안 가이드(컨테이너 보안) - Docker, Kubernetes 발간사

안녕하십니까? SK인포섹 EQST그룹입니다.

최근 ICT 인프라의 주요 이슈는 '클라우드 환경 전환으로 인한 컨테이너화'입니다.
클라우드로 전환하는 기업이 증가하면서 현재의 인프라 운영방식은 한계를 드러 내고 있습니다. 그리고 이를 보완할 수 있는 방안으로 컨테이너화가 떠오르고 있습니다. 컨테이너화는 클라우드로 가는 여정의 필수 전제이며, 컨테이너의 보안은 클라우드 보안과도 직결됩니다.

이번 가이드에서는 클라우드 내 DevOps 환경에서 가장 많이 이용하는 Docker와 Kubernetes의 보안을 다룹니다. 자동화 배포와 표준화 생산 환경으로 반복 작업을 없애는 것이 핵심인 DevOps 환경 구성 시 컨테이너 보안을 어떻게 점검하고 대응할 것인지에 대한 기준도 제시합니다. 그리고 보안 가이드를 기반으로 사용자 환경에서 자체 보안점검이 가능하도록 할 방침입니다.

Docker는 Docker Configuration, Host OS, Image, Docker Swarm 등으로 분류하여 환경 구축 시 안전한 보안 설정을 쉽게 소개하였고, Kubernetes는 Master Node와 Worker Node로 분류해 컨테이터 보안 관리를 위한 설정을 함께 안내하고 있습니다.

올해 EQST그룹은 '클라우드 보안 가이드' 시리즈를 발간하고 있습니다. 금년 1월에 발간한 '클라우드 보안 가이드 - AWS, Cloud Z'는 퍼블릭 클라우드 2종의 보안 정책 설정 방법을 제시한 보안 가이드이며, 이번 가이드는 컨테이너 보안에 초점을 두고 제작한 클라우드 보안 가이드입니다.

향후 SK인포섹 EQST그룹은 퍼블릭 클라우드 중 마이크로소프트 애저(Azure)와 구글 클라우드 플랫폼(GCP) 보안가이드를 발간할 예정입니다. 이를 통해 ICT 인프라 환경에 따라 빠르게 변화하는 보안 위협을 사용자 수준에서 대응할 수 있도록 실무 중심의 보안 가이드 기준을 제시하겠습니다.

EQST그룹 취약점진단팀장
김 희 호

목 차

Part 01. Docker

Part 02. Kubernetes

그 림 목 차

Part 01. Docker

Part 02. Kubernetes

Part 01

- Docker -

I. 전체목록

1. 체크리스트 항목

진단에 사용될 체크리스트는 국내외 기술 자료를 바탕으로 작성하였습니다. Docker 보안 가이드에서의 영역은 Docker Configuration(9개 항목), 호스트 OS(2개 항목), 이미지(3개 항목), Docker Swarm(3개 항목), 기타(3개 항목)로 총 5개 영역에서 20개 항목으로 구성하였습니다.

[표] 1. Docker 보안진단 체크리스트

구분	항목코드	항목명	중요도
Docker Configuration	1-1	호스트 OS 주요 자원 접근 제어	상
	1-2	인증-권한 제어	상
	1-3	SSL/TLS 적용	상
	1-4	네임스페이스 관리	중
	1-5	컨테이너 네트워크 제어	중
	1-6	컨테이너 리소스 제어	하
	1-7	컨테이너 권한 제어	상
	1-8	컨테이너 보안 정책	중
	1-9	로그 관리	하
호스트 OS	2-1	설정 파일 및 주요 디렉터리 권한 설정	하
	2-2	audit 설정	하
이미지	3-1	Dockerfile Config	중
	3-2	이미지 취약점 및 구성 결함	중
	3-3	레지스트리 운영 관리	중
Docker Swarm	4-1	인증제어	상
	4-2	SSL/TLS 적용	상
	4-3	네트워크 제어	중
기타	5-1	중요도 수준에 따른 격리 운영	N/A
	5-2	앱 취약점	N/A
	5-3	보안 패치 적용	중

2. 대응 버전

본 보안 가이드 문서를 이용하여 대응 가능한 Docker 버전은 아래 표와 같습니다.

[표] 2. Docker 보안 가이드 대응 버전

버전 대역	상세 버전	비고
Docker	Docker CE 18.09	

※ 해당 가이드 작성을 위한 테스트 환경은 아래와 같습니다.

- Operating System: CentOS Linux 7 (Core)
- Architecture: x86_64

3. 위험도 및 적용 권고 시기 구분

3.1. 위험도

각 취약점으로 인해 발생 가능한 피해에 대하여 위험도 산정을 통해 상, 중, 하 3단계로 분류하였습니다.

[표] 3. 위험도 구분

위험도	내 용	비고
상	관리자 계정 및 주요 정보 유출로 인한 치명적인 피해 발생	
중	노출된 정보를 통해 서비스/시스템 관련 추가 정보 유출 발생 우려	
하	타 취약점과 연계 가능한 잠재적인 위협 내재	

3.2. 적용 권고 시기

각 취약점 항목의 위험도 및 그에 대한 대응 조치 과정에서 예상되는 서비스/시스템 영향도를 고려하여 단기, 중기, 장기로 적용 권고 시기를 분류하였습니다.

[표] 4. 적용 권고 시기 구분

적용 권고 시기	내 용	비고
단기	연계 서비스/시스템 영향도가 낮으며 빠른 조치가 필요한 경우	
중기(기본)	기본적인 서비스/시스템 영향도 검토가 필요한 경우	
장기	조치로 인한 연계 서비스/시스템 영향이 예상되는 경우	

II. 세부항목 설정

1. Docker Configuration

1.1. 호스트 OS 주요 자원 접근 제어

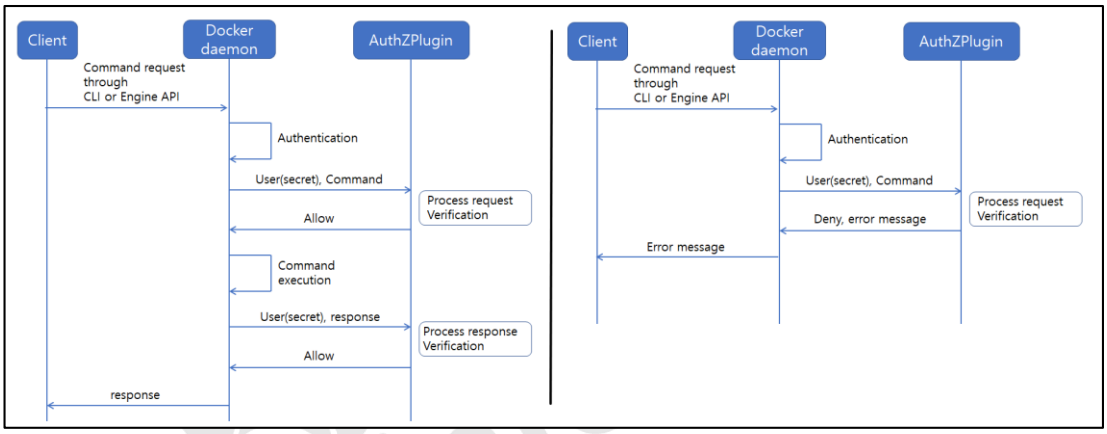
분류	Docker Configuration	중요도	상
항목명	호스트 OS 주요 자원 접근 제어		
항목 설명	컨테이너 구성 시 컨테이너가 호스트 OS 의 주요 자원에 접근 가능하도록 설정된 경우, 컨테이너에서 호스트 OS 의 주요 자원에 대한 수정으로 인해 해당 호스트 및 호스트에서 실행되는 다른 컨테이너의 안정성과 보안에 영향을 미칠 수 있습니다. **- 주요 시스템 디렉터리 마운트 금지** 컨테이너 볼륨 생성 시 호스트 OS 의 주요한 시스템 디렉터리가 읽기-쓰기 모드로 구성될 경우 주요한 디렉터리 안에 있는 파일들에 대한 변경이 될 수 있으므로 읽기-쓰기 모드에서는 주요한 로컬 디렉터리가 컨테이너에 마운트 되지 않도록 설정해야 합니다. **- 호스트 장치 파일 컨테이너에 직접 노출 금지** 가능한 컨테이너에서 호스트의 장치가 직접 노출되지 않도록 설정해야 하며, 서비스 상 필요에 의해 호스트 장치의 공유가 필요한 경우 장치 파일에 대한 권한을 적절히 설정하여 컨테이너에서 호스트 자원에 대한 접근을 최소화해야 합니다.		

진단 방법	**[진단예시]**
	- 주요 시스템 디렉터리 마운트 금지
	: 각 컨테이너에 매핑된 디렉터리의 목록과 권한 확인
	`docker ps --quiet --all | xargs docker inspect --format '{{.Id}}: Volumes={{.Mounts}}'`
	
	- 호스트 장치 파일 컨테이너에 직접 노출 금지
	: 컨테이너 내 불필요하게 접근 가능한 호스트 장치의 존재 여부 및 필요에 의해 접근 가능한 장치에 대한 권한이 올바르게 설정되어 있는지 확인
	`docker ps --quiet --all | xargs docker inspect --format '{{.Id}}: Devices={{.HostConfig.Devices}}'`
	※ 서비스 상 필요할 경우 예외처리 요청		

설정 방법	**- 주요 시스템 디렉터리 마운트 금지**
	`# 주요 시스템 디렉터리`
	`/boot`
	`/dev`
	`/etc`
	`/lib`
	`/proc`
	`/sys`
	`/usr`
	
	- 호스트 장치 파일 컨테이너에 직접 노출 금지(Container Runtime)		

	: 호스트 장치를 컨테이너에 직접 노출하지 말아야 하며 만일 호스트 장치를 컨테이너에 표시하려면 컨테이너 실행 시 올바른 사용 권한을 부여해야 함. 예시) docker run --interactive --tty --device=/dev/tty0:/dev/tty0:rw --device=/dev/temp_sda:/dev/temp_sda:r centos bash
비고	장기 적용(적용 시 개발자 및 운영자 협의)

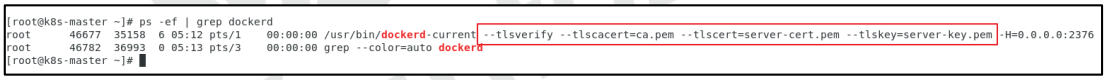
1.2. 인증-권한 제어

분류	Docker Configuration	중요도	상
항목명	인증-권한 제어		
항목 설명	Docker 에서는 기본적으로 사용자 인증을 위한 기능을 지원하고 있지 않기 때문에 기본 설정의 경우 Docker daemon 에 접속할 수만 있으면 Docker 클라이언트에 대한 모든 명령을 실행할 수 있습니다. docker group 내 신뢰할 수 있는 사용자만 존재하는지 확인해야 합니다. 만일 더 높은 수준의 Docker에 대한 인증-권한 제어를 위해서는 authorization plugin을 별도로 설정 후 사용하거나 운영 환경에 따라 Kubernetes, Cloud Foundry 등과 같은 외부의 프로그램 내의 인증-권한 관리 기능을 활용하여 Docker가 인증-권한 제어를 할 수 있도록 구축해야 합니다.  [그림 1] Docker AuthZ Plugins 권한 검증 flow		

진단 방법	**[진단예시]**
	- docker group 내 사용자 확인
	: 아래의 명령어를 실행하여 docker group 내 사용자 확인
	`getent group docker`
	
	- authorization plugin을 사용한 권한 제어
	: 아래의 명령어를 실행하여 authorization-plugin 사용 및 설정값 확인
	`ps -ef | grep dockerd`		

설정 방법	**- docker group 내 사용자 확인** : 조직 내 정책 검토를 통해 신뢰하지 않는 사용자가 존재할 경우 삭제해야 함. **- authorization plugin을 사용한 권한 제어(Docker daemon)** 1) authorization plugin 설치(ex. AuthZPlugin) 2) 각 조직의 구성환경 및 정책에 plugin 작성 3) Docker 서비스 실행 시 작성한 plugin을 아래 명령어를 통해 반영 `dockerd --authorization-plugin=<PLUGIN_ID>`		
비고	장기 적용(적용 시 개발자 및 운영자 협의)		

1.3. SSL/TLS 적용

분류	Docker Configuration	중요도	상
항목명	SSL/TLS 적용		
항목 설명	TLS(Transport Layer Security)은 인터넷에서 정보를 암호화해서 송수신하는 프로토콜로, Netscape Communications 사가 개발한 SSL(Secure Socket Layer)에서 표준화하여, 국제 인터넷 표준화 기구에서 표준으로 인정받은 프로토콜입니다. Docker는 기본적으로 네트워크로 연결되지 않고 UNIX 소켓을 통해 실행됩니다. 만일 Docker daemon을 TCP Socket을 통해 네트워크에 연결 후 원격으로 접속하도록 구축하는 경우에는 SSL/TLS 프로토콜을 적용을 통해 네트워크상에서 secrets 및 keys와 같은 주요 데이터를 스니핑과 같은 방법을 통해 노출되지 않도록 보호하고, TLS 인증서를 활용하여 Docker daemon에 접근하는 클라이언트 또는 사용자 검증을 할 수 있도록 설정해야 합니다. 또한 SSL/TLS 통신 적용 시에는 주기적으로 인증서를 변경하고 안전한 버전의 암호화 방식을 사용하는 방법을 통해 위험을 최소화할 수 있는 정책 설정이 필요합니다.		
진단 방법	**[진단예시]** **- SSL/TLS 적용을 통한 네트워크 구간 데이터 보호 및 사용자 인증** : 아래의 명령어를 실행하여 --tlsverify --tlscacert --tlscert --tlskey 사용 여부 확인 `ps -ef | grep dockerd`  [그림 2] SSL/TLS 적용 관련 인증서 설정 여부		
설정 방법	**- SSL/TLS 적용을 통한 네트워크 구간 데이터 보호 및 사용자 인증(Docker daemon)** 1) OpenSSL을 사용하여 CA, 서버 및 클라이언트 키 생성 2) Docker 서비스 실행 시 옵션을 통한 SSL/TLS 적용 예시) `docker --tlsverify --tlscacert=ca.pem --tlscert=cert.pem --tlskey=key.pem -H=$HOST:2376 version'`		
비고	중기 적용(적용 시 개발자 및 운영자 협의)		

1.4. 네임스페이스 관리

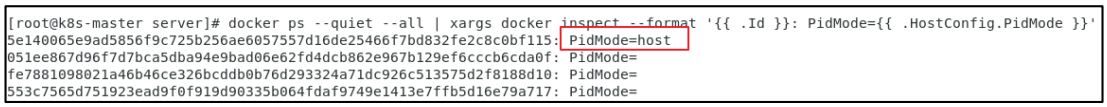
<table>
<tr><td>분류</td><td>Docker Configuration</td><td>중요도</td><td>중</td></tr>
<tr><td>항목명</td><td colspan="3">네임스페이스 관리</td></tr>
<tr><td>항목
설명</td><td colspan="3">Docker 는 컨테이너에 격리된 작업 영역을 제공하기 위해 리눅스에서 지원하고 있는 namespace 라는 시스템 자원을 가상화하여 각각의 독립된 공간을 만들어주는 기술을 사용하고 있습니다.

Docker 는 컨테이너를 실행할 때 해당 컨테이너에 대한 네임스페이스를 생성하고, 컨테이너는 각각의 별도 네임스페이스에서 실행됩니다.

하지만 이러한 컨테이너의 격리는 완전한 격리가 아닌 결국 호스트 OS 안에서 실행되는 논리적 격리이므로, 컨테이너 내에서 호스트 OS 로의 권한 상승 공격을 방지하기 위해서는 컨테이너가 불필요하게 과도한 권한을 가지지 않도록 설정해야 합니다.

■ Docker Engine 에서 사용하는 리눅스 네임스페이스
<table><tr><th>구분</th><th>설명</th></tr><tr><td>Process namespace (PID)</td><td>프로세스를 격리하여 독립적인 프로세스 공간을 할당</td></tr><tr><td>Network namespace (NET)</td><td>네트워크 인터페이스 격리</td></tr><tr><td>IPC namespace</td><td>프로세스 간 IPC 격리 및 관리</td></tr><tr><td>Mount namespace (mnt)</td><td>마운트 지점을 제어하여 네임스페이스 별 격리</td></tr><tr><td>UTS namespace</td><td>독립적인 hostname 및 domainname 할당</td></tr><tr><td>User namespace</td><td>새로운 uid/gid (root) 할당</td></tr></table></td></tr>
<tr><td>진단
방법</td><td colspan="3">[진단예시]
- 호스트의 network namespace 공유 금지
: 아래의 명령어를 실행하여 네임 스페이스 공유 여부 확인
(NetworkMode=host 일 경우 공유, NetworkMode=는 비공유)
<code>docker ps --quiet --all | xargs docker inspect --format '{{.Id}}: NetworkMode={{.HostConfig.NetworkMode}}'</code>

- 호스트의 process namespace 공유 금지
: 아래의 명령어를 실행하여 네임 스페이스 공유 여부 확인
(PodMode=host 일 경우 공유, PodMode=는 비공유)
<code>docker ps --quiet --all | xargs docker inspect --format '{{.Id}}: PidMode={{.HostConfig.PidMode}}'</code>

[그림 3] process namespace 공유 설정 확인

- 호스트의 IPC namespace 공유 금지
: 아래의 명령어를 실행하여 네임 스페이스 공유 여부 확인
(IpcMode=host 일 경우 공유, IpcMode=는 비공유)
<code>docker ps --quiet --all | xargs docker inspect --format '{{.Id}}: IpcMode={{.HostConfig.IpcMode}}'</code></td></tr>
</table>

- 호스트의 UTS namespace 공유 금지

: 아래의 명령어를 실행하여 네임 스페이스 공유 여부 확인
(UTSMode=host 일 경우 공유, UTSMode=는 비공유)

```
docker ps --quiet --all | xargs docker inspect --format '{{.Id}}:
UTSMode={{.HostConfig.UTSMode}}'
```

```
[root@k8s-master server]# docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: UTSMode={{ .HostConfig.UTSMode }}'
20e0b4978fff4870c7eba183bc06524d83cc2749037fcc03aa0016b1558be841: UTSMode=host
902348d6afd2631e2bfe73cd08f6713e98112d281d17f8448c9700c6a9050713: UTSMode=host
b6e8d1c66bad50561e2061aac9b0905de3cbc5fe5ebd8708b8a4d80a66ae4f42: UTSMode=
3d9e34c6aff5d875b389f9fdde93166dd6aca687fa1dd33dc24ab1825f48784c: UTSMode=host
```

[그림 4] UTS namespace 공유 설정 확인

- 호스트의 user namespace 공유 금지

: 아래의 명령어를 실행하여 네임 스페이스 공유 여부 확인
(Userns=host 일 경우 공유, Userns=는 비공유)

```
docker ps --quiet --all | xargs docker inspect --format '{{.Id}}:
UsernsMode={{.HostConfig.UsernsMode}}'
```

- user namespace support 사용

: 아래의 명령어를 실행하여 컨테이너 프로세스가 root 사용 여부 확인
(root 일 경우 user namespace support 사용은 권장하지 않음.)

```
ps -p $(docker inspect --format='{{.State.Pid}}' <컨테이너 ID>) -o pid,user
```

```
[root@k8s-master server]# ps -p $(docker inspect --format='{{ .State.Pid }}' f05a6c344c20) -o pid,user
  PID USER
41174 root
[root@k8s-master server]#
```

[그림 5] user namespace support 사용 여부 확인

설정 방법

- 호스트의 network namespace 공유 금지(Container Runtime)

: 컨테이너 실행 시 아래의 옵션을 사용하지 않음

```
--net=host
```

- 호스트의 process namespace 공유 금지(Container Runtime)

: 컨테이너 실행 시 아래의 옵션을 사용하지 않음

```
--pid=host
```

- 호스트의 IPC namespace 공유 금지(Container Runtime)

: 컨테이너 실행 시 아래의 옵션을 사용하지 않음

```
--ipc=host
```

- 호스트의 UTS namespace 공유 금지(Container Runtime)

: 컨테이너 실행 시 아래의 옵션을 사용하지 않음

```
--uts=host
```

- 호스트의 user namespace 공유 금지(Container Runtime)

: 컨테이너 실행 시 아래의 옵션을 사용하지 않음

	--userns=host **- user namespace support 사용(Docker daemon)** 1) 조직 내 정책을 고려하여 /etc/subuid, /etc/subgid 파일 생성 및 정의 2) 아래 명령어를 통해 Docker 서비스 실행 --userns-remap=default 3) Docker 서비스 재시작 service docker restart
비고	중기 적용(적용 시 개발자 및 운영자 협의)

1.5. 컨테이너 네트워크 제어

<table>
<tr><td>분류</td><td>Docker Configuration</td><td>중요도</td><td>중</td></tr>
<tr><td>항목명</td><td colspan="3">컨테이너 네트워크 제어</td></tr>
<tr><td>항목
설명</td><td colspan="3">
- bridge 방식의 default 네트워크 제한
Docker 구현 시 기본적으로 적용되어 있는 bridge 방식의 네트워크를 사용할 경우 호스트 내 각각의 컨테이너가 독립적인 네트워크 영역을 갖지 않고 호스트의 네트워크를 함께 사용하게 되어, 결과적으로 하나의 컨테이너가 동일한 호스트의 모든 네트워크의 패킷을 확인할 수 있기 때문에 의도치 않는 정보 유출의 가능성이 존재합니다.

■ Docker 컨테이너 네트워크
<table>
<tr><th>구분</th><th>설명</th></tr>
<tr><td>bridge</td><td>▷ Docker 에서 기본 설정되는 네트워크 방식, 각각의 컨테이너가 고유한 network namespace 를 가지고 있음
▷ 각 컨테이너의 네트워크 인터페이스들은 docker0 라는 공용 bridge 에 연결</td></tr>
<tr><td>host</td><td>▷ 컨테이너가 독립적인 네트워크 영역을 갖지 않고 host 의 네트워크를 함께 사용
※ Docker 17.06 이상의 swarm 서비스에서만 사용 가능</td></tr>
<tr><td>overlay</td><td>▷ 물리적으로 떨어진 서버에서 Docker 만의 네트워크를 구성하기 위한 네트워크
▷ 여러 Docker daemon 간 연결 또는 swarm 서비스 통신을 위해 사용</td></tr>
<tr><td>macvlan</td><td>▷ MAC 주소를 컨테이너에 할당하여 대상 컨테이너 네트워크를 물리적 장치로 사용
▷ Docker 데몬은 트래픽을 컨테이너의 MAC 주소로 라우팅</td></tr>
<tr><td>none</td><td>▷ 모든 네트워킹을 비활성화하며 인터페이스가 없는 상태로 컨테이너를 생성
▷ 일반적으로 커스텀 네트워크 드라이버와 함께 사용</td></tr>
</table>
- 불필요한 포트 매핑 금지
Dockerfile 에 포트를 정의하거나 런타임 명령을 통해 컨테이너에서 사용하는 포트를 오픈할 수 있습니다. 만약 컨테이너에 불필요하게 포트가 오픈된 경우 컨테이너에 대해 FTP / SSH 를 통해 시스템에 접근하여 저장된 주요정보를 획득하거나 침입 후 권한 상승 등을 통해 시스템을 장악하는 것과 같은 공격에 노출될 가능성이 존재하므로 설정 확인을 통해 불필요한 포트가 오픈되지 않도록 하여야 합니다.

- 호스트 네트워크 인터페이스 설정
만약 호스트 시스템에서 여러 개의 네트워크 인터페이스가 존재하는 경우 기본적으로 모든 인터페이스에서 Docker의 컨테이너에 접근이 가능하여, 보안이 적용되지 않은 경로를 통해 접근할 수 있습니다. 특정 외부 인터페이스에서 들어오는 연결만 허용하도록 설정해야 합니다.

- 컨테이너 내 SSH 실행 금지
컨테이너에 SSH 서버 실행 및 접속을 허용하는 경우 SSH 버전 취약점과 SSH 보안 관리의 복잡성이 증가하며, 비인가자의 접근을 통한 컨테이너 변경 및 정보 노출 등의 가능성이 존재하기 때문에 컨테이너 내에서 SSH 가 실행되지 않도록 설정해야 합니다.

만약 서비스로 인해 SSH 접속을 허용해야 하는 경우에는 root 로그인을 비활성화해서 사용하여야 합니다.

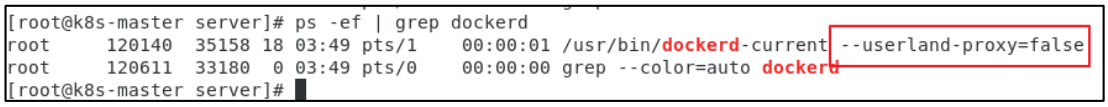
- Userland 프록시 사용 제한
</td></tr>
</table>

	Docker 엔진은 내부 통신을 위해 호스트에서 컨테이너로 포워딩 시 userland proxy 또는 iptables을 활용한 hairpin NAT를 사용할 수 있습니다. 일반적으로 성능 향상을 이유로 userland proxy 대신 hairpin NAT를 많이 사용하고 있으며, 만일 hairpin NAT를 사용할 경우 userland proxy는 일반적으로 불필요하기 때문에 비활성화하여 해당 기능을 통한 위험을 최소화해야 합니다.

진단 방법	**[진단예시]**
	- bridge 방식의 default 네트워크를 제한
	1) 아래의 명령어를 실행하여 bridge 방식의 default 네트워크 사용 여부 확인
	`docker network ls --quiet | xargs docker network inspect --format '{{.Name}}: {{.Options}}'` // true: 제한 없음, false: 제한
	
	2) 아래의 명령어를 실행하여 docker0 인터페이스 사용 여부 확인
	`docker network ls --quiet | xargs xargs docker network inspect --format '{{.Name}}: {{.Options}}'`
	
	[그림 6] docker0 인터페이스 사용 여부 확인
	
	- 불필요한 포트 매핑 금지
	: 아래의 명령어를 실행하여 1024 이하 포트 또는 서비스 운영 시 불필요한 포트가 매핑되어 있는지 확인
	`docker ps --quiet | xargs docker inspect --format '{{.Id}}: Ports={{.NetworkSettings.Ports}}'`
	
	[그림 7] 매핑된 포트 확인
	
	- 호스트 네트워크 인터페이스 설정
	: 아래의 명령어를 실행하여 컨테이너 포트가 0.0.0.0이 아닌 특정 인터페이스에 연결되어 있는지 확인
	`docker ps --quiet | xargs docker inspect --format '{{.Id}}: Ports={{.NetworkSettings.Ports}}'`
	※ 호스트의 네트워크 인터페이스가 2개 이상인 경우 위 설정 제외
	
	- 컨테이너 내 SSH 실행 금지
	1) 아래의 명령어 실행을 통해 컨테이너의 실행 중인 INSTANCE 조회
	`docker ps --quiet`
	
	2) 아래의 명령어 실행을 SSH에 대한 프로세스 존재 여부 확인
	`docker exec $INSTANCE_ID ps -el`

<table>
<tr>
<td></td>
<td>
- Userland 프록시 사용 제한
1) 아래의 명령어 실행
<code>ps -ef | grep dockerd</code>

2) 파라미터 존재 여부 및 적절한 인자 값 설정 여부 확인
<code>--userland-proxy</code>



[그림 8] userland-proxy 활성화 여부 확인

- 컨테이너에 docker.sock 마운트 금지
: 아래의 명령어를 실행하여 docker.sock 파일의 마운트 여부 확인
<code>docker ps --quiet --all | xargs docker inspect --format '{{.Id}}: Volumes={{.Mounts}}' | grep docker.sock</code>
</td>
</tr>
<tr>
<td>설정
방법</td>
<td>
- bridge 방식의 default 네트워크를 제한(Container Runtime)
1) 아래의 명령어 사용하여 bridge 방식의 default 네트워크 제한
<code>dockerd --icc=false</code>

2) 사용자 정의 네트워크를 별도로 설정해서 사용

- 불필요한 포트 매핑 금지(Container Runtime)
: Dockerfile 내 권한이 부여된 포트 매핑 선언을 하는지 확인하고, 컨테이너 시작할 때 권한이 부여된(1024 이하) 포트에 매핑하지 말아야 함.
또한 컨테이너 실행 시 -p or --publish 옵션을 사용, 컨테이너 인스턴스에 필요한 포트를 명시하여 사용해야 함.
예시)
<code>docker run --interactive --tty --publish 5000 --publish 5001 --publish 5002 centos /bin/bash</code>

- 호스트 네트워크 인터페이스 설정(Container Runtime)
: 컨테이너 실행 시 아래의 명령어를 통해 컨테이너 포트를 특정 호스트 인터페이스에만 연결하도록 설정
예시) 컨테이너의 80번 포트를 “10.2.3.4”인 호스트 인터페이스의 49153 포트에 연결
<code>docker run --detach --publish 10.2.3.4:49153:80 nginx</code>

- 컨테이너 내 SSH 실행 금지
: 컨테이너에서 SSH 서버를 제거

- Userland 프록시 사용 제한(Docker daemon)
: Docker 서비스 실행 시 아래의 옵션을 통한 실행
<code>dockerd --userland-proxy=false</code>
</td>
</tr>
</table>

	- 컨테이너에 docker.sock 마운트 금지 : 컨테이너 내에서 아래 명령어를 실행하여 mount된 docker.sock 삭제 `rm -rf /var/run/docker.sock`
비고	장기 적용(적용 시 개발자 및 운영자 협의)

1.6. 컨테이너 리소스 제어

분류	Docker Configuration	중요도	하
항목명	컨테이너 리소스 제어		
항목 설명	기본적으로 모든 컨테이너는 호스트의 리소스를 동일하게 공유하고 있지만, 설정에 따라 CPU, 메모리, 저장공간과 같은 리소스를 컨테이너 별로 차등해서 공유할 수 있습니다. 컨테이너 내 애플리케이션의 중요도 및 기능에 따라 리소스 관리를 하여 특정 컨테이너에 과도하게 리소스가 할당되거나 또는 리소스가 부족하여 서비스 거부 공격(Distributed DoS)이 발생하지 않도록 설정하여야 합니다.		

진단 방법	**[진단예시]**
	- 컨테이너 크기 설정
	1) 아래의 명령어를 실행하여 --storage-opt dm.basesize 값 확인 후 컨테이너별 할당 된 공간 확인
	`ps -ef | grep dockerd`
	
	2) 아래의 명령어를 실행하여 마운트 된 볼륨을 공유하는지 확인
	`docker ps --quiet --all | xargs docker inspect --format '{{.Id}}: Propagation={{range $mnt := .Mounts}} {{json $mnt.Propagation}} {{end}}'`
	
	- 메모리 설정
	: 아래의 명령어를 실행하여 메모리가 적절하게 할당되어 있는지 확인
	(memory=0이면 제한 없음)
	`docker ps --quiet --all | xargs docker inspect --format '{{.Id}}: Memory={{.HostConfig.Memory}}'`
	
	`[root@k8s-master server]# docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: Memory={{ .HostConfig.Memory }}'`
	`28fab03364ec1d47ee17bd35aab5367b2eb7dc6d9380513948848345e495b923: Memory=2.68435456e+08`
	`bc30ba8bdb275029971b4f4fd539485fbc664b75df0e64615950c64e8da82651: Memory=0`
	`23a82c6a3cef890a386b152b9caa366214f4f4c3f80ae00af859dd1e9289c75e: Memory=0`
	`72f73cae026ef1080e328669e69273055516bf1d6313b444a1e96928da397d41: Memory=0`
	`d1eab4955d28ec1fbcd5a9faf1d8326f434ec5eb53665d1ee03c2951610f624f: Memory=0`
	`ebe837bf48169138683d4728fe10460a9300cc57cdb6b928354403849c09642e: Memory=0`
	[그림 9] 컨테이너에 할당된 메모리 확인
	
	- CPU 설정
	: 아래의 명령어를 실행하여 CPU에 대한 우선순위가 적절하게 할당되어 있는지 확인
	`docker ps --quiet --all | xargs docker inspect --format '{{.Id}}: CpuShares={{.HostConfig.CpuShares}}'`
	
	`[root@k8s-master ~]# docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: CpuShares={{ .HostConfig.CpuShares }}'`
	`8c64c15a5c7fba25e28b9e4fd9aaed350c6c7021f1e6555fd3ed94ff3fd47648: CpuShares=512`
	`aa22870ab3710246958f6a26d91dd1245478374d4a31c235f8e7f915157d0e5d: CpuShares=2`
	`2ecfca2a440625111e2359a364892e1ddaf28816c58f8248014502ade8008843: CpuShares=256`
	`28fab03364ec1d47ee17bd35aab5367b2eb7dc6d9380513948848345e495b923: CpuShares=0`
	`72f73cae026ef1080e328669e69273055516bf1d6313b444a1e96928da397d41: CpuShares=102`
	`d1eab4955d28ec1fbcd5a9faf1d8326f434ec5eb53665d1ee03c2951610f624f: CpuShares=204`
	`ebe837bf48169138683d4728fe10460a9300cc57cdb6b928354403849c09642e: CpuShares=2`
	`fffa9eeb04a247598ccecc9c463a3c230369c49dc823a07817e061ff5c596f4f: CpuShares=2`
	`d5c571797f2956457e2ab657767fa5488649f4fef9dc16408f2113aa8eed807a: CpuShares=2`
	[그림 10] 컨테이너별 CPU 우선순위 확인
	
	- 프로세스 제한 설정
	1-1) ulimit - docker daemon 설정
	: 아래의 명령어를 실행하여 --default-ulimit 파라미터값이 적절하게 설정되어 있는지 확인		

	ps -ef | grep dockerd 1-2) ulimit - docker runtime 설정 : 아래의 명령어를 실행하여 Ulimits 설정 여부 확인 docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: Ulimits={{.HostConfig.Ulimits }}' 2) 프로세스 생성 제한 : 아래의 명령어를 실행하여 제한 여부 확인 docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: PidsLimit={{.HostConfig.PidsLimit }}' **- 컨테이너 재시작 횟수 설정** : 아래의 명령어를 실행하여 재시작 횟수 설정 여부 확인 (RestartPolicyName 값이 always, n, null인 경우 재시작 횟수 미설정) docker ps --quiet --all | xargs docker inspect --format '{{.Id }}: RestartPolicyName={{.HostConfig.RestartPolicy.Name }} MaximumRetryCount={{.HostConfig.RestartPolicy.MaximumRetryCount }}' [root@k8s-master server]# docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: RestartPolicyName={{ .HostConfig.RestartPolicy.Name }} MaximumRetryCount={{ .HostConfig.RestartPolicy.MaximumRetryCount }}' f05a6c344c205f21886a713d037c490a965d22d22304f7026ed2cc81a0ea7907: RestartPolicyName=on-failure MaximumRetryCount=5 e5feed806679fe226dfc899c00aef93b831dfc9fc954ed5eb2eeeaa26b7a0c45: RestartPolicyName=no MaximumRetryCount=0 4a565ceeea11f23867d7192dfbbd520d3e7eaba88f87a479483b57bdf5d0ba7a: RestartPolicyName=no MaximumRetryCount=0 8c64c15a5c7fba25e28b9e4fd9aaed350c6c7021f1e6555fd3ed94ff3fd47648: RestartPolicyName=no MaximumRetryCount=0 28fab03364ec1d47ee17bd35aab5367b2eb7dc6d9380513948848345e495b923: RestartPolicyName=no MaximumRetryCount=0 72f73cae026ef1080e328669e69273055516bf1d6313b444a1e96928da397d41: RestartPolicyName=no MaximumRetryCount=0 [그림 11] 컨테이너별 재시작 횟수 확인
설정 방법	**- 컨테이너 크기 설정** : 가급적 컨테이너 운영 시 별도의 파티션을 생성하고, 서비스 환경 분석을 통해 필요하지 않다면 Docker 서비스 실행 시 아래 옵션값을 사용한 컨테이너 볼륨 사이즈를 제한하지 않는 것을 권고 --storage-opt dm.basesize 또한 마운트 된 볼륨 내 수정 사항이 공유된 모든 컨테이너에 반영이 되는 mount propagation mode를 사용할 경우 의도치 않게 볼륨 내 파일이나 설정이 변경되거나 정보 유출의 가능성이 존재하므로 서비스 운영상 꼭 필요한 경우가 아닌 경우 해당 기능을 통해 공유되지 않도록 설정 예시) 컨테이너 실행 시 마운트 된 볼륨을 공유하도록 설정 docker run 〈Run arguments〉 --volume=/hostPath:/containerPath:shared 〈Container Image Name or ID〉 〈Command〉 **- 메모리 설정(Container Runtime)** : 컨테이너 실행 시 아래의 옵션 통해 사용자 환경에 맞게 설정 --memory 예시) docker run --interactive --tty --memory 256m centos /bin/bash **- CPU 설정(Container Runtime)**

	: 컨테이너 실행 시 아래의 옵션 통해 사용자 환경에 맞게 설정
	`--cpu-shares`
	
	예시)
	`docker run --interactive --tty --cpu-shares 512 centos /bin/bash`
	
	- 프로세스 제한 설정
	1-1) ulimit - docker daemon 설정
	: Docker 서비스 실행 시 아래의 옵션 통해 프로세스 제한 설정
	`--default-ulimit`
	
	예시)
	`dockerd --default-ulimit nproc=1024:2048 --default-ulimit nofile=100:200`
	※ 서비스상 필요할 경우 예외처리 요청
	
	1-2) ulimit - docker runtime 설정
	: 필요한 경우 아래 예시와 같은 명령을 통해 컨테이너 실행 시 Docker daemon에서 설정한 ulimit을 무시하도록 설정
	예시)
	`docker run --ulimit nofile=1024:1024 --interactive --tty centos /bin/bash`
	※ 서비스상 필요할 경우 예외처리 요청
	
	2) 프로세스 생성 제한
	: 컨테이너 실행 시 사용자 환경에 맞게 아래의 옵션 통해 설정
	`--pids-limit flag`
	
	- 컨테이너 재시작 횟수 설정(Container Runtime)
	: 컨테이너 실행 시 아래의 옵션을 통해 재시작 횟수 설정(5회 권고)
	`-restart=on-failure`

비고	장기 적용(적용 시 개발자 및 운영자 협의)

1.7. 컨테이너 권한 제어

<table>
<tr><td>분류</td><td>Docker Configuration</td><td>중요도</td><td>상</td></tr>
<tr><td>항목명</td><td colspan="3">컨테이너 권한 제어</td></tr>
<tr><td>항목
설명</td><td colspan="3">컨테이너 런타임 또는 이미지에서 불필요하게 과도한 권한이 설정된 경우 잠재적인 권한 상승 공격에 악용될 가능성이 존재하므로 런타임 또는 이미지 내 불필요한 권한을 삭제하여, Docker 내 컨테이너가 과도한 권한을 가지고 실행되지 않도록 하여야 합니다.

■ Privileged/ Unprivileged 컨테이너
<table>
<tr><th>구분</th><th>설명</th></tr>
<tr><td>Privileged 컨테이너</td><td>▷ 컨테이너의 uid0 이 호스트의 uid0 로 매핑된 컨테이너
▷ 호스트 보호와 탈출 방지를 위한 apparmor, selinux, seccomp, 네임스페이스, 캐퍼빌리티 제거와 같은 방법이 있으나 root 권한을 이용한 공격에 안전하지 않음
▷ 신뢰할 수 있는 작업만 수행되거나 root 로는 신뢰할 수 없는 작업이 실행되지 않는 환경에서 사용</td></tr>
<tr><td>Unprivileged 컨테이너</td><td>▷ 컨테이너의 uid0 이 컨테이너 밖의 비특권 사용자로 매핑
▷ 자기 자신의 자원만 접근할 수 있는 권한만 존재</td></tr>
</table></td></tr>
<tr><td>진단
방법</td><td colspan="3">[진단예시]
- suid/sgid 제한
1) docker daemon 설정
: 아래의 명령어를 실행하여 --no-new-privileges 값이 적절하게 설정되어 있는지 확인
(false 일 경우 suid/sgid 허용)
ps -ef | grep dockerd

2) docker runtime 설정
: 아래의 명령어를 실행하여 --security-opt 값이 적절하게 설정되어 있는지 확인
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}:
SecurityOpt={{ .HostConfig.SecurityOpt }}'

- 실험(Experimental) 기능 비활성화
: 아래의 명령어를 실행하여 실험(Experimental) 기능이 활성화되어 있는지 확인
docker version --format '{{ .Server.Experimental }}'

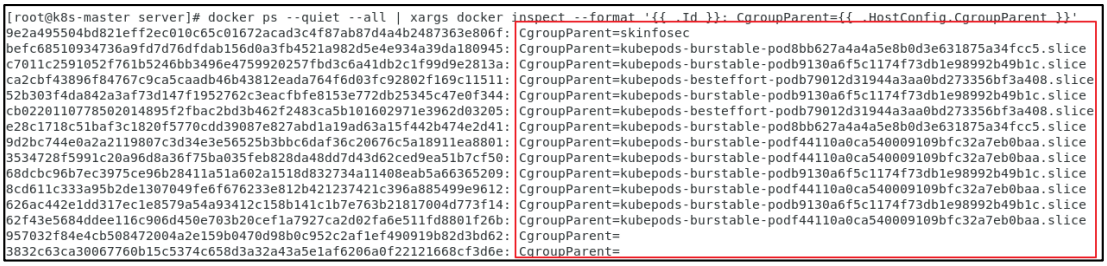
- cgroup 변경 금지
1) docker daemon 설정
: 아래의 명령어를 실행하여 --cgroup-parent 및 적절한 cgroup 설정 여부 확인
ps -ef | grep dockerd

[그림 12] cgroup 확인(docker daemon)

2) docker runtime 설정</td></tr>
</table>

: 아래의 명령어를 실행하여 사용하여 디폴트 cgroup 또는 그 외의 cgroup 설정 여부 확인

```
docker ps --quiet --all | xargs docker inspect --format '{{.Id}}:
CgroupParent={{.HostConfig.CgroupParent}}'
```



[그림 13] cgroup 확인(docker runtime)

- privilege 컨테이너 사용 제한

: 아래의 명령어를 실행하여 컨테이너 권한 여부 확인

```
docker ps --quiet --all | xargs docker inspect --format '{{.Id}}:
Privileged={{.HostConfig.Privileged}}'
```

[그림 14] privilege 컨테이너 존재 여부 확인

- exec 사용 제한

1) 아래의 명령어를 실행하여 exec와 privileged 옵션 동시 사용 여부 확인

```
ausearch -k docker | grep exec | grep privileged
```

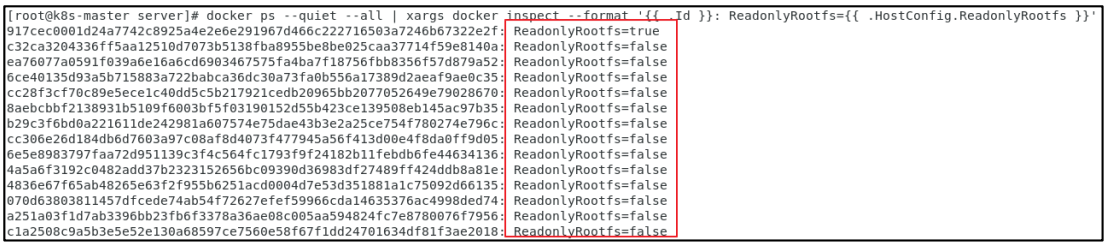
2) 아래의 명령어를 실행하여 exec와 user 옵션 동시 사용 여부 확인

```
ausearch -k docker | grep exec | grep user
```

- 컨테이너의 Root Filesystem을 read only으로 설정

: 아래의 명령어를 실행하여 컨테이너의 Root Filesystem 상태 확인
(true=read only, flase=read/write)

```
docker ps --quiet --all | xargs docker inspect --format '{{.Id}}:
ReadonlyRootfs={{.HostConfig.ReadonlyRootfs}}'
```



[그림 15] Root Filesystem 권한 확인

설정 방법	**- suid/sgid 제한** 1) docker daemon 설정**(Docker daemon)**

	: Docker 서비스 실행 시 아래의 옵션 통해 suid/sgid 제한
	`--no-new-privileges`
	
	예시)
	`dockerd --no-new-privileges`
	
	2) docker runtime 설정**(Container Runtime)**
	: 컨테이너 실행 시 아래의 옵션 통해 suid/sgid 제한
	`--security-opt=no-new-privileges`
	
	예시)
	`docker run --rm -it --security-opt=no-new-privileges ubuntu bash`
	
	- 실험 기능 비활성화(Docker daemon)
	: 필요한 경우가 아닐 경우 아래의 옵션을 통한 Docker 서비스 실행 금지
	`--experimental`
	
	- cgroup 변경 금지
	1) docker daemon 설정**(Docker daemon)**
	: 운영 환경에 대한 검토 후 cgroup 변경 필요 시 Docker 서비스를 실행할 때 아래의 옵션을 사용하여 cgroup 변경
	`--cgroup-parent`
	
	예시)
	`dockerd --cgroup-parent=/foobar`
	
	2) docker runtime 설정**(Container Runtime)**
	: 필요한 경우가 아닐 경우 아래의 옵션을 사용하여 컨테이너 실행 금지
	`--cgroup-parent`
	
	- privilege 컨테이너 사용 제한(Container Runtime)
	: 필요한 경우가 아닐 경우 아래의 옵션을 사용하여 컨테이너 실행 금지
	`--privileged`
	
	- exec 사용 제한(Container Runtime)
	: 컨테이너 실행 시 exec와 -privileged, --user 옵션을 동시에 사용하지 않도록 설정
	
	- 컨테이너의 Root Filesystem을 read only로 설정(Container Runtime)
	: 컨테이너 실행 시 아래 명령어를 통해 read only로 설정
	`docker run 〈Run arguments〉 --read-only 〈Container Image Name or ID〉 〈Command〉`

비고	중기 적용(적용 시 개발자 및 운영자 협의)

1.8. 컨테이너 보안 정책

<table>
<tr><td>분류</td><td>Docker Configuration</td><td>중요도</td><td>중</td></tr>
<tr><td>항목명</td><td colspan="3">컨테이너 보안 정책</td></tr>
<tr><td>항목
설명</td><td colspan="3">AppArmor, SELinux, seccomp 와 같은 리눅스 커널에서 지원하는 보안 정책 적용을 통해 컨테이너 내 리눅스 OS 및 응용프로그램을 보호하고 호스트와 컨테이너 간 컨테이너 권한 상승으로 인해 호스트가 손상되는 것을 방지할 수 있습니다.

■ 리눅스 커널 보안 모듈
<table>
<tr><th>구분</th><th>설명</th></tr>
<tr><td>AppArmor</td><td>▷ 시스템 관리자가 프로그램별 프로필을 기반으로 권한 제어
▷ 프로필을 통해 네트워크 액세스, raw 소켓 액세스 그리고 파일의 읽기, 쓰기, 실행 같은 권한 설정 가능</td></tr>
<tr><td>SELinux</td><td>▷ 강제 접근 통제(MAC)와 같은 접근 제어 보안 정책 매커니즘을 제공하는 커널 레벨의 보안 모듈</td></tr>
<tr><td>seccomp</td><td>▷ 리눅스 커널 레벨에서 어플리케이션 샌드박싱 메커니즘을 제공하는 보안 기능, 기본적으로 프로세스가 호출할 수 있는 system call 을 read(), write(), exit(), sigreturn()로 제한</td></tr>
</table></td></tr>
<tr><td>진단
방법</td><td colspan="3">[진단예시]
- seccomp 프로필을 적용
1-1) 아래의 명령을 실행하여 default seccomp 프로필 사용 여부 확인
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: SecurityOpt={{.HostConfig.SecurityOpt}}'

1-2) 아래의 명령을 실행하여 사용자 정의 seccomp 프로필 적용 여부 확인
docker info --format '{{.SecurityOptions}}'

[root@k8s-master server]# docker info --format '{{ .SecurityOptions }}'
[name=seccomp,profile=default]
[root@k8s-master server]#
[그림 16] seccomp 프로필 적용 여부 확인

- AppArmor 프로필 활성화
: 아래의 명령을 실행하여 AppArmor 프로필 적용 여부 확인
docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: AppArmorProfile={{.AppArmorProfile}}'

[root@k8s-master etc]# docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: AppArmorProfile={{ .AppArmorProfile }}'
e1315c02c81cf470ff715ec6c1296732178966b147d2d02979e6451cccb7445d: AppArmorProfile=
c8449ea85720745c7cae548b17c94869c7745b5acf2e5265675fe0338beb162e: AppArmorProfile=
9ffa5bffa92c2646453bd90621e6646b170df9934ed2960cb9fdbbb19bd9c021: AppArmorProfile=skinfosec
67b967c3374c4557c9a6eda642611f11c66099f6fd988442884dc7c2409a113f: AppArmorProfile=PROFILENAME
[그림 17] AppArmor 프로필 적용 여부 확인

- SELinux 사용
: 아래의 명령을 실행하여 SELinux 사용 여부 확인</td></tr>
</table>

	`docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: SecurityOpt={{.HostConfig.SecurityOpt}}'`
	
	- 컨테이너 내에서 리눅스 커널 Capabilities 제한
	: 아래의 명령을 실행하여 필요 없는 커널 기능 존재 여부 확인
	`docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: CapAdd={{.HostConfig.CapAdd}} CapDrop={{.HostConfig.CapDrop}}'`

설정 방법	**- seccomp 프로필을 적용(Docker daemon)**
	: 특별한 경우가 아닌 경우 가급적 default seccomp 프로필을 사용하는 것을 권고하며, Docker 서비스 실행 시 아래의 명령어를 통해 seccomp 프로필 적용
	`dockerd --seccomp-profile <seccomp 프로파일 파일 경로>`
	
	- AppArmor 프로필 활성화(Container Runtime)
	1) docker 컨테이너를 위한 apparmor 프로필 생성
	
	2) 아래의 옵션을 통한 컨테이너 실행
	`--security-opt="apparmor=<생성한 apparmor 프로필>"`
	
	- SELinux 사용(Container Runtime)
	1) SELinux 정책을 설정
	
	2) Docker 컨테이너에 대한 SELinux 정책 템플릿 생성
	
	3) SELinux가 활성화 된 상태에서 아래와 같이 실행
	`docker daemon --selinux-enabled`
	
	4) 컨테이너 실행 시 아래와 같은 옵션 사용
	`docker run --interactive --tty --security-opt label=level:TopSecret centos /bin/bash`
	
	- 컨테이너 내에서 리눅스 커널 Capabilities 제한(Container Runtime)
	: 컨테이너 실행 시 사용 환경에 맞게 아래와 같이 기능 추가 및 제거
	`# 필요한 Capabilities 추가`
	`docker run --cap-add={"Capability 1","Capability 2"}`
	
	`# 필요하지 않은 Capabilities 삭제`
	`docker run --cap-drop={"Capability 1","Capability 2"}`
	
	`# 모든 Capabilities 삭제 후 필요한 Capabilities 추가`
	`docker run --cap-drop=all --cap-add={"Capability 1","Capability 2"}`

비고	중기 적용(적용 시 개발자 및 운영자 협의)

1.9. 로그 관리

분류	Docker Configuration	중요도	하
항목명	로그 관리		
항목 설명	Docker의 로그 관리를 통해 개별 사용자, 관리자 또는 시스템의 다른 구성 요소에 의해 시스템에 영향을 미친 활동 순서를 문서화해야 하며, 담당자는 로그 기록을 정기적으로 확인/감독하여 사용자 접속과 관련하여 오류 및 부정행위가 발생하거나 예상되는 경우 즉각적인 보고 조치가 되도록 해야 합니다. 또한 로그 파일이 위·변조되지 않도록 하기 위해 별도 저장 장치에 백업 보관하고, 쓰기 권한을 제한하여 보관하는 것이 바람직하며, 그 외 수정이 가능하더라도 위·변조 여부를 확인할 수 있는 정보(HMAC 값 또는 전자서명 값) 등을 이용한 별도의 보호조치를 취할 수 있습니다.		

진단 방법	**[진단예시]**
	- 로그 레벨 설정
	1) 아래 명령어 실행
	`ps -ef | grep docker`
	
	2) 파라미터 존재 여부 및 적절한 인자 값 설정 여부 확인
	`--log-level`
	
	[그림 18] 설정된 로그 레벨 확인
	
	- 중앙 집중식 원격 로깅 구성
	1) 아래 명령어 실행
	`docker info --format '{{ .LoggingDriver }}'`
	or
	`ps -ef | grep dockerd`
	
	2) 파라미터 존재 여부 및 적절한 인자 값 설정 여부 확인
	`--log-driver`
	
	[그림 19] 원격 로깅 구성 여부 확인		

설정 방법	**- 로그 레벨 설정 (Docker daemon)** : 아래의 명령어를 통한 Docker 서비스 실행 `dockerd --log-level="info"` ※ 로그 상위 debug 레벨은 로그 내용에 중요정보 등이 존재할 가능성이 있으므로 로그 기록을 사용한다면 info 레벨로 사용하는 것을 권고		

	- 중앙 집중식 원격 로깅 구성 (Docker daemon) : 아래의 명령어를 통한 Docker 서비스 실행 `dockerd --log-driver=syslog --log-opt syslog-address=tcp://x.x.x.x`
비고	단기 적용(적용 시 개발자 및 운영자 협의)

2. 호스트 OS

2.1. 설정 파일 및 주요 디렉터리 권한 설정

분류	호스트 OS	중요도	하
항목명	설정 파일 및 주요 디렉터리 권한 설정		
항목 설명	Docker 설정 파일의 퍼미션이 잘못 설정된 경우 비인가자가 다양한 방법으로 Docker 설정을 변경하여 침해사고를 일으킬 가능성이 존재하기 때문에 root 사용자/그룹이 소유권을 가지고 있도록 파일의 권한을 제한하여 파일의 무결성을 유지해야 합니다. 또한 Docker는 SSL/TLS 구성을 통한 네트워크상 DATA 보호 및 사용자 인증을 위해 많은 인증서를 사용합니다. 해당 인증서와 인증서가 포함된 디렉터리가 root 사용자/그룹이 소유권을 가지고 있도록 파일의 권한을 제한하여 파일의 무결성을 유지해야 합니다.		

진단 방법	**[진단예시]**
	- 설정 파일 및 디렉터리 권한 설정
	: 파일 확인 후, 소유권 및 그룹 권한 확인
	`# docker.service`
	`systemctl show -p FragmentPath docker.service //파일 위치 확인`
	`stat -c %a:%U:%G /usr/lib/systemd/system/docker.service //파일 소유 및 권한 확인`
	
	`# docker.socket(1)`
	`systemctl show -p FragmentPath docker.socket //파일 위치 확인`
	`stat -c %a:%U:%G /usr/lib/systemd/system/docker.socket //파일 소유 및 권한 확인`
	
	`# docker.socket(2)`
	`stat -c %a:%U:%G /var/run/docker.sock`
	
	`# daemon.json`
	`stat -c %a:%U:%G /etc/docker/daemon.json`
	
	`# /etc/docker`
	`stat -c %a:%U:%G /etc/docker`
	
	`# /etc/default/docker`
	`stat -c %a:%U:%G /etc/default/docker`
	
	`[root@k8s-master ~]# stat -c %a:%U:%G /usr/lib/systemd/system/docker.service`
	`777:skinfosec:skinfosec`
	`[root@k8s-master ~]#`
	[그림 20] 설정 파일 권한 확인
	
	- 인증서 파일 권한 설정
	: 해당 파일 소유권 및 그룹 권한 확인
	`# 레지스트리 인증서 (일반적으로 /etc/docker/certs.d/ <registry-name> 디렉터리 아래)`
	`stat -c %a:%U:%G /etc/docker/certs.d/*`		

	# TLS CA 인증서 stat -c %a:%U:%G 〈TLS ca 인증서 파일〉 # Docker 서버 인증서 stat -c %a:%U:%G 〈Docker Server 인증서 파일〉 # Docker 서버 인증서 키 파일 stat -c %a:%U:%G 〈Docker Server 인증서 키 파일〉
설정 방법	**- 설정 파일 및 디렉터리 권한 설정** : 사용자 및 그룹 권한 적용 # docker.service chown root:root /usr/lib/systemd/system/docker.service chmod 644 /usr/lib/systemd/system/docker.service # docker.socket(1) chown root:root /usr/lib/systemd/system/docker.socket chmod 644 /usr/lib/systemd/system/docker.socket # docker.socket(2) chown root:docker /var/run/docker.sock chmod 660 /var/run/docker.sock # daemon.json chown root:root /etc/docker/daemon.json chmod 644 /etc/docker/daemon.json # /etc/docker chown root:root /etc/docker chmod 755 /etc/docker # /etc/default/docke chown root:root /etc/default/docker chmod 644 /etc/default/docker **- 인증서 파일 권한 설정** : 사용자 및 그룹 권한 적용 # 레지스트리 인증서 (일반적으로 /etc/docker/certs.d/ 〈registry-name〉 디렉터리 아래) chown root:root /etc/docker/certs.d/〈registry-name〉/* chmod 444 /etc/docker/certs.d/〈registry-name〉/* # TLS CA 인증서 chown root:root 〈TLS ca 인증서 파일〉 chmod 444 〈TLS ca 인증서 파일〉 # Docker 서버 인증서

	chown root:root 〈Docker Server 인증서 파일〉 chmod 444 〈Docker Server 인증서 파일〉 # Docker 서버 인증서 키 파일 chown root:root 〈Docker Server 인증서 키 파일〉 chmod 400 〈Docker Server 인증서 키 파일〉
비고	중기 적용(적용 시 개발자 및 운영자 협의)

2.2. audit 설정

분류	호스트 OS	중요도	하
항목명	audit 설정		
항목 설명	호스트에서 일반적인 Linux 파일 시스템 및 시스템 호출의 Audit 로그 외에도 모든 Docker 관련 파일 및 디렉터리의 Audit 로그 관리를 통해 개별 사용자, 관리자 또는 시스템의 다른 구성 요소에 의해 시스템에 영향을 미친 활동 순서를 문서화합니다. 담당자는 로그 기록을 정기적으로 확인/감독하여 사용자 접속과 관련하여 오류 및 부정행위가 발생하거나 예상되는 경우 즉각적인 보고 조치가 되도록 해야 합니다. 또한 로그 파일이 위·변조되지 않도록 하기 위해 별도 저장 장치에 백업 보관하고, 쓰기 권한을 제한하여 보관하는 것이 바람직하며, 그 외 수정이 가능하더라도 위·변조 여부를 확인할 수 있는 정보(HMAC 값 또는 전자서명 값) 등을 이용한 별도의 보호조치를 취할 수 있습니다.		
진단 방법	**[진단예시]** **- docker 관련 주요 디렉터리 감사(audit)** : 아래의 명령을 실행하여 감사 여부 확인 auditctl -l | grep /usr/bin/docker auditctl -l | grep /var/lib/docker auditctl -l | grep /etc/docker auditctl -l | grep /etc/default/docker auditctl -l | grep /etc/docker/daemon.json auditctl -l | grep /usr/bin/docker-containerd auditctl -l | grep /usr/bin/docker-runc **- docker 관련 주요 파일 감사(audit)** : 아래의 명령을 실행하여 감사 여부 확인 # docker.service systemctl show -p FragmentPath docker.service //파일 위치 확인 auditctl -l | grep docker.service //감사 여부 확인 # docker.socket systemctl show -p FragmentPath docker.socket //파일 위치 확인 auditctl -l | grep docker.socket //감사 여부 확인		
설정 방법	**- docker 관련 주요 디렉터리 감사(audit)** 1) /etc/audit/rules.d/audit.rules 파일 내 아래와 같이 설정 -w /usr/bin/docker -k docker 추가 -w /var/lib/docker -k docker 추가 -w /etc/docker -k docker 추가 -w /etc/default/docker -k docker 추가 -w /etc/docker/daemon.json -k docker 추가 -w /usr/bin/docker-containerd -k docker 추가 -w /usr/bin/docker-runc -k docker 추가 2) audit 데몬 재시작		

service auditd restart

- docker 관련 주요 파일 감사(audit)

1) /etc/audit/rules.d/audit.rules 파일 내 아래와 같이 설정

```
# docker.service 파일이 존재할 경우
-w /usr/lib/systemd/system/docker.service -k docker 추가

# docker.socket 파일이 존재할 경우
-w /usr/lib/systemd/system/docker.socket -k docker 추가
```

2) audit 데몬 재시작

```
service auditd restart
```

```
[root@k8s-master ~]# cat /etc/audit/rules.d/audit.rules
## First rule - delete all
-D

## Increase the buffers to survive stress events.
## Make this bigger for busy systems
-b 8192

## Set failure mode to syslog
-f 1

-w /usr/bin/docker -k docker
-w /var/lib/docker -k docker
-w /etc/docker -k docker
-w /usr/lib/systemd/system/docker.service -k docker
-w /etc/default/docker -k docker
-w /etc/docker/daemon.json -k docker
-w /usr/bin/docker-containerd -k docker
-w /usr/bin/docker-runc -k docker
[root@k8s-master ~]#
```

[그림 21] Docker 관련 주요 파일 audit.rules 파일에 설정

비고	중기 적용(적용 시 개발자 및 운영자 협의)

3. 이미지

3.1. Dockerfile Config

<table>
<tr><td>분류</td><td>이미지</td><td>중요도</td><td>중</td></tr>
<tr><td>항목명</td><td colspan="3">Dockerfile Config</td></tr>
<tr><td>항목
설명</td><td colspan="3">Dockerfile 은 새 컨테이너 이미지를 만드는 데 필요한 모든 명령을 순서대로 포함하는 텍스트 파일로 Docker 는 Dockerfile 에 작성된 명령에 따라 자동으로 이미지를 작성합니다.

Dockerfile 내 명령에는 기반으로 사용할 기존 이미지의 ID, 이미지 만들기 프로세스 중 실행할 명령 및 컨테이너 이미지의 새 인스턴스가 배포될 때 실행될 명령이 포함됩니다.

■ Dockerfile 커맨드
<table>
<tr><th>분류</th><th>설명</th></tr>
<tr><td>FROM</td><td>어떤 이미지를 기반으로 새로운 이미지를 생성할 것인지 지정</td></tr>
<tr><td>RUN</td><td>SHELL 명령을 해당 docker 이미지에 실행시킬 때 사용</td></tr>
<tr><td>EXPOSE</td><td>외부에 노출할 포트를 지정할 때 사용</td></tr>
<tr><td>ENV</td><td>환경 변수를 지정할 때 사용</td></tr>
<tr><td>CMD</td><td>컨테이너가 시작 시 실행할 명령을 지정할 때 사용</td></tr>
<tr><td>ENTRYPOINT</td><td>이미지가 실행될 때 실행되어야 할 데몬을 지정할 때 사용</td></tr>
<tr><td>WORKDIR</td><td>Working 디렉터리를 지정할 때 사용</td></tr>
<tr><td>USER</td><td>이미지를 실행할 user 를 지정할 때 사용</td></tr>
<tr><td>VOLUME</td><td>호스트의 디렉터리를 컨테이너에 연결할 때 사용</td></tr>
<tr><td>ADD</td><td>파일과 디렉터리를 호스트에서 이미지로 copy 할 때 사용</td></tr>
<tr><td>COPY</td><td>ADD 와 기본적으로 동일하나 차이점은 URL 을 지정할 수 없고 압축 파일을 자동으로 풀어주지 않음</td></tr>
<tr><td>LABEL</td><td>LABEL 을 생성</td></tr>
<tr><td>ARG</td><td>이미지 빌드 시 설정할 수 있는 옵션들을 지정할 때 사용</td></tr>
<tr><td>SHELL</td><td>디폴트로 지정되어 있는 SHELL 타입을 변경</td></tr>
</table>
Dockerfile 작성 시 이미지 내 과도한 권한이 부여되지 않도록 설정하여 컨테이너 내 OS 및 응용프로그램을 보호하고 호스트와 컨테이너 간에서 컨테이너 권한 상승으로 인해 호스트가 손상되는 것을 방지해야 합니다.

또한 Dockerfile 은 일반적으로 쉽게 오픈되어 사용되는 경우가 많으므로 주요 정보가 포함될 경우 정보 노출의 가능성이 존재합니다.</td></tr>
<tr><td>진단
방법</td><td colspan="3">[진단예시]
- 컨테이너 사용자 지정
: 아래 명령을 실행 후 공백 여부 확인(공백이면 root로 실행)
<code>docker ps --quiet --all | xargs docker inspect --format '{{.Id}}: User={{.Config.User}}'</code></td></tr>
</table>

```
exit
[root@k8s-master docker2]# docker ps --quiet --all | xargs docker inspect --format '{{ .Id }}: User={{ .Config.User }}'
afb579804da6237045899d234d7fed760b18a53be261c54a3fe84ad72939b011: User=docker_test
00424289635f796d757d7d5e4b2e8e5580dce044d080cc52372d19d7ca762356: User=0
7e7b348bac7758777501f0450f508153dd02ab9f0bcf8379e895a5f53efe0312: User=0
2c96233d5066852ce9cd918111ab727c5196469476a17427c834edc5292b4635: User=
3a6e004e250e934778cf7a321bf958153b20f3912b3d845355015f07164f8847: User=
27141e9b33612b7bfbc52c45fada5107fa3a112028439b4c66ec64986d488e23: User=docker_test
75cba2247781be656b6ab859c21da66abe74fbf24ce9f70c944b3b3cc8712f4c: User=
cc6c67a63887cc0227292ab4cd3e4503116545858002 1d50fac25dab75886646: User=
93f4a237e0fb49419e57ae9cdfde7b39d18c0fc9a8650b76635b32d97de91d66: User=
786a8f5e4364cc7e7da789ba77252e7c435f7d8ef631cd424609ef6694811344: User=0
edd2247f26026f33a88d918a0f78f241d26c0b5586cb1e3c2a2005b214f0573e: User=
94c096790f3ca10b2f7808cb7208bb525f6ac8b8221f786ebf5d917c6d65cc25: User=0
1038702409a16709fef34f2354d799695e774d963f7a950cc8919a6036d90a68: User=0
179e4b93c4881f522407ca022eb265299d7860532c86107521b81fa48b4b7076: User=
ceb48e515d48f8768eee2e381ff6b06c79afa87b6c666b0d5f55c0ee71b04f47: User=
f1559987c682a2186c1c316673e4283b87a6fe402022e91ec305de0a56cea31c: User=
0ec1a2fdca0360495ce18f0a0c5bd32c3c4c7c278bf8e4472e766cb9d2622d38: User=
3946b189f3868807156a3c8df3665a6a830307169c386c2ae6f682f203173449: User=
```

[그림 22] 컨테이너 별 사용자 지정 여부 확인

- Dockerfile 내 secrets 존재 여부 확인

1) 아래 명령을 실행 후 이미지 리스트 확인

```
docker images
```

2) 아래 명령을 실행하여 secret(패스워드, 키 등) 존재 여부 확인

```
docker history <Image_ID>
```

- setuid 및 setgid 권한 제거

: 아래 명령을 실행 후 SUID, SGID 확인

```
docker run <Image_ID> find / -perm +6000 -type f -exec ls -ld {} ₩; 2> /dev/null
```

```
[root@k8s-master docker2]# docker run d23bdf5b1b1b find / -perm +6000 -type f -exec ls -ld {} \; 2> /dev/null
-rwsr-xr-. 1 root messagebus 294512 Nov 22  2016 /usr/lib/dbus-1.0/dbus-daemon-launch-helper
-rwsr-xr-x. 1 root root 464904 Jul 22  2016 /usr/lib/openssh/ssh-keysign
-rwxr-sr-x. 1 root shadow 62272 Nov 18  2015 /usr/bin/chage
-rwxr-sr-x. 1 root tty 27232 Mar 29  2015 /usr/bin/wall
-rwsr-xr-x. 1 root root 75376 Nov 18  2015 /usr/bin/gpasswd
-rwsr-xr-x. 1 root root 53616 Nov 18  2015 /usr/bin/chfn
-rwsr-xr-x. 1 root root 44464 Nov 18  2015 /usr/bin/chsh
-rwsr-xr-x. 1 root root 39912 Nov 18  2015 /usr/bin/newgrp
-rwxr-sr-x. 1 root shadow 22744 Nov 18  2015 /usr/bin/expiry
-rwsr-xr-x. 1 root root 54192 Nov 18  2015 /usr/bin/passwd
-rwxr-sr-x. 1 root ssh 350232 Jul 22  2016 /usr/bin/ssh-agent
-rwxr-sr-x. 1 root shadow 35408 Nov 12  2016 /sbin/unix_chkpwd
-rwsr-xr-x. 1 root root 40168 Nov 18  2015 /bin/su
-rwsr-xr-x. 1 root root 27416 Mar 29  2015 /bin/umount
-rwsr-xr-x. 1 root root 61392 Oct 28  2014 /bin/ping6
-rwsr-xr-x. 1 root root 40000 Mar 29  2015 /bin/mount
-rwsr-xr-x. 1 root root 70576 Oct 28  2014 /bin/ping
```

[그림 23] 컨테이너 권한 확인

- ADD 대신 COPY 사용

1) 아래 명령을 실행 후 이미지 리스트 확인

```
docker images
```

2) 아래 명령을 실행하여 ADD 명령어 사용 여부 확인

```
docker history <Image_ID>
```

```
[root@k8s-master docker2]# docker history 47bb9dd99916
IMAGE               CREATED             CREATED BY                                      SIZE                COMMENT
47bb9dd99916        21 months ago       /bin/sh -c #(nop)  CMD ["/usr/local/bin/et...   0 B
<missing>           21 months ago       /bin/sh -c #(nop)  EXPOSE 2379/tcp 2380/tcp     0 B
<missing>           21 months ago       /bin/sh -c echo 'hosts: files mdns4_minima...   55 B
<missing>           21 months ago       /bin/sh -c mkdir -p /var/lib/etcd/              0 B
<missing>           21 months ago       /bin/sh -c mkdir -p /var/etcd/                  0 B
<missing>           21 months ago       /bin/sh -c #(nop) ADD file:ffcad48a93b5ad0...   14.3 MB
<missing>           21 months ago       /bin/sh -c #(nop) ADD file:7028271b64a6e66...   16.3 MB
<missing>           23 months ago       /bin/sh -c #(nop)  CMD ["/bin/sh"]              0 B
<missing>           23 months ago       /bin/sh -c #(nop) ADD file:63f63606d6e289e...   3.99 MB
[root@k8s-master docker2]#
```

[그림 24] ADD 명령어 사용 확인

- Dockerfile을 통한 업데이트 금지

1) 아래 명령을 실행 후 이미지 리스트 확인

```
docker images
```

2) 아래 명령을 실행하여 apt-get update와 같은 명령어 존재 여부 확인

```
docker history <Image_ID>
```

설정 방법

- 컨테이너 사용자 지정

: dockerfile 생성 시 아래 명령어 추가하여 사용자 지정

```
RUN useradd -d /home/username -m -s /bin/bash username
USER username
```

```
#
# https://github.com/dockerfile/python
#

# Pull base image.
#FROM dockerfile/ubuntu
FROM ubuntu:16.04

# Install Python.
RUN \
  apt-get update && \
#  apt-get install -y python python-dev python-pip python-virtualenv && \
  rm -rf /var/lib/apt/lists/*

RUN useradd -d /home/docker_test -m -s /bin/bash docker_test
USER docker_test


# Define working directory.
WORKDIR /data

# Define default command.
CMD ["bash"]
```

[그림 25] dockerfile 파일 내 사용자 지정

- Dockerfile 내 secrets 존재 여부 확인

: dockerfile 내 secret(패스워드, 키 등) 삭제

- setuid 및 setgid 권한 제거

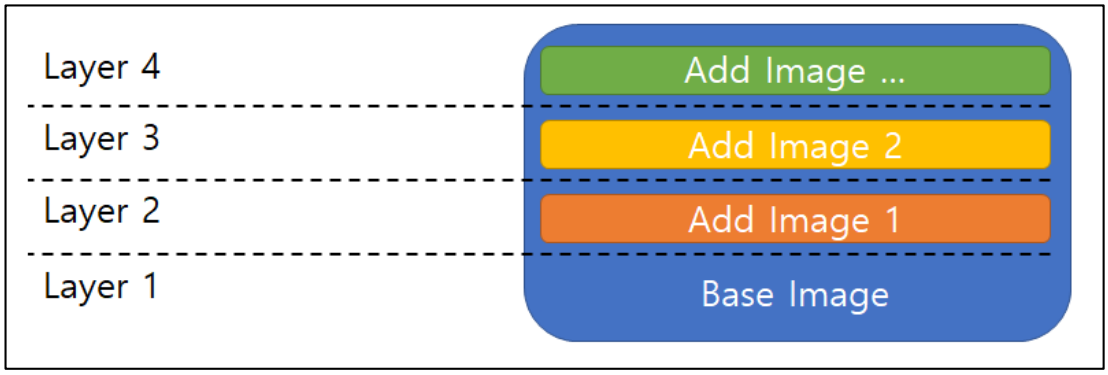
: dockerfile 생성 시 아래 명령어 추가하여 권한 제거

```
docker run <Image_ID> find / -perm +6000 -type f -exec ls -ld {} ₩; 2> /dev/null
```

※ 서비스상 필요할 경우 예외처리 요청

	- ADD 대신 COPY 사용 : dockerfile 내 ADD대신 COPY 명령어 사용 ※ 서비스상 필요할 경우 예외처리 요청 **- Dockerfile을 통한 업데이트 금지** : dockerfile 내 업데이트 명령어 삭제 ※ 서비스상 필요할 경우 예외처리 요청
비고	중기 적용(적용 시 개발자 및 운영자 협의)

3.2. 이미지 취약점 및 구성 결함

분류	이미지	중요도	중
항목명	이미지 취약점 및 구성 결함		
항목 설명	Docker 에서 이미지는 컨테이너를 생성하기 위한 기초로 컨테이너가 동작하기 위한 모든 구성요소가 포함되어 있는 파일입니다. 따라서 보안상 불안전한 이미지를 생성하여 사용할 경우 이를 기초로 실행되는 컨테이너 및 컨테이너가 실행되는 호스트에 영향을 끼칠 수 있습니다.  [그림 26] Docker 이미지의 Layer 구조 **■ 신뢰할 수 없는 이미지 사용** 외부에서 제 3 자가 제공하는 이미지에 대한 유효성 및 보안성 검토 없이 사용하여 컨테이너를 생성할 경우 악성코드 감염, 데이터 누출 등과 같은 유형의 위험이 초래되거나 취약점이 있는 구성 요소가 포함되어 2 차 공격에 활용될 가능성이 있습니다. **■ 취약한 이미지 구성요소** 이미지에는 컨테이너를 실행하는데 필요한 모든 구성요소가 포함되어 있는데, 이러한 이미지의 구성요소에 SSH 와 같은 위험에 노출될 가능성이 큰 패키지가 존재하거나, 치명적인 보안 업데이트가 누락 또는 오래된 버전의 구성요소가 포함될 수 있습니다. 이러한 이미지로 컨테이너가 실행될 경우 구성요소의 취약점을 활용한 2 차 공격에 활용될 가능성이 존재합니다. **■ 이미지 내 주요 정보 직접 포함** 컨테이너에서 서비스를 구동하여 운영 시 환경에 따라 secrets, DB 연결을 위한 ID 와 암호, X.509 개인키 등과 같은 주요 정보를 사용해야 하는 경우가 존재합니다. 하지만 이미지 내 이러한 정보가 포함되어 있을 경우 해당 이미지에 접근 가능한 누구나 획득할 수 있으므로 정보 노출의 가능성이 존재합니다.		

진단 방법	**[진단예시]**
	- 호스트 내 이미지 검증
	1) 아래 명령을 실행 후 호스트에서 현재 사용할 수 있는 모든 컨테이너 이미지 조회
	`docker images`
	
	2) 위 과정을 통해 조회된 이미지의 생성 기록을 통해 생성 시 기본 이미지 및 이미지 내 설치된 패키지를 신뢰할 수 있는지 확인
	`docker history <imageName>`
	
	- 컨테이너 내 패키지 검증		

	1) 아래 명령을 통해 실행 중인 컨테이너의 인스턴스 조회
	`docker ps - quiet`
	
	2) 위 과정을 통해 조회된 컨테이너 인스턴스 내 설치된 패키지 확인 후 불필요한 패키지가 존재하는지 및 보안 패치가 되어 있지 않은 패키지가 존재하는지 확인
	`docker exec $INSTANCE_ID rpm -qa`

설정 방법	**- 신뢰할 수 없는 기본 이미지 사용 및 패키지 사용** : Docker Content trust를 구성하고, 주기적인 이미지 검사를 통해 사용하는 이미지의 유효성 및 보안성 검토 ※ 서비스상 필요할 경우 예외처리 요청 **- 컨테이너 내 패키지 검증** : 사용하지 않는 패키지가 있는 경우 삭제해야 하며, 패키지 업데이트가 필요한 경우 컨테이너 내에서 직접 업데이트 하지 말고 보안 패치가 적용된 패키지를 이용하여 이미지를 재생성하여 사용할 것을 권고 ※ 서비스상 필요할 경우 예외처리 요청
비고	중기 적용(적용 시 개발자 및 운영자 협의)

3.3. 레지스트리 운영 관리

분류	이미지	중요도	중
항목명	레지스트리 운영 관리		
항목 설명	레지스트리는 개발자가 이미지를 쉽게 저장할 수 있도록 해주는 서비스로, 식별 및 버전 관리를 위해 이미지에 태그를 지정하고 카탈로그화하여 검색 및 재사용을 지원하고 다른 사람들이 만든 이미지를 찾아서 다운로드할 수 있는 기능을 제공합니다. 레지스트리는 자체 호스팅 되거나 별도 외부 서비스를 활용하여 구성할 수 있으며, 레지스트리의 예로 Amazon EC2 Container Registry, Docker Hub, Docker Trusted Registry 및 Quay Container Registry 등이 있습니다. 레지스트리에 저장되는 이미지 내에는 독점 소프트웨어나 계정정보 등과 같은 민감한 구성요소 또는 정보가 포함될 가능성이 존재하고 이미지를 활용한 컨테이너 실행의 가능성이 존재하므로 레지스트리 내 이미지에 대한 보안성 보장을 위한 대책을 마련하여야 합니다. **■ 레지스트리와 암호화된 연결 설정** 레지스트리와 통신 시 SSL/TLS 프로토콜 적용을 통해 네트워크상에서 독점 소프트웨어나 계정정보 등과 같은 민감한 구성요소 또는 정보를 스니핑과 같은 방법을 통해 노출되지 않도록 보호해야 합니다. 또, TLS 인증서를 활용하여 Docker 와 레지스트리 간 사용자 검증을 할 수 있도록 설정해야 합니다. **■ 레지스트리 내 인증 및 권한 제한** 레지스트리에는 독점 소프트웨어나 계정정보 등과 같은 민감한 구성요소가 포함된 이미지가 존재할 수 있으므로 인증 및 권한 부여 제어가 제대로 이루어지지 않으면 이미지 내 주요 정보가 노출될 수 있습니다. 취약한 이미지의 업로드를 통해 이를 활용하여 구성한 컨테이너 및 호스트가 손상될 수 있습니다. **■ 레지스트리 내 이미지 관리** 일반적으로 레지스트리는 조직에서 배포하는 모든 이미지의 원본 위치이므로 시간이 지남에 따라 레지스트리에 저장된 이미지 중 취약하거나 오래된 버전이 존재할 수 있습니다. 이러한 취약한 이미지는 단순히 레지스트리에 저장되는 것으로는 직접적인 위협이 발생하지는 않지만 의도치 않은 상황에서 취약점이 있는 이미지가 배포될 가능성이 있습니다.		
진단 방법	**[진단예시]** **- 원격 레지스트리와 주고받는 데이터에 Content trust 적용(Docker Hub만 적용 가능)** : 아래 명령을 실행 후 실행하여 Content trust 활성화 여부 확인 `echo $DOCKER_CONTENT_TRUST` [그림 27] Content trust 활성화		

```
[root@k8s-master docker2]# echo $DOCKER_CONTENT_TRUST

[root@k8s-master docker2]# export DOCKER_CONTENT_TRUST=1
[root@k8s-master docker2]#
[root@k8s-master docker2]#
[root@k8s-master docker2]# echo $DOCKER_CONTENT_TRUST
1
[root@k8s-master docker2]#
```

	- 레지스트리와 암호화되지 않은 연결 금지 : 아래 명령을 실행 후 사용하고 있는 레지스트리 및 --insecure-registry 옵션 적용 여부 확인 `ps -ef | grep dockerd` **- 구버전(v1) legacy registry 사용 금지** : 아래 명령을 실행 후 --disable-legacy-registry 값 확인 `ps -ef | grep dockerd` ※ v17.12 에서 더 이상 지원되지 않음

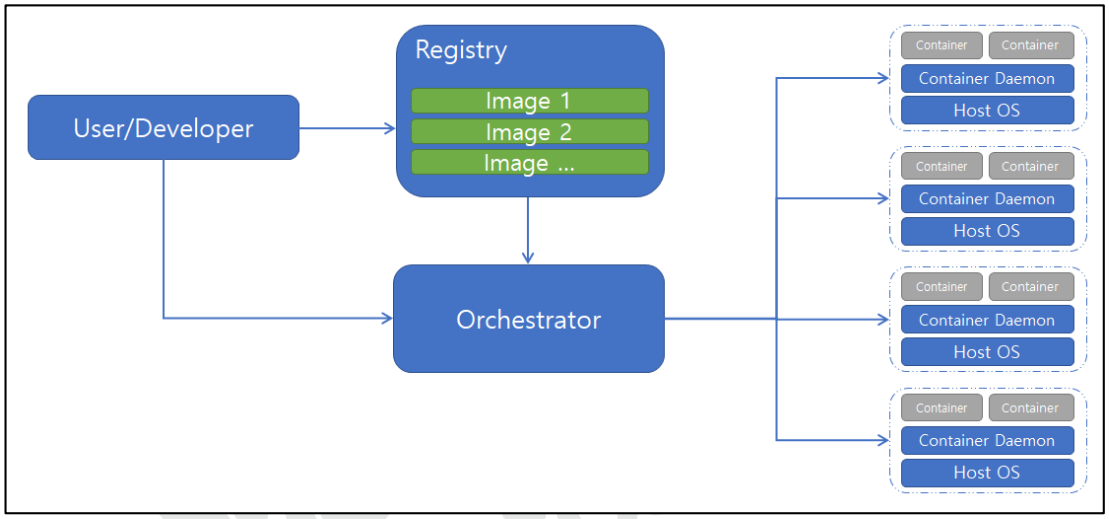
설정 방법	**- 원격 레지스트리와 주고 받는 데이터에 Content trust 적용(Docker Hub만 적용 가능)**
	: 아래 명령을 실행을 통해 Content trust 활성화
	`export DOCKER_CONTENT_TRUST=1`
	
	- 레지스트리와 암호화 되지 않은 연결 금지(Docker daemon)
	: Docker 서비스 실행 시 아래의 옵션 사용 금지
	`--insecure-registry`
	
	- 구 버전(v1) legacy registry 사용 금지(Docker daemon)
	: Docker 서비스 실행 시 아래의 명령어를 사용하여 구 버전 레지스트리 비 활성화
	`dockerd --disable-legacy-registry`

비고	중기 적용(적용 시 개발자 및 운영자 협의)

4. Docker Swarm

4.1. 인증 제어

분류	Docker Swarm	중요도	상
항목명	인증 제어		
항목 설명	컨테이너에서 오케스트레이터는 개발자로부터 이미지를 가져와서 이미지를 컨테이너에 배포하고 실행 중인 컨테이너를 관리하는 역할을 합니다.		

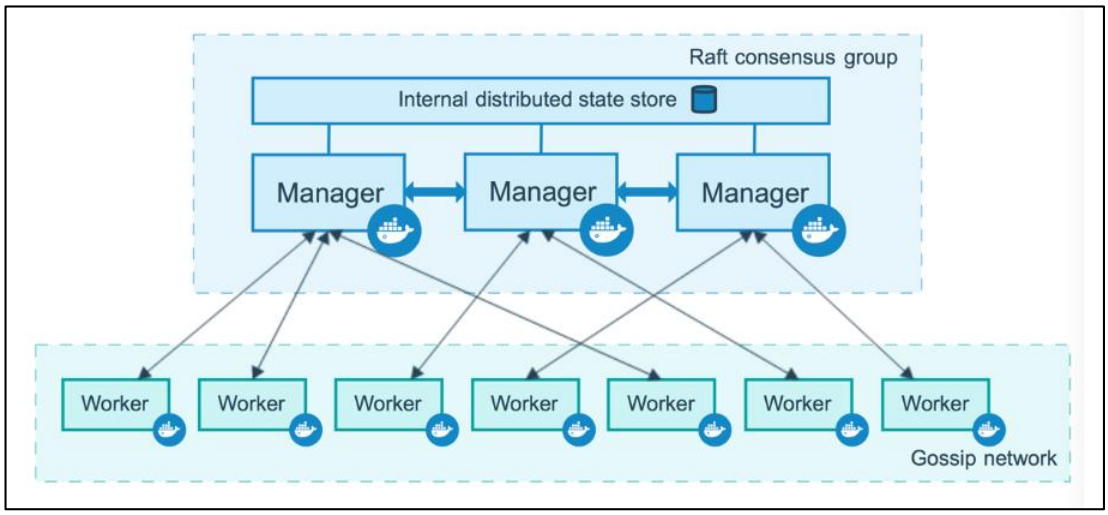


[그림 28] 컨테이너에서의 오케스트레이터

Docker 에서는 Docker Engine CLI 를 사용하여 컨테이너를 배포관리 할 수 있는 swarm mode 라는 오케스트레이터를 자체적으로 지원하고 있어 추가적인 오케스트레이터 소프트웨어 없이 클러스터 생성 및 관리가 가능합니다.

swarm mode 에서는 manager node 와 worker node 가 있으며, manager node 의 경우 swarm 명령을 통해 worker node 에게 작업 지시를 할 수 있으며 동시에 worker node 와 같이 컨테이너 생성이 가능하나 가급적 manager node 에서는 클러스터 상태 관리만 하도록 구성하는 것을 권고드립니다.

worker node 는 manager node 로부터 작업을 수신받아 컨테이너를 생성하고 상태를 체크하는 등의 역할을 수행합니다.



[그림 29] swarm mod 구조

취약한 설정 및 운영으로 인해 잘못된 인증이 구현될 경우 Docker 시스템 내 컨테이너, 호스트 등의 모든 요소에 영향을 줄 수 있으므로 manager node 인증에 대해 swarm mode 불필요한 활성화 금지, 잠금 모드 사용 등과 같은 방법을 통해 엄격하게 제어되어야 합니다.

진단 방법

[진단예시]

- swarm mode 불필요하게 활성화 금지

: 아래의 명령을 실행하여 swarm mode 동작 여부 확인
(동작 시= Swarm: active)

```
docker info
```

- 관리자 노드 최소화

: 아래의 명령을 실행하여 관리자 노드 확인

```
docker info --format '{{ .Swarm.Managers }}'
```

- 자동 잠금 모드 사용

: 아래의 명령을 실행하여 자동 잠금 모드 사용 여부 확인

```
docker swarm unlock-key
```

설정 방법

```
[osboxes@master ~]$ sudo docker swarm unlock-key
no unlock key is set
[osboxes@master ~]$
```

[그림 30] 자동 잠금 모드 설정 여부 확인

- swarm mode 불필요하게 활성화 금지

: 불필요하게 swarm mode가 활성화된 경우 아래 명령어를 통한 비활성화

```
docker swarm leave
```

- 관리자 노드 최소화

: 불필요하게 활성화된 관리자 노드가 존재할 경우 아래의 명령을 통해 삭제

```
docker node demote <ID>
```

	- 자동 잠금 모드 사용
	1) 아래 명령어를 통한 자동 잠금 모드 실행
	`# swarm 모드 시작 시 적용`
	`docker swarm init - autolock`
	
	`# swarm 모드 동작 중 적용`
	`docker swarm update --autolock`
	
	2) 아래 명령어를 통해 조직 내 정책에 따라 주기적으로 자동 잠금키 변경
	`docker swarm unlock-key - rotate`

비고	장기 적용(적용 시 개발자 및 운영자 협의)

4.2. SSL/TLS 적용

분류	Docker Swarm	중요도	상
항목명	SSL/TLS 적용		
항목 설명	Swarm mode 에서 node 간 통신 시 SSL/TLS 를 이용하여 네트워크 상의 DATA 보호 및 사용자 검증을 수행하도록 구성할 수 있습니다. SSL/TLS 통신 적용을 통해 네트워크 스니핑과 같은 방법으로 주요 정보가 노출되어 다른 공격에 이용되지 않도록 하고, node 간 접근하는 대상들에 대해 검증할 수 있도록 설정하여야 합니다. 또한 SSL/TLS 통신 적용 시에는 주기적으로 인증서를 변경하고 안전한 버전의 암호화 방식을 사용하는 방법을 통해 위험을 최소화할 수 있는 정책 설정이 필요합니다.		

진단 방법	**[진단예시]**
	- SSL/TLS 적용을 통한 네트워크 구간 데이터 보호 및 사용자 인증
	: 아래의 명령어를 실행하여 SSL/TLS 적용 여부 확인
	`docker network ls --filter driver=overlay --quiet | xargs docker network inspect --format '{{.Name}} {{ .Options }}'`
	
	- 인증서 관리(인증서 교환주기 설정)
	1) 아래의 명령어를 실행하여 노드 인증서 교환주기 확인
	`docker info | grep "Expiry Duration"`
	
	`[osboxes@master ~]$ sudo docker info | grep "Expiry Duration"`
	`  Expiry Duration: 3 months`
	`[osboxes@master ~]$`
	[그림 31] 인증서 만료일 확인
	
	2) 아래의 명령어를 실행하여 CA인증서 교환주기 확인
	`ls -l /var/lib/docker/swarm/certificates/swarm-root-ca.crt`
	
	`[osboxes@master ~]$ sudo ls -l /var/lib/docker/swarm/certificates/swarm-root-ca.crt`
	`[sudo] password for osboxes:`
	`-rw-r--r--. 1 root root 554 Apr 16 05:43 /var/lib/docker/swarm/certificates/swarm-root-ca.crt`
	`[osboxes@master ~]$`
	[그림 32] CA 인증서 만료일 확인		

설정 방법	**- SSL/TLS 적용을 통한 네트워크 구간 데이터 보호 및 사용자 인증**
	: 실행 시 아래의 옵션을 통해 SSL/TLS 적용
	`--opt encrypted`
	
	- 인증서 관리(인증서 교환주기 설정)
	1) 아래 명령어를 통해 조직 내 정책에 따라 노드 인증서 교환 주기 설정
	`docker swarm update --cert-expiry 48h`
	
	2) 아래 명령어를 통해 조직 내 정책에 따라 주기적으로 CA인증서 변경
	`docker swarm ca --rotate`		

비고	중기 적용(적용 시 개발자 및 운영자 협의)

4.3. 네트워크 제어

분류	Docker Swarm	중요도	중
항목명	네트워크 제어		
항목 설명	**- Docker Swarm 네트워크 인터페이스 설정** 만약 호스트 시스템에서 여러 개의 네트워크 인터페이스가 존재하는 경우 기본적으로 모든 인터페이스에서 Docker Swarm에 접근이 가능하여, 보안이 적용되지 않은 경로를 통한 접근 가능성이 존재하므로, 특정 외부 인터페이스에서 들어오는 연결만 허용하도록 설정해야 합니다. **- 네트워크 트래픽 분리** 컨테이너의 데이터 트래픽과 클러스터 관리 트래픽을 분리하여 구성할 경우 각각의 특성에 맞게끔 개별적으로 모니터링될 수 있으며 다양한 트래픽 제어 정책 및 모니터링에 연결하여 관리하기 용이합니다.		
진단 방법	**[진단예시]** **- Docker Swarm 네트워크 인터페이스 설정** : 아래의 명령어를 실행하여 특정 인터페이스에서만 수신 대기 중인지 확인 예시) netstat -lt | grep -i 2377		
설정 방법	**- Docker Swarm 네트워크 인터페이스 설정** : 아래의 명령어를 실행하여 특정 인터페이스에서만 수신하도록 설정 docker swarm init --listen-addr example IP [그림 33] 네트워크 인터페이스 수정		
비고	장기 적용(적용 시 개발자 및 운영자 협의)		

5. 기타

5.1. 중요도 수준에 따른 격리 운영

분류	기타	중요도	N/A
항목명	중요도 수준에 따른 격리 운영		
항목 설명	조직에서 컨테이너를 통해 서비스를 운영하는 경우 때에 따라 외부에 오픈된 웹 서비스, 내부 관리자 서비스와 같이 민감도 수준이 다른 기능을 수행하는 컨테이너들이 존재할 수 있으며, 컨테이너 구축 전 각각 서비스하고자 하는 바를 파악 후 별도 구분하여 네트워크, 호스트에서 각각 격리 운영하는 것을 권고드립니다. **■ 민감도에 따른 컨테이너 네트워크 격리** **■ 민감도에 따른 호스트 격리** **■ 하나의 호스트 내 컨테이너와 비 컨테이너 서비스 동시 사용 금지**		
진단 방법	※ 해당 항목은 체크리스트에 포함하지 않음.		
설정 방법			
비고	N/A		

5.2. 앱 취약점

분류	기타	중요도	N/A
항목명	앱 취약점		
항목 설명	예를 들어, 컨테이너화 된 웹 애플리케이션에서 크로스 사이트 스크립팅, SQL injection 등과 같은 전형적인 취약점이 존재하여 컨테이너가 손상되어 주요 정보가 노출되거나 컨테이너를 통한 호스트 OS 에 대한 공격과 같이 Docker 의 환경 구성에 대한 보안 문제뿐 아니라 컨테이너에서 운영되는 애플리케이션에 대한 보안 문제도 존재합니다. 따라서 컨테이너에서 운영되는 애플리케이션에 대해서 기존의 호스트 기반 어플리케이션에서 수행하는 침입을 탐지하고 공격을 예방하는 프로세스를 적용하여 운영하여야 합니다.		
진단 방법	※ 해당 항목은 체크리스트에 포함하지 않음.		
설정 방법			
비고	N/A		

5.3. 보안 패치 적용

<table>
<tr><td>분류</td><td>기타</td><td>중요도</td><td>중</td></tr>
<tr><td>항목명</td><td colspan="3">보안 패치 적용</td></tr>
<tr><td>항목
설명</td><td colspan="3">Docker 소프트웨어는 보안 취약점, 제품 버그를 해결한 릴리즈를 제공하고 있습니다.

Docker 업데이트를 최신 상태로 유지하면 소프트웨어 취약성을 줄일 수 있습니다.</td></tr>
<tr><td>진단
방법</td><td colspan="3">[진단예시]
- 최신 엔진 업데이트 설치 여부 확인
docker version</td></tr>
<tr><td>설정
방법</td><td colspan="3">1. 기간 산정해서 보안 패치 적용(정기 PM 등)
※ 업데이트로 인하여 서비스 장애 등이 발생할 수 있으므로 내부 검토 후 적용 검토

- Docker 최신 Release 현황
<table><tr><th>버전(Docker Engine EE and CE)</th><th>Release 일자</th><th>비고</th></tr><tr><td>18.09.2</td><td>2019-02-11</td><td>-</td></tr><tr><td>18.03.1-ee-6</td><td>2019-02-11</td><td>-</td></tr><tr><td>17.06.2-ee-19</td><td>2019-02-11</td><td>-</td></tr></table>
[참고]
- CVE-2019-5736: 컨테이너 탈출 취약점(2019 년 2 월 11 일)

2. 아래 사이트를 참고하여, 원격 exploit 취약점, 제로데이 취약점 등 크리티컬한 취약점 발견 시 즉시 패치 권고

※ Docker Security Update Site
https://docs.docker.com/engine/release-notes/</td></tr>
<tr><td>비고</td><td colspan="3">장기 적용(적용 시 개발자 및 운영자 협의)</td></tr>
</table>

Part 02
- Kubernetes -

I. 전체목록

1. 체크리스트 항목

진단에 사용될 체크리스트는 국내외 기술 자료를 바탕으로 작성하였습니다. Kubernetes 보안 가이드에서의 영역은 Master Node(14개 항목), Worker Node(7개 항목), 기타(2개 항목)로 총 3개 영역에서 23개 항목으로 구성하였습니다.

[표] 5. Kubernetes 보안 진단 체크리스트

구분	분류	항목코드	항목명	중요도
Master Node	API Server Configuration	1-1	API server 인증 제어	상
		1-2	API server 권한 제어	상
		1-3	SSL/TLS 적용	상
		1-4	Admission Control Plugin 설정	중
		1-5	로그 관리	하
	etcd Configuration	2-1	SSL/TLS 적용	상
		2-2	etcd 암호화	중
	Controller Manager Configuration	3-1	Controller 인증 제어	중
		3-2	SSL/TLS 적용	중
	PodSecurityPolicy Configuration	4-1	컨테이너 권한 제어	상
		4-2	네임스페이스 관리	중
	Host OS	5-1	설정 파일 권한 설정	하
		5-2	etcd 데이터 디렉터리 권한 설정	하
		5-3	인증서 파일 권한 설정	하
Worker Node	Kubelet Configuration	6-1	Kubelet 인증 제어	상
		6-2	Kubelet 권한 제어	상
		6-3	SSL/TLS 적용	상
		6-4	로그 관리	하
		6-5	Kernel 파라미터 설정	중
	Host OS	7-1	설정 파일 권한 설정	하
		7-2	인증서 파일 권한 설정	하
기타	기타	8-1	네트워크 정책 설정	N/A
		8-2	보안 패치 적용	중

2. 대응 버전

본 보안 가이드 문서를 이용하여 대응 가능한 Kubernetes 버전은 아래 표와 같습니다.

[표] 6. Kubernetes 보안 가이드 대응 버전

버전 대역	상세 버전	비고
Kubernetes	V1.14.1	

※ 해당 가이드 작성을 위한 테스트 환경은 아래와 같습니다.
- Operating System: CentOS Linux 7 (Core)
- Architecture: x86_64

3. 위험도 및 적용 권고 시기 구분

3.1 위험도

각 취약점으로 인해 발생 가능한 피해에 대하여 위험도 산정을 통해 상, 중, 하 3단계로 분류하였습니다.

[표] 7. 위험도 구분

위험도	내 용	비고
상	관리자 계정 및 주요 정보 유출로 인한 치명적인 피해 발생	
중	노출된 정보를 통해 서비스/시스템 관련 추가 정보 유출 발생 우려	
하	타 취약점과 연계 가능한 잠재적인 위협 내재	

3.2 적용 권고 시기

각 취약점 항목의 위험도 및 그에 대한 대응 조치 과정에서 예상되는 서비스/시스템 영향도를 고려하여 단기, 중기, 장기로 적용 권고 시기를 분류하였습니다.

[표] 8. 적용 권고 시기 구분

적용 권고 시기	내 용	비고
단기	연계 서비스/시스템 영향도가 낮으며 빠른 조치가 필요한 경우	
중기(기본)	기본적인 서비스/시스템 영향도 검토가 필요한 경우	
장기	조치로 인한 연계 서비스/시스템 영향이 예상되는 경우	

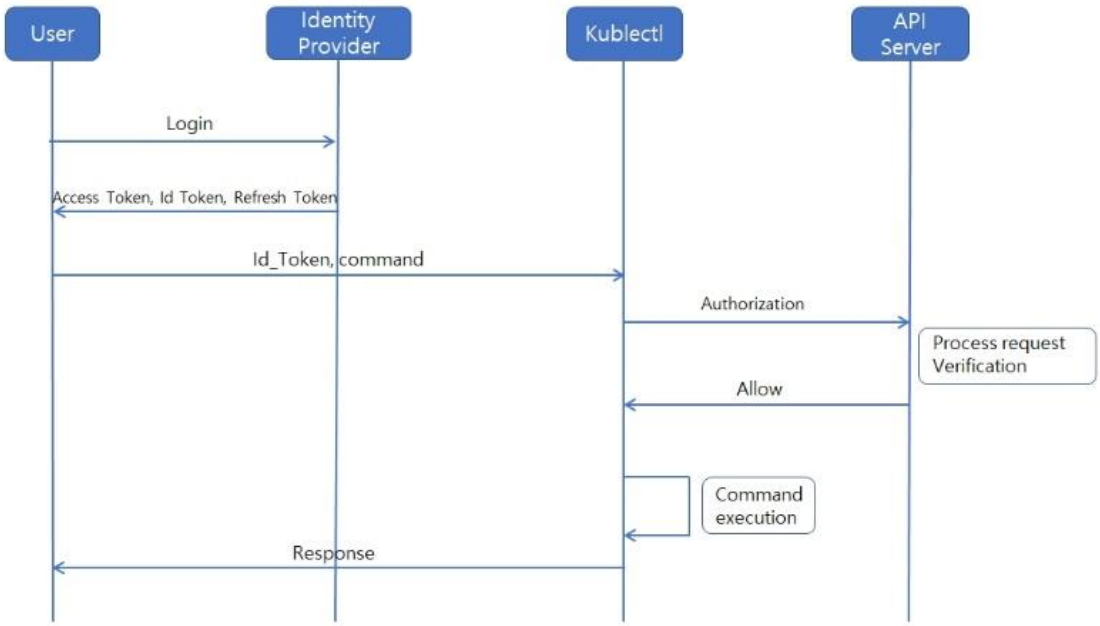
II. 세부항목 설정

1. Master Node - API Server Configuration

1.1. API Server 인증 제어

분류	API Server Configuration	중요도	상
항목명	API Server 인증 제어		
항목 설명	API Server는 Kubernetes 요소들의 허브로 HTTP, HTTPS 기반의 REST API를 제공하여 다른 모든 구성 요소를 상호 작용할 수 있도록 연결하는 역할을 합니다. API Server의 잘못된 인증 구현은 Kubernetes 시스템의 모든 요소에 영향을 줄 수 있으므로 API Server의 외부 독립 서비스를 이용한 사용자 계정 관리, 설정을 통한 비인증 서비스 접근에 대한 차단, 안전한 방식의 인증 시스템 사용 등의 강력한 인증 방법을 사용하여, API Server 인증에 대해 엄격하게 제어되어야 합니다. Kubernetes에서는 사용자 인증을 위해 서비스 계정과 사용자 계정의 두 가지 계정을 사용하고 있습니다. 서비스 계정은 Kubernetes에서 관리하는 계정으로 특정 네임 스페이스에 바인딩 되어 클라이언트가 Kubernetes API를 호출하거나, 콘솔이나 기타 클라이언트가 Kubernetes API에 접근하고자 할 때 사용하며, API 서버에서 자동으로 만들거나 API 호출을 통해 수동으로 만듭니다. 사용자 계정의 경우 자체 인증 서비스를 제공하지 않아 외부 독립 서비스를 사용해야 하지만 구글 계정(Google Account)이나 오픈스택의 키스톤(keystone), LDAP 등의 인증 시스템과 계정 연동 방식을 지원하고 있습니다.		

진단 방법	**[진단예시]**
	- 사용자 계정 관리를 위한 시스템 연동
	: Kubernetes 내 사용자 인증을 위한 프로세스 적용 여부 확인
	
	- 비인증 접근 차단
	: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
	`--anonymous-auth`
	`--insecure-allow-any-token`
	`--insecure-bind-address`
	`--insecure-port`
	`--repair-malformed-updates`
	`--service-account-lookup`
	
	- 취약한 방식의 인증 방식 사용
	: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
	`--basic-auth-file`
	`--token-auth-file`
	
	- 서비스 API 외부 오픈 금지		

	1) Scheduler API 서비스 : /etc/kubernetes/manifests/kube-scheduler.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인 --address 2) Controller Manager API 서비스 : /etc/kubernetes/manifests/kube-controller-manager.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인 --address
설정 방법	**- 사용자 계정 관리를 위한 시스템 연동** : Kubernetes에서는 OAuth나 Webhook과 같은 계정 연동 서비스를 지원하고 있으므로 이를 활용하여, 외부 인증 관리 서비스 연동을 통해 사용자 계정을 관리하는 것을 권고함.  [그림 34] OAuth 를 통한 계정 연동 예시 **- 비인증 접근 차단** : /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래와 같이 설정 --anonymous-auth=false 추가 ※ health check 및 CustomResourceDefinitions와 같은 일반적인 정보 조회가 필요한 경우 검토 후 위 설정은 제외 --insecure-allow-any-token=true 존재할 경우 제거 --insecure-bind-address=X.X.X.X 존재할 경우 제거 --insecure-port=0 (default) --repair-malformed-updates=false 추가 --service-account-lookup=true (default)

- 취약한 방식의 인증 방식 사용

: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래와 같이 설정

--basic-auth-file=<filename> 존재할 경우 제거

--token-auth-file=<filename> 존재할 경우 제거

```
[root@k8s-master osboxes]# cat /etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.10.105
    - --anonymous-auth=false
    - --insecure-bind-address=0.0.0.0
    - --allow-privileged=true
    - --insecure-allow-any-token=true
    - --authorization-mode=Node,RBAC
    - --repair-malformed-updates=false
    - --basic-auth-file=/etc/kubernetes/basic/authentificate.auth
    - --token-auth-file=/etc/kubernetes/token.auth
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
    - --insecure-port=0
    - --service-account-lookup=true
    - --service-account-key-file=/etc/kubernetes/account.key
    - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
```

[그림 35] kube-apiserver.yaml 내 인증 설정 확인

- 서비스 API 외부 오픈 금지

1) Scheduler API 서비스

: /etc/kubernetes/manifests/kube-scheduler.yaml 파일 내 아래와 같이 설정

--address=127.0.0.1 (default)

2) Controller Manager API 서비스

: /etc/kubernetes/manifests/kube-controller-manager.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인

--address=127.0.0.1 (default)

비고	장기 적용(적용 시 개발자 및 운영자 협의)

1.2. API server 권한 제어

<table>
<tr><td>분류</td><td>API Server Configuration</td><td>중요도</td><td>상</td></tr>
<tr><td>항목명</td><td colspan="3">API Server 인증 제어</td></tr>
<tr><td>항목
설명</td><td colspan="3">Kubernetes는 API Server를 통해 정책에 따라 요청에 대한 승인을 수행하고 있습니다.

Kubernetes 특성상 민감도 수준이 다른 컨테이너를 운영하는 다양한 사용자 또는 그룹이 접근하여 사용할 수 있습니다. 이때 필요한 권한 이상으로 사용자 또는 그룹에 부여될 경우 악의적인 사용자나 부주의한 사용자에 의해 Kubernetes에서 관리하는 다른 컨테이너의 작업에 영향을 주거나 파괴할 수 있습니다.

따라서 사용자 또는 그룹에 직무에 필요한 특정 호스트, 컨테이너 및 이미지에서 특정 작업을 수행할 수 있는 최소한의 권한만 부여해야 합니다.

Kubernetes는 ABAC, RBAC, Node 및 Webhook과 같은 여러 인증 모듈을 지원하고 있으며, API Server 내 설정을 통해 권한 모듈을 구성할 수 있습니다.

■ Kubernetes 권한 모듈
<table>
<tr><th>분류</th><th>설명</th></tr>
<tr><td>RBAC</td><td>▷ 역할에 따른 권한 부여 모델
▷ 사용자 그룹 및 그룹 내 역할에 따라 권한 부여</td></tr>
<tr><td>ABAC</td><td>▷ 속성에 따른 권한 부여 모델
▷ 네임스페이스, 그룹, 경로 등을 속성으로 하여 이에 따른 권한 부여</td></tr>
<tr><td>Webhook</td><td>▷ Webhook은 HTTP POST를 통한 이벤트 전달 방식
▷ 해당 모드의 경우 외부 REST서비스를 통한 권한 결정 시 사용</td></tr>
<tr><td>Node</td><td>▷ kubelet에 의해 만들어진 API 요청에 대한 특수 용도의 인증 모드</td></tr>
</table></td></tr>
<tr><td>진단
방법</td><td colspan="3">- 권한 검증 부재
: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
<code>--authorization-mode</code>

- Node 권한 사용
: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
<code>--authorization-mode</code>

- 역할 기반 액세스 제어(RBAC)를 사용
: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
<code>--authorization-mode</code></td></tr>
<tr><td>설정
방법</td><td colspan="3">- 권한 검증 부재
: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래와 같이 설정
<code>--authorization-mode=AlwaysAllow 값 존재할 경우 제거</code></td></tr>
</table>

	- Node 권한 사용
	: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래와 같이 설정
	`--authorization-mode=Node`
	
	- 역할 기반 액세스 제어(RBAC)를 사용
	: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래와 같이 설정
	`--authorization-mode=RBAC`
	
	[그림 36] kube-apiserver.yaml 내 권한 설정 확인

비고	장기 적용(적용 시 개발자 및 운영자 협의)

1.3. SSL/TLS 적용

분류	API Server Configuration	중요도	상
항목명	SSL/TLS 적용		
항목 설명	TLS(Transport Layer Security)는 인터넷에서 정보를 암호화해서 송수신하는 프로토콜로, Netscape Communications 사가 개발한 SSL(Secure Socket Layer)에서 표준화하여, 국제 인터넷 표준화 기구에서 표준으로 인정받은 프로토콜 입니다. Kubernetes에서 API Server와 kubelet, 사용자 간의 통신 시 SSL/TLS 프로토콜을 이용하여 네트워크상에서의 secrets 및 keys와 같은 주요 데이터 보호 및 API Server와 통신하는 클라이언트의 검증을 하는 데 사용할 수 있습니다. SSL/TLS 통신 적용을 통해 네트워크 스니핑과 같은 방법으로 클러스터 내 주요 정보가 노출되어 다른 공격에 이용되지 않도록 하고, API Server가 kubelet 클라이언트를 검증할 수 있도록 설정하여야 합니다. 또한 SSL/TLS 통신 적용 시에는 주기적으로 인증서를 변경하고 안전한 버전의 암호화 방식을 사용하는 방법을 통해 위험을 최소화할 수 있는 정책 설정이 필요합니다.		

진단 방법	**[진단예시]**
	- SSL/TLS 적용을 통한 네트워크 구간 데이터 보호
	: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
	`--kubelet-https 미설정 or true 설정`
	`--secure-port 미설정 or 1~65535로 설정`
	
	- 인증서 관리(apiserver to kubelet) (인증서설정)
	: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
	`--kubelet-certificate-authority`
	`--kubelet-client-certificate`
	`--kubelet-client-key`
	`--service-account-key-file`
	
	- 인증서 관리(apiserver) (인증서설정)
	: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
	`--tls-cert-file`
	`--tls-private-key-file`
	`--client-ca-file`
	
	- 안전한 SSL/TLS 버전 사용
	: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
	`--tls-cipher-suites`		

설정	**- SSL/TLS 적용을 통한 네트워크 구간 데이터 보호**		

방법	: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래와 같이 설정 --kubelet-https 제거 --secure-port 제거 or 0 아닌 값으로 설정(default 6443) **- 인증서 관리(apiserver to kubelet) (인증서설정)** : /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래와 같이 설정 --kubelet-certificate-authority=〈인증서 파일〉 추가 --kubelet-client-certificate=〈client 인증서 파일〉 추가 --kubelet-client-key=〈client 키 파일〉 추가 --service-account-key-file=〈service account 키 파일〉 추가 **- 인증서 관리(apiserver) (인증서설정)** : /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래와 같이 설정 --tls-cert-file=〈tls 인증서 파일〉 추가 --tls-private-key-file=〈tls 키 파일〉 추가 --client-ca-file=〈client ca 인증서 파일〉 추가 **- 안전한 SSL/TLS 버전 사용** : /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래와 같이 설정 --tls-ciphersuites=TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_AES_128_GCM_SHA256
비고	단기 적용(적용 시 개발자 및 운영자 협의)

```
[root@k8s-master osboxes]# cat /etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.10.105
    - --enable-admission-plugins=NodeRestriction
    - --kubelet-https=true
    - --secure-port=6443
    - --kubelet-certificate-authority=/etc/kubernetes/kublnet.crt
    - --kubelet-client-certificate=/etc/kubernetes/clinet-certification.crt
    - --kubelet-client-key=/etc/kubernetes/client-key.key
    - --tls-cert-file=/etc/tls/tls-cert.crt
    - --tls-private-key-file=/etc/tls/tls-key.key
    - --client-ca-file=/etc/kubernetes/client-ca.crt
    - --tls-cipher-suites=TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_ECDSA_WIT
                    H_CHACHA20_POLY1305,TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,TL
              S_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_AES_128_GCM_SHA256
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
```

[그림 37] SSL 인증서 관련 설정

1.4. Admission Control Plugin 설정

<table>
<tr><td>분류</td><td>API Server Configuration</td><td>중요도</td><td>중</td></tr>
<tr><td>항목명</td><td colspan="3">Admission Control Plugin 설정</td></tr>
<tr><td>항목
설명</td><td colspan="3">Admission Control은 클러스터의 사용 방식을 제어하는 플러그인으로 PodSecurityPolicy 와 같은 Kubernetes의 정책 준수에 대해 지원합니다.

■ Admission Control의 주요 역할
<table>
<tr><th>분류</th><th>설명</th></tr>
<tr><td>보안</td><td>Admission Control을 통해 전체 네임 스페이스 또는 클러스터 수준에서의 보안 기준 수립이 가능하며, PodSecurityPolicy을 통한 컨테이너 권한 설정 및 이미지 접근제어, 이미지 내 플래그 권한검사 등 webhook 기반의 보안 기능 수행 가능</td></tr>
<tr><td>거버넌스</td><td>라벨에 대한 유효성 검사, 자동 주석 추가, 리소스 제한 등과 같은 설정에 대한 관리 가능</td></tr>
<tr><td>구성 관리</td><td>클러스터에서 실행 중인 개체의 구성을 확인하고 잘못된 구성으로 인해 클러스터가 손상되는 것을 방지</td></tr>
</table></td></tr>
<tr><td>진단
방법</td><td colspan="3">[진단예시]
- Admission Control 설정 검토
: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
--enable-admission-plugins
--disable-admission-plugins
--admission-control-config-file</td></tr>
<tr><td>설정
방법</td><td colspan="3">- Admission Control 설정 검토
: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래와 같이 설정
Admission Control이 정책을 적용하도록 설정
--enable-admission-plugins=AlwaysAdmit(제거)

매번 새로운 이미지를 사용하도록 설정
--enable-admission-plugins=AlwaysPullImages(추가)

privileged pod에서 exec 및 attach 사용 금지
--enable-admission-plugins=DenyEscalatingExec(추가)

kubelet이 자신의 Node 및 Pod만 수정 가능하도록 제한
--enable-admission-plugins=NodeRestriction 추가

SecurityContext 사용 금지
--enable-admission-plugins=SecurityContextDeny 추가
※ PodSecurityPolicy를 클러스터에서 사용할 수 없는 경우에만 사용
PodSecurityPolicy 사용 시 위 설정은 제외

PodSecurityPolicy 사용</td></tr>
</table>

--enable-admission-plugins=PodSecurityPolicy 추가

기본 서비스 계정 사용 금지
--disable-admission-plugins=ServiceAccount

네임 스페이스 재사용 금지
--disable-admission-plugins=NamespaceLifecycle 제거

API 서버의 요청 승인 속도 제한
--enable-admissions-plugins=EventRateLimit 추가
--admission-control-config-file=〈path/to/configuration/file〉

```
[root@k8s-master osboxes]# cat /etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.10.105
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --enable-admission-plugins=DenyEscalatingExec,NodeRestriction,SecurityContextDeny,PodSecurityPolicy,EventRateLimit
    - --admission-control-config-file=/etc/kubernetes/adminssion-control.config
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
    - --insecure-port=0
    - --service-account-lookup=true
    - --service-account-key-file=/etc/kubernetes/account.key
    - --kubelet-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt
    - --kubelet-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key
```

[그림 38] Admission Control 설정 확인

```
[root@k8s-master osboxes]# cat /etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.10.105
    - --anonymous-auth=false
    - --insecure-bind-address=0.0.0.0
    - --allow-privileged=true
    - --insecure-allow-any-token=true
    - --authorization-mode=Node,RBAC
      --repair-malformed-updates=false
    - --basic-auth-file=/etc/kubernetes/basic/authentificate.auth
    - --token-auth-file=/etc/kubernetes/token.auth
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
```

[그림 39] kube-apiserver.yaml 내 권한 설정 확인

비고	중기 적용(적용 시 개발자 및 운영자 협의)

1.5. 로그 관리

분류	API Server Configuration	중요도	하
항목명	로그 관리		
항목 설명	Kubernetes API Server의 로그 관리를 통해 개별 사용자, 관리자 또는 시스템의 다른 구성 요소에 의해 시스템에 영향을 미친 활동 순서를 문서화하여야 합니다. 담당자는 로그 기록을 정기적으로 확인/감독하여 사용자 접속과 관련하여 오류 및 부정행위가 발생하거나 예상되는 경우 즉각적인 보고 조치가 되도록 해야 합니다. 또한 로그 파일이 위·변조되지 않도록 하기 위해 별도 저장 장치에 백업 보관하고, 쓰기 권한을 제한하여 보관하는 것이 바람직하며, 그 외 수정이 가능하더라도 위·변조 여부를 확인할 수 있는 정보(HMAC 값 또는 전자서명 값) 등을 이용한 별도의 보호조치를 취할 수 있습니다.		

진단 방법	**[진단예시]**
	- 로그 디렉터리 및 파일 설정
	: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
	`--audit-log-path`
	`--audit-log-maxbackup`
	`--audit-log-maxsize`
	
	- 로그 저장 주기
	: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
	`--audit-log-maxage`
	
	- AdvancedAuditing 설정
	: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
	`--audit-policy-file`		

설정 방법	**- 로그 디렉터리 및 파일 권한 설정**
	: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래와 같이 설정
	`--audit-log-path=〈filename〉 추가`
	`--audit-log-maxbackup=10 or 적절한 사이즈 추가`
	`--audit-log-maxsize=100 or 적절한 사이즈 추가`
	
	- 로그 저장 주기
	: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래와 같이 설정
	`--audit-log-maxage=30 or 적절한 기간 추가`
	
	- AdvancedAuditing 설정
	: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래와 같이 설정
	`--audit-policy-file=〈정책 설정 파일〉 추가`
	=〉 정책 설정 파일 내 아래 사항 최소 설정		

rules:
  -level: Metadata

```
[root@k8s-master osboxes]# cat /etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.10.105
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --audit-log-path=/var/log/apiserver/audit.log
    - --audit-log-maxage=30
    - --audit-log-maxbackup=10
    - --audit-log-maxsize=100
    - --profiling=false
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
    - --insecure-port=0
```

[그림 40] kube-apiserver.yaml 파일 내 로그 확인

```
[root@k8s-master osboxes]# cat /etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.10.105
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --audit-policy-file=/etc/kubernetes/audit-policy.yaml
    - --audit-log-path=/var/log/apiserver/audit.log
    - --audit-log-maxage=30
    - --audit-log-maxbackup=10
    - --audit-log-maxsize=100
    - --profiling=false
```

[그림 41] kube-apiserver.yaml 파일 내 audit 설정 확인

비고	단기 적용(적용 시 개발자 및 운영자 협의)

2. Master Node – etcd Configuration

2.1. SSL/TLS 적용

분류	etcd Configuration	중요도	상
항목명	SSL/TLS 적용		
항목 설명	etcd는 중요한 데이터에 대한 액세스를 안정적이고 신속하게 보존하고 제공하도록 설계된 분산된 key-value 저장소로 Kubernetes에서 설정, 네임스페이스 등의 pod/service 상태 저장과 DNS 데이터 저장에 사용되고 있습니다. etcd에 저장되는 DATA는 민감하므로 네트워크 구간 암호화 및 접근하는 클라이언트에 대한 인증을 통한 DATA 보호가 이루어져야 합니다. Kubernetes에서는 etcd와 클라이언트 또는 etcd 클러스터의 다른 peer와 통신 시 SSL/TLS 프로토콜을 이용하여 네트워크 상에서 주요 데이터 보호 및 etcd에 접근을 요청하는 클라이언트, etcd peer에 대한 검증을 위해 사용할 수 있습니다. SSL/TLS 통신 적용을 통해 네트워크 스니핑과 같은 방법으로 etcd 내 주요 정보가 노출되어 다른 공격에 이용되지 않도록 하고, etcd에 접근하는 대상에 대해 검증할 수 있도록 설정하여야 합니다. 또한 SSL/TLS 통신 적용 시에는 주기적으로 인증서를 변경하고 안전한 버전의 암호화 방식을 사용하는 방법을 통해 위험을 최소화할 수 있는 정책 설정이 필요합니다.		

진단 방법	**- SSL/TLS 적용을 통한 클라이언트 인증(etcd peer 및 클라이언트)**
	: /etc/kubernetes/manifests/etcd.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
	`--client-cert-auth`
	`--peer-client-cert-auth (etcd server의 경우 적용 필요 없음)`
	
	- 인증서 관리(etcd peer 및 클라이언트) (인증서설정)
	1) /etc/kubernetes/manifests/etcd.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
	`--cert-file`
	`--key-file`
	`--peer-cert-file`
	`--peer-key-file`
	
	2) /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
	`--etcd-certfile`
	`--etcd-keyfile`
	`--etcd-cafile`
	
	- 인증서 관리(자체서명인증서 사용금지)
	: /etc/kubernetes/manifests/etcd.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
	`--auto-tls`
	`--peer-auto-tls (etcd server의 경우 적용 필요 없음)`		

	--trusted-ca-file
설정 방법	**- SSL/TLS 적용을 통한 클라이언트 인증(etcd peer 및 클라이언트)** : /etc/kubernetes/manifests/etcd.yaml 파일 내 아래와 같이 설정 --client-cert-auth=true 추가 --peer-client-cert-auth=true 수정 (default= false) (etcd server의 경우 적용 필요 없음) **- 인증서 관리(etcd peer 및 클라이언트) (인증서설정)** 1) /etc/kubernetes/manifests/etcd.yaml 파일 내 아래와 같이 설정 --cert-file=〈인증서 파일〉 추가 --key-file=〈키 파일〉 추가 --peer-cert-file=〈peer 인증서 파일〉 --peer-key-file=〈peer 키 파일〉 추가 2) /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래와 같이 설정 --etcd-certfile=〈etcd cert인증서 파일〉 추가 --etcd-keyfile=〈etcd 키 파일〉 추가 --etcd-cafile=〈etcd ca인증서 파일〉 추가 **- etcd 인증서 관리(자체서명인증서 사용금지)** : /etc/kubernetes/manifests/etcd.yaml 파일 내 아래와 같이 설정 --auto-tls=false --peer-auto-tls=false or 제거 (etcd server의 경우 적용 필요 없음) --trusted-ca-file=〈인증서 파일〉 추가

```
[root@k8s-master server]# cat /etc/kubernetes/manifests/etcd.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: etcd
    tier: control-plane
  name: etcd
  namespace: kube-system
spec:
  containers:
  - command:
    - etcd
    - --advertise-client-urls=https://192.168.10.105:2379
    - --cert-file=/etc/kubernetes/pki/etcd/server.crt
    - --client-cert-auth=true
    - --data-dir=/var/lib/etcd
    - --initial-advertise-peer-urls=https://192.168.10.105:2380
    - --initial-cluster=k8s-master=https://192.168.10.105:2380
    - --key-file=/etc/kubernetes/pki/etcd/server.key
    - --listen-client-urls=https://127.0.0.1:2379,https://192.168.10.105:2379
    - --listen-peer-urls=https://192.168.10.105:2380
    - --name=k8s-master
    - --peer-cert-file=/etc/kubernetes/pki/etcd/peer.crt
    - --peer-client-cert-auth=true
    - --peer-key-file=/etc/kubernetes/pki/etcd/peer.key
    - --peer-trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
    - --snapshot-count=10000
    - --trusted-ca-file=/etc/kubernetes/pki/etcd/ca.crt
    image: k8s.gcr.io/etcd:3.3.10
```

[그림 42] etcd.yaml 내 인증서 설정 확인

```
[root@k8s-master server]# cat /etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.10.105
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
```

[그림 43] kube-apiserver.yaml 내 인증서 설정 확인

비고	중기 적용(적용 시 개발자 및 운영자 협의)

2.2. etcd 암호화

분류	etcd Configuration	중요도	중
항목명	etcd 암호화		
항목 설명	etcd는 중요한 데이터에 대한 액세스를 안정적이고 신속하게 보존하고 제공하도록 설계된 분산된 key-value 저장소로 Kubernetes에서 설정, 네임스페이스 등의 pod/service 상태 저장과 DNS 데이터 저장에 사용되고 있습니다. etcd에 저장되는 DATA는 민감하므로 암호화된 상태로 저장되어야 합니다. DATA 암호화 시에는 DES, MD5, RC4 등과 같이 암호 해독기법이 공개된 알고리즘 사용 시 시스템 환경에 따라 수시간에서 수일 내에 해독이 가능하여 암호화 값의 평문 형태 확인이 가능합니다. 이를 통해 정보 유출/변조가 가능하기 때문에 검증된 안전한 알고리즘을 사용해야 합니다.		

진단 방법	**- etcd 암호화 적용**
	: /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
	`--experimental-encryption-provider-config`
	
	- 안전한 암호화 방식 사용
	1) 아래 명령어 실행 후 --experimental-encryption-provider-config 값 확인
	`ps -ef | grep kube-apiserver`
	
	2) --experimental-encryption-provider-config 내 설정된 경로의 암호화 설정 파일 분석을 통해 안전한 암호화 방식 사용 여부 확인		

설정 방법	**- etcd 암호화 적용** : /etc/kubernetes/manifests/kube-apiserver.yaml 파일 내 아래와 같이 설정 `--experimental-encryption-provider-config=〈/path/to/EncryptionConfig/File〉 추가` **- 안전한 암호화 방식 사용** : 파일 내 아래와 같이 설정(aescbc 권고) 예시) kind: EncryptionConfig apiVersion: v1 resources: - resources: - secrets providers: - aescbc: keys: - name: key1 secret: 〈32-byte base64-encoded secret〉		

```
[root@k8s-master osboxes]# cat /etc/kubernetes/manifests/kube-apiserver.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --advertise-address=192.168.10.105
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --experimental-encryption-provider-config=/etc/kubernetes/encrypt.conf
    - --audit-policy-file=/etc/kubernetes/audit-policy.yaml
    - --audit-log-path=/var/log/apiserver/audit.log
    - --audit-log-maxage=30
    - --audit-log-maxbackup=10
    - --audit-log-maxsize=100
    - --profiling=false
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
```

[그림 44] kube-apiserver.yaml 내 암호화 설정 확인

비고	중기 적용(적용 시 개발자 및 운영자 협의)

3. Master Node – Controller Manager Configuration

3.1. Controller 인증 제어

<table>
<tr><td>분류</td><td>Controller Manager Configuration</td><td>중요도</td><td>중</td></tr>
<tr><td>항목명</td><td colspan="3">Controller 인증 제어</td></tr>
<tr><td>항목
설명</td><td colspan="3">Kubernetes에서 Controller는 API Server를 통해 클러스터의 공유 상태를 감시하고 현재 상태를 원하는 상태로 이동하려고 변경하는 제어 루프입니다. 논리적으로, 각 컨트롤러는 별도의 프로세스이지만 복잡성을 줄이기 위해 모두 단일 바이너리로 컴파일되고 단일 프로세스로 실행됩니다.

Kubernetes의 Controller에는 replication controller, endpoints controller, namespace controller 및 service accounts controller가 있으며, 이러한 각각의 개별 Controller가 Pod를 관리하고 있습니다.

Controller Manager는 설정에 따라 이러한 각 컨트롤러를 모니터링하고 클러스터의 공유 상태를 감시, 제어하고 있습니다.

■ Kubernetes controller
<table>
<tr><th>구분</th><th>설명</th></tr>
<tr><td>Node controller</td><td>노드에 대한 상태 체크 및 전파</td></tr>
<tr><td>Replication controller</td><td>시스템상의 모든 replication controller 객체에 대해 올바른 수의 pod를 유지 관리</td></tr>
<tr><td>Endpoints controller</td><td>서비스 및 Pod 접근 관리</td></tr>
<tr><td>Service Account &
Token Controllers</td><td>새 네임스페이스에 대한 기본 계정 및 API 액세스 토큰 생성</td></tr>
</table></td></tr>
<tr><td>진단
방법</td><td colspan="3">[진단예시]
- 각 컨트롤러에 대해 개별 서비스 계정 자격 증명
: /etc/kubernetes/manifests/kube-controller-manager.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
<code>--use-service-account-credentials</code>

- 컨트롤러 계정 자격증명에 사용되는 인증서 관리
: /etc/kubernetes/manifests/kube-controller-manager.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
<code>--service-account-private-key-file</code></td></tr>
<tr><td>설정
방법</td><td colspan="3">- 각 컨트롤러에 대해 개별 서비스 계정 자격 증명
: /etc/kubernetes/manifests/kube-controller-manager.yaml 파일 내 아래와 같이 설정
<code>--use-service-account-credentials=true (default)</code>

- 컨트롤러 계정 자격증명에 사용되는 인증서 관리
: /etc/kubernetes/manifests/kube-controller-manager.yaml 파일 내 아래와 같이 설정
<code>--service-account-private-key-file=/etc/kubernetes/pki/sa.key (default)</code></td></tr>
</table>

```
[root@k8s-master osboxes]# cat /etc/kubernetes/manifests/kube-controller-manager.yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    component: kube-controller-manager
    tier: control-plane
  name: kube-controller-manager
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-controller-manager
    - --authentication-kubeconfig=/etc/kubernetes/controller-manager.conf
    - --authorization-kubeconfig=/etc/kubernetes/controller-manager.conf
    - --bind-address=127.0.0.1
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --cluster-signing-cert-file=/etc/kubernetes/pki/ca.crt
    - --cluster-signing-key-file=/etc/kubernetes/pki/ca.key
    - --controllers=*,bootstrapsigner,tokencleaner
    - --kubeconfig=/etc/kubernetes/controller-manager.conf
    - --leader-elect=true
    - --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt
    - --root-ca-file=/etc/kubernetes/pki/ca.crt
    - --service-account-private-key-file=/etc/kubernetes/pki/sa.key
    - --use-service-account-credentials=true
    image: k8s.gcr.io/kube-controller-manager:v1.14.1
    imagePullPolicy: IfNotPresent
    livenessProbe:
      failureThreshold: 8
      httpGet:
```

[그림 45] kube-controller-manager.yaml 내 인증서 설정 확인

비고	중기 적용(적용 시 개발자 및 운영자 협의)

3.2. SSL/TLS 적용

분류	Controller Manager Configuration		중
항목명	SSL/TLS 적용		
항목 설명	Kubernetes에서는 SSL/TLS를 이용하여 네트워크 상의 DATA 보호 및 각 구성요소에 접하는 클라이언트에 대한 검증을 위해 사용할 수 있습니다. SSL/TLS 통신 적용을 통해 네트워크 스니핑과 같은 방법으로 주요 정보가 노출되어 다른 공격에 이용되지 않도록 하고, API server에 접근하는 대상에 대해 검증할 수 있도록 설정하여야 합니다. 또한 SSL/TLS 통신 적용 시에는 주기적으로 인증서를 변경하고 안전한 버전의 암호화 방식을 사용하는 방법을 통해 위험을 최소화할 수 있는 정책 설정이 필요합니다.		

진단 방법	**[진단예시]**
	- SSL/TLS 적용을 통한 클라이언트 인증(포드)
	: /etc/kubernetes/manifests/kube-controller-manager.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
	`--root-ca-file`
	
	- 인증서 관리(인증서 교환주기 설정)
	: /etc/kubernetes/manifests/kube-controller-manager.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
	`--feature-gates`		

설정 방법	**[진단예시]**
	- SSL/TLS 적용을 통한 클라이언트 인증(포드)
	: /etc/kubernetes/manifests/kube-controller-manager.yaml 파일 내 아래와 같이 설정
	`--root-ca-file=/etc/kubernetes/pki/ca.crt (default)`
	
	- 인증서 관리(인증서 교환주기 설정)
	: /etc/kubernetes/manifests/kube-controller-manager.yaml 파일 내 아래와 같이 설정
	`--feature-gates=RotateKubeletServerCertificate=true 추가`		

```
[root@k8s-master osboxes]# cat /etc/kubernetes/manifests/kube-controller-manager.yaml
apiVersion: v1
metadata:
  creationTimestamp: null
  labels:
    component: kube-controller-manager
    tier: control-plane
  name: kube-controller-manager
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-controller-manager
    - --authentication-kubeconfig=/etc/kubernetes/controller-manager.conf
    - --authorization-kubeconfig=/etc/kubernetes/controller-manager.conf
    - --bind-address=127.0.0.1
    - --root-ca-file=/etc/kubernetes/root-ca.crt
    - --feature-gates=RotateKubeletServerCertificate=true
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --cluster-signing-cert-file=/etc/kubernetes/pki/ca.crt
    - --cluster-signing-key-file=/etc/kubernetes/pki/ca.key
    - --controllers=*,bootstrapsigner,tokencleaner
    - --kubeconfig=/etc/kubernetes/controller-manager.conf
    - --leader-elect=true
    - --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt
    - --service-account-private-key-file=/etc/kubernetes/pki/sa.key
    - --use-service-account-credentials=true
    image: k8s.gcr.io/kube-controller-manager:v1.14.1
    imagePullPolicy: IfNotPresent
```

[그림 46] kube-controller-manager.yaml 내 SSL/TLS 설정 확인

비고	중기 적용(적용 시 개발자 및 운영자 협의)

4. Master Node – PodSecurityPolicy Configuration

4.1. 컨테이너 권한 제어

<table>
<tr><td>분류</td><td>PodSecurityPolicy Configuration</td><td>중요도</td><td>상</td></tr>
<tr><td>항목명</td><td colspan="3">컨테이너 권한 제어</td></tr>
<tr><td>항목
설명</td><td colspan="3">Pod Security Policy(PSP)는 클러스터 전체에 적용되는 Pod 보안 정책으로, Admission Control Plugin을 통해 활성화됩니다.

Pod Security Policy를 통해 생성되는 Pod 내 컨테이너의 권한에 대한 정책 설정이 가능하며 Pod 내 컨테이너가 불필요하게 과도한 권한을 가지지 않도록 하여야 합니다.

Pod Security Policy는 보안을 정의하고 제어할 수 있는 클러스터 수준의 자원입니다.

PodSecurityPolicy를 통해 Pod 내 컨테이너의 권한에 대한 정책 및 볼륨 및 파일 권한 등 Pod 내 컨테이너가 불필요하게 과도한 권한을 가지지 않도록 해야 하며 네트워크 및 포트에 대한 정책 설정을 통해 접근 제어를 구현해야 합니다.

■ Pod Security Policy(PSP)를 통한 제어 기능
<table>
<tr><th>Policy 내 설정</th><th>설명</th></tr>
<tr><td>privileged</td><td>권한있는 컨테이너 실행</td></tr>
<tr><td>hostPID, hostIPC</td><td>호스트 네임 스페이스 사용 제어</td></tr>
<tr><td>hostNetwork, hostPorts</td><td>호스트 네트워킹 및 포트 사용 제어</td></tr>
<tr><td>volumes</td><td>volumes 사용 제어</td></tr>
<tr><td>allowedHostPaths</td><td>호스트 파일 시스템의 사용 제어</td></tr>
<tr><td>allowedFlexVolumes</td><td>Flex volume 드라이버 설정</td></tr>
<tr><td>fsGroup</td><td>Pod의 volume을 소유한 FSGroup 할당</td></tr>
<tr><td>readOnlyRootFilesystem</td><td>읽기 전용 root filesystem 설정</td></tr>
<tr><td>runAsUser, runAsGroup,
supplementalGroups</td><td>컨테이너의 사용자 및 그룹 ID 지정</td></tr>
<tr><td>allowPrivilegeEscalation,
defaultAllowPrivilegeEscalation</td><td>루트 권한 부여 제어</td></tr>
<tr><td>defaultAddCapabilities,
requiredDropCapabilities, allowedCapabilities</td><td>Linux 기능 사용 여부</td></tr>
<tr><td>seLinux</td><td>컨테이너의 SELinux 프로필</td></tr>
<tr><td>allowedProcMountTypes</td><td>컨테이너의 마운트 타입 설정</td></tr>
<tr><td>annotations</td><td>컨테이너의 AppArmor, seccomp, sysctl 프로필</td></tr>
</table></td></tr>
<tr><td>진단
방법</td><td colspan="3">[진단예시]
- 컨테이너 권한 제어
: PSP 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
kubectl get psp 〈name〉 -o=jsonpath='{.spec.allowPrivilegeEscalation}'
kubectl get psp 〈name〉 -o=jsonpath='{.spec.privileged}'
kubectl get psp 〈name〉 -o=jsonpath='{.spec.runAsUser.rule}'
kubectl get psp 〈name〉 -o=jsonpath='{.spec.requiredDropCapabilities}'</td></tr>
</table>

설정 방법

- 컨테이너 권한 제어

: PSP 내 아래와 같이 설정

Privilege 컨테이너 실행 제한 1
.spec.allowPrivilegeEscalation 필드 삭제 or .spec.allowPrivilegeEscalation=false 설정

Privilege 컨테이너 실행 제한 2
※ 서비스상 필요할 경우 예외처리 요청
.spec.privileged 필드 삭제 or .spec.privileged=false 설정

컨테이너가 루트 사용자로 실행되는 것을 제한
.spec.runAsUser.rule 필드 0을 포함하지 않는 UID 범위로 MustRunAsNonRoot or MustRunAs로 설정
※ 서비스상 필요할 경우 예외처리 요청

컨테이너의 NET_RAW 기능 제한
requiredDropCapabilities필드 NET_RAW or ALL 설정
※ 서비스상 필요할 경우 예외처리 요청

```
[root@k8s-master osboxes]# cat restricted.yaml
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: 'docker/default'
    apparmor.security.beta.kubernetes.io/allowedProfileNames: 'runtime/default'
    seccomp.security.alpha.kubernetes.io/defaultProfileName:  'docker/default'
    apparmor.security.beta.kubernetes.io/defaultProfileName:  'runtime/default'
spec:
  privileged: false

  # Required to prevent escalations to root.
  allowPrivilegeEscalation: false
  # This is redundant with non-root + disallow privilege escalation,
  # but we can provide it for defense in depth.
  requiredDropCapabilities:
    - ALL
  # Allow core volume types.
  volumes:
    - 'configMap'
    - 'emptyDir'
    - 'projected'
    - 'secret'
    - 'downwardAPI'
    # Assume that persistentVolumes set up by the cluster admin are safe to use.
    - 'persistentVolumeClaim'
```

[그림 47] PodSecurityPolicy 내 권한 설정 확인

비고

장기 적용(적용 시 개발자 및 운영자 협의)

4.2. 네임스페이스 관리

<table>
<tr><td>분류</td><td>PodSecurityPolicy Configuration</td><td>중요도</td><td>중</td></tr>
<tr><td>항목명</td><td colspan="3">네임스페이스 관리</td></tr>
<tr><td>항목
설명</td><td colspan="3">Kubernetes는 하나의 물리적 클러스터 내 여러 개의 논리적 가상 클러스터를 생성하도록 지원하고 있으며, 이러한 가상 클러스터를 네임스페이스라고 합니다.

네임스페이스 별 리소스 할당, 사용자 접근 권한 제어를 통해 Kubernetes 물리적 리소스 관리에 용이하고 프로젝트 내 여러 사용자 또는 팀이 분산되어있는 환경에서 효율적인 운영을 가능하게 합니다.

Pod Security Policy를 통해 네임스페이스에 대한 정책 설정이 가능하며 네임스페이스에서 실행되는 컨테이너가 불필요하게 과도한 권한을 가지지 않도록 하여야 합니다.

■ Pod Security Policy(PSP) 내 namespace 설정
<table>
<tr><th>구분</th><th>설명</th></tr>
<tr><td>HostPID</td><td>Pod 내 컨테이너가 호스트의 PID 네임스페이스 공유 여부 제어</td></tr>
<tr><td>HostIPC</td><td>Pod 내 컨테이너가 호스트의 IPC 네임스페이스 공유 여부 제어</td></tr>
<tr><td>HostNetwork</td><td>Pod가 노드의 Network 네임스페이스 사용 여부 제어</td></tr>
<tr><td>HostPorts</td><td>호스트의 Network 네임스페이스에서 허용되는 포트 범위 제어</td></tr>
<tr><td>AllowedHostPaths</td><td>hostPath 볼륨에서 사용할 수 있는 호스트 경로의 화이트리스트를 지정</td></tr>
</table></td></tr>
<tr><td>진단
방법</td><td colspan="3">[진단예시]
- 네임스페이스 공유 금지
: PSP 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인
kubectl get psp 〈name〉 -o=jsonpath='{.spec.hostPID}'
kubectl get psp 〈name〉 -o=jsonpath='{.spec.hostIPC}'
kubectl get psp 〈name〉 -o=jsonpath='{.spec.hostNetwork}'</td></tr>
<tr><td>설정
방법</td><td colspan="3">- 네임스페이스 공유 금지
: PSP 내 아래와 같이 설정
.spec.hostPID 필드 삭제 or .spec.hostPID=false 설정
.spec.hostIPC 필드 삭제 or .spec.hostIPC=false 설정
.spec.hostNetwork 필드 삭제 or .spec.hostNetwork=false 설정</td></tr>
</table>

```
[root@k8s-master osboxes]# cat restricted.yaml
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: restricted
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: 'docker/default'
    apparmor.security.beta.kubernetes.io/allowedProfileNames: 'runtime/default'
    seccomp.security.alpha.kubernetes.io/defaultProfileName:  'docker/default'
    apparmor.security.beta.kubernetes.io/defaultProfileName:  'runtime/default'
spec:
  privileged: false

  # Required to prevent escalations to root.
  allowPrivilegeEscalation: false
  # This is redundant with non-root + disallow privilege escalation,
  # but we can provide it for defense in depth.
  requiredDropCapabilities:
    - ALL
  # Allow core volume types.
  volumes:
    - 'configMap'
    - 'emptyDir'
    - 'projected'
    - 'secret'
    - 'downwardAPI'
    # Assume that persistentVolumes set up by the cluster admin are safe to use.
    - 'persistentVolumeClaim'
  hostNetwork: false
  hostIPC: false
  hostPID: true
  runAsUser:
    # Require the container to run without root privileges.
    rule: 'MustRunAsNonRoot'
```

[그림 48] PodSecurityPolicy 내 네임스페이스 설정 확인

비고	단기 적용(적용 시 개발자 및 운영자 협의)

5. Master Node - Host OS

5.1. 설정 파일 권한 설정

분류	Host OS	중요도	하
항목명	설정 파일 권한 설정		
항목 설명	Kubernetes 설정 파일의 퍼미션이 잘못 설정된 경우 비인가자가 다양한 방법으로 Kubernetes 설정을 변경하여 침해사고를 일으킬 가능성이 있습니다. 따라서, root 사용자/그룹이 소유권을 가지고 root 외 다른 사용자가 이 파일을 수정할 수 없도록 파일의 권한을 제한하여 파일의 무결성을 유지하여야 합니다.		

진단 방법	**[진단예시]**
	- 설정 파일 권한 설정
	: 파일 확인 후, 소유권 및 그룹 권한 확인
	`# kube-apiserver.yaml`
	`stat -c %a /etc/kubernetes/manifests/kube-apiserver.yaml`
	`stat -c %U:%G /etc/kubernetes/manifests/kube-apiserver.yaml`
	
	`# kube-controller-manager.yaml`
	`stat -c %a /etc/kubernetes/manifests/kube-controller-manager.yaml`
	`stat -c %U:%G /etc/kubernetes/manifests/kube-controller-manager.yaml`
	
	`# kube-scheduler.yaml`
	`stat -c %a /etc/kubernetes/manifests/kube-scheduler.yaml`
	`stat -c %U:%G /etc/kubernetes/manifests/kube-scheduler.yaml`
	
	`# etcd.yaml`
	`stat -c %a /etc/kubernetes/manifests/etcd.yaml`
	`stat -c %U:%G /etc/kubernetes/manifests/etcd.yaml`
	
	`# Container Network Interface 파일 (default : /etc/cni/net.d)`
	※ 서비스상 필요할 경우 예외처리 요청
	`stat -c %a <CNI 파일 경로>`
	`stat -c %U:%G <CNI 파일 경로>`
	
	`# admin.conf`
	`stat -c %a /etc/kubernetes/admin.conf`
	`stat -c %U:%G /etc/kubernetes/admin.conf`
	
	`# scheduler.conf`
	`stat -c %a /etc/kubernetes/scheduler.conf`
	`stat -c %U:%G /etc/kubernetes/scheduler.conf`
	
	`# controller-manager.conf`
	`stat -c %a /etc/kubernetes/controller-manager.conf`
	`stat -c %U:%G /etc/kubernetes/controller-manager.conf`		

설정 방법	**- 설정 파일 권한 설정** : 사용자 및 그룹 권한 적용 ``` # kube-scheduler.yaml chmod 644 /etc/kubernetes/manifests/kube-scheduler.yaml chown root:root /etc/kubernetes/manifests/kube-scheduler.yaml # kube-controller-manager.yaml chmod 644 /etc/kubernetes/manifests/kube-controller-manager.yaml chown root:root /etc/kubernetes/manifests/kube-controller-manager.yaml # kube-scheduler.yaml chmod 644 /etc/kubernetes/manifests/kube-scheduler.yaml chown root:root /etc/kubernetes/manifests/kube-scheduler.yaml # etcd.yaml chmod 644 /etc/kubernetes/manifests/etcd.yaml chown root:root /etc/kubernetes/manifests/etcd.yaml # Container Network Interface 파일 chmod 644 <path/to/cni/files> chown root:root <path/to/cni/files> # admin.conf chmod 644 /etc/kubernetes/admin.conf chown root:root /etc/kubernetes/admin.conf # scheduler.conf chmod 644 /etc/kubernetes/scheduler.conf chown root:root /etc/kubernetes/scheduler.conf # controller-manager.conf chmod 644 /etc/kubernetes/controller-manager.conf chown root:root /etc/kubernetes/controller-manager.conf ```
비고	중기 적용(적용 시 개발자 및 운영자 협의)

5.2. etcd 데이터 디렉터리 권한 설정

분류	Host OS	중요도	하
항목명	etcd 데이터 디렉터리 권한 설정		
항목 설명	etcd는 중요한 데이터에 대한 액세스를 안정적이고 신속하게 보존하고 제공하도록 설계된 분산된 key-value 저장소로 Kubernetes 에서 설정, 네임스페이스, 애플리케이션 등의 pod/service 상태 및 DNS 데이터와 같이 민감한 DATA 가 존재하므로 해당 디렉터리는 비인가자의 읽기 또는 쓰기로부터 보호되어야 합니다.		
진단 방법	**[진단예시]** **- etcd 데이터 디렉터리 권한 설정** : 파일 확인 후, 소유권 및 그룹 권한 확인 `ls -alR /var/lib/etcd`		

설정 방법	**- etcd 데이터 디렉터리 권한 설정**
	: 사용자 및 그룹 권한 적용
	`chmod 700 /var/lib/etcd`
	`chown etcd:etcd /var/lib/etcd`		

비고	중기 적용(적용 시 개발자 및 운영자 협의)		

5.3. 인증서 파일 권한 설정

분류	Host OS	중요도	하
항목명	인증서 파일 권한 설정		
항목 설명	Kubernetes 는 SSL/TLS 구성을 통한 네트워크상 DATA 보호 및 사용자 인증을 위해 많은 인증서를 사용합니다. 해당 인증서와 인증서가 포함된 디렉터리가 root 사용자/그룹이 소유권을 가지고 있고 root 외 다른 사용자가 이 파일 및 디렉터리에 접근할 수 없도록 파일의 권한을 제한하여 파일의 무결성을 유지하여야 합니다.		

진단 방법	**[진단예시]**
	- 인증서 파일 권한 설정
	: 파일 확인 후, 소유권 및 그룹 권한 확인
	`# PKI 인증 디렉터리`
	`ls -laR /etc/kubernetes/pki/`
	
	`# PKI 인증서 파일`
	`ls -laR /etc/kubernetes/pki/*.crt`
	
	`# PKI 키 파일`
	`ls -laR /etc/kubernetes/pki/*.key`		

설정 방법	**- 인증서 파일 권한 설정**
	: 사용자 및 그룹 권한 적용
	`# PKI 인증 디렉터리`
	`chown -R root:root /etc/kubernetes/pki/`
	
	`# PKI 인증서 파일`
	`chmod -R 644 /etc/kubernetes/pki/*.crt`
	
	`# PKI 키 파일`
	`chmod -R 600 /etc/kubernetes/pki/*.key`		

비고	중기 적용(적용 시 개발자 및 운영자 협의)		

6. Worker Node – Kubelet Configuration

6.1. Kubelet 인증 제어

분류	Kubelet Configuration	중요도	상
항목명	Kubelet 인증 제어		
항목 설명	Kubelet은 Kubernetes 각 노드에서 실행되는 에이전트로 Pod에 대해 정의된 YAML 또는 JSON 형태의 PodSpec에 따라 컨테이너를 실행하고 관리하는 역할을 합니다. Kubelet 잘못된 인증 구현은 Worker Node 내 Pod, 컨테이너에 대한 비인증 접근 후 정보 노출 및 리소스 수정과 같은 영향을 줄 수 있습니다.		
진단 방법	**[진단예시]** **- 비인증 접근 차단** 1-1) kublet service 파일 사용하는 경우 : /usr/lib/systemd/system/kublet.service.d/10-kubeadm.conf 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인 KUBELET_SYSTEM_PODS_ARGS 내 --anonymous-auth, --read-only-port KUBELET_CADVISOR_ARGS 내 --cadvisor-port 1-2) kubelet config 파일 사용하는 경우 : /var/lib/kubelet/config.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인 authentication readOnlyPort		
설정 방법	**- 비인증 접근 차단** 1-1) kublet service 파일 사용하는 경우 : /usr/lib/systemd/system/kublet.service.d/10-kubeadm.conf 파일 내 아래와 같이 설정 KUBELET_SYSTEM_PODS_ARGS 내 --anonymous-auth=false --read-only-port=0 KUBELET_CADVISOR_ARGS 내 --cadvisor-port=0 1-2) kubelet config 파일 사용하는 경우 : /var/lib/kubelet/config.yaml 파일 내 아래와 같이 설정 authentication: anonymous: enabled: false readOnlyPort :0		

※ 적용 후 kubelet 서비스 재시작 필요

```
systemctl daemon-reload
systemctl restart kubelet.service
```

```
[root@worker-node1 osboxes]# cat /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf
# Note: This dropin only works with kubeadm and kubelet v1.11+
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf"
Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml"
Environment="KUBELET_SYSTEM_PODS_ARGS=--anonymous-auth=false --read-only-port=0 --hostname-override"
# This is a file that "kubeadm init" and "kubeadm join" generates at runtime, populating the KUBELET_KUBEADM_ARGS variable dynamically
EnvironmentFile=-/var/lib/kubelet/kubeadm-flags.env
# This is a file that the user can use for overrides of the kubelet args as a last resort. Preferably, the user should use
# the .NodeRegistration.KubeletExtraArgs object in the configuration files instead. KUBELET_EXTRA_ARGS should be sourced from this file.
EnvironmentFile=-/etc/sysconfig/kubelet
ExecStart=
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS $KUBELET_KUBEADM_ARGS $KUBELET_EXTRA_ARGS
[root@worker-node1 osboxes]#
```

[그림 49] kublet service 파일 내 인증 설정 확인

```
[root@k8s-master ~]# cat /var/lib/kubelet/config.yaml
address: 0.0.0.0
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: false
  webhook:
    cacheTTL: 2m0s
    enabled: true
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
authorization:
  mode: Webhook
  webhook:
    cacheAuthorizedTTL: 5m0s
    cacheUnauthorizedTTL: 30s
readOnlyPort : 0
cgroupDriver: cgroupfs
```

[그림 50] kublet 설정 파일 내 인증 설정 확인

비고	장기 적용(적용 시 개발자 및 운영자 협의)

6.2. Kubelet 권한 제어

분류	Kubelet Configuration	중요도	상
항목명	Kubelet 권한 제어		
항목 설명	Kubelets 는 기본적으로 Kubernetes Master 의 API Server 에서 전달되는 요청에 대해 권한 검사 없이 모두 허용하고 있으므로 설정 변경을 통해 권한 검증을 수행하도록 하여야 합니다.		
진단 방법	**[진단예시]** **- 권한 검증 수행** 1-1) kublet service 파일 사용하는 경우 : /usr/lib/systemd/system/kublet.service.d/10-kubeadm.conf 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인 KUBELET_AUTHZ_ARGS 내 --authorization-mode 1-2) kubelet config 파일 사용하는 경우 : /var/lib/kubelet/config.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인 authorization		
설정 방법	**- 권한 검증 수행** 1-1) kublet service 파일 사용하는 경우 : /usr/lib/systemd/system/kublet.service.d/10-kubeadm.conf 파일 내 아래와 같이 설정 KUBELET_AUTHZ_ARGS 내 --authorization-mode=Webhook 1-2) kubelet config 파일 사용하는 경우 : : /var/lib/kubelet/config.yaml 파일 내 아래와 같이 설정 authorization: mode: Webhook ※ 적용 후 kubelet 서비스 재시작 필요 systemctl daemon-reload systemctl restart kubelet.service		

```
[root@worker-node1 osboxes]# cat /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf
# Note: This dropin only works with kubeadm and kubelet v1.11+
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf"
Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml"
Environment="KUBELET_SYSTEM_AUTHZ_ARGS=--authorization-mode=Webhook"
# This is a file that "kubeadm init" and "kubeadm join" generates at runtime, populating the KUBELET_KUBEADM_ARGS variable dynamically
EnvironmentFile=-/var/lib/kubelet/kubeadm-flags.env
# This is a file that the user can use for overrides of the kubelet args as a last resort. Preferably, the user should use
# the .NodeRegistration.KubeletExtraArgs object in the configuration files instead. KUBELET_EXTRA_ARGS should be sourced from this file.
EnvironmentFile=-/etc/sysconfig/kubelet
ExecStart=
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS $KUBELET_KUBEADM_ARGS $KUBELET_EXTRA_ARGS
[root@worker-node1 osboxes]#
```

[그림 51] kublet service 파일 내 권한 설정 확인

[그림 52] kublet 설정 파일 내 권한 설정 확인

비고	장기 적용(적용 시 개발자 및 운영자 협의)

6.3. SSL/TLS 적용

분류	Kubelet Configuration	중요도	상
항목명	SSL/TLS 적용		
항목 설명	Kubernetes에서는 SSL/TLS를 이용하여 네트워크상의 DATA 보호 및 각 구성요소에 접하는 클라이언트에 대한 검증을 위해 사용할 수 있습니다. SSL/TLS 통신 적용을 통해 네트워크 스니핑과 같은 방법으로 주요 정보가 노출되어 다른 공격에 이용되지 않도록 하고, API server에 접근하는 대상에 대해 검증할 수 있도록 설정하여야 합니다. 또한 SSL/TLS 통신 적용 시에는 주기적으로 인증서를 변경하고 안전한 버전의 암호화 방식을 사용하는 방법을 통해 위험을 최소화할 수 있는 정책 설정이 필요합니다.		
진단 방법	**[진단예시]** **- SSL/TLS 적용을 통한 클라이언트 인증(kubelet to apiserver)** 1-1) kublet service 파일 사용하는 경우 : /usr/lib/systemd/system/kublet.service.d/10-kubeadm.conf 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인 KUBELET_AUTHZ_ARGS 내 --client-ca-file 1-2) kubelet config 파일 사용하는 경우 : /var/lib/kubelet/config.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 authentication: x509: clientCAFile **- 인증서 관리(Kubelets) (인증서설정)** 1-1) kublet service 파일 사용하는 경우 : /usr/lib/systemd/system/kublet.service.d/10-kubeadm.conf 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인 KUBELET_CERTIFICATE_ARGS 내 --tls-cert-file, --tls-private-key-file 1-2) kubelet config 파일 사용하는 경우 : /var/lib/kubelet/config.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 TlsCertFile, tlsPrivateKeyFile **- 인증서 관리(인증서 교환주기 설정)** 1-1) kublet service 파일 사용하는 경우 : /usr/lib/systemd/system/kublet.service.d/10-kubeadm.conf 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인 KUBELET_CERTIFICATE_ARGS 내 --rotate-certificates, --feature-gates=RotateKubeletServerCertificate 1-2) kubelet config 파일 사용하는 경우		

	: /var/lib/kubelet/config.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 rotateCertificates: true. **- 안전한 SSL/TLS 버전 사용** 1-1) kublet service 파일 사용하는 경우 : 아래 명령어 실행 후 --tls-cipher-suites 값 확인 ps -ef | grep kubelet 1-2) kubelet config 파일 사용하는 경우 : /var/lib/kubelet/config.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 TLSCipherSuites **- node hostname 임의 변경 금지** : /usr/lib/systemd/system/kublet.service.d/10-kubeadm.conf 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인 KUBELET_SYSTEM_PODS_ARGS 내 --hostname-override
설정 방법	**- SSL/TLS 적용을 통한 클라이언트 인증(kubelet to apiserver)** 1-1) kublet service 파일 사용하는 경우 : /usr/lib/systemd/system/kublet.service.d/10-kubeadm.conf 파일 내 아래와 같이 설정 KUBELET_AUTHZ_ARGS 내 --client-ca-file =〈client ca인증서 위치〉 1-2) kubelet config 파일 사용하는 경우 : /var/lib/kubelet/config.yaml 파일 내 아래와 같이 설정 authentication: x509: clientCAFile: client CA 파일 경로 **- 인증서 관리(Kubelets) (인증서설정)** 1-1) kublet service 파일 사용하는 경우 : /usr/lib/systemd/system/kublet.service.d/10-kubeadm.conf 파일 내 아래와 같이 설정 KUBELET_CERTIFICATE_ARGS 내 --tls-cert-file=〈tls인증서 위치〉 --tls-private-key-file=〈tls키 파일 위치〉 1-2) kubelet config 파일 사용하는 경우 : /var/lib/kubelet/config.yaml 파일 내 아래와 같이 설정 tlsCertFile: tls인증서 위치 tlsPrivateKeyFile: tls키 파일 위치

- 인증서 관리(인증서 교환주기 설정)

1-1) kublet service 파일 사용하는 경우

: /usr/lib/systemd/system/kublet.service.d/10-kubeadm.conf 파일 내 아래와 같이 설정

```
KUBELET_CERTIFICATE_ARGS 내
    --rotate-certificates=true
    --feature-gates=RotateKubeletServerCertificate=true
```

1-2) kubelet config 파일 사용하는 경우

: /var/lib/kubelet/config.yaml 파일 내 아래와 같이 설정

```
rotateCertificates: true.
```

- 안전한 SSL/TLS 버전 사용

1-1) kublet service 파일 사용하는 경우

: /usr/lib/systemd/system/kublet.service.d/10-kubeadm.conf 파일 내 아래와 같이 설정

```
    --tls-
ciphersuites=TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM
_SHA256,TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_RSA_WITH_AES_256_GCM
_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_ECDSA_WITH_AES_256_GCM
_SHA384,TLS_RSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_AES_128_GCM_SHA256
```

1-2) kubelet config 파일 사용하는 경우

: /var/lib/kubelet/config.yaml 파일 내 아래와 같이 설정

```
TLSCipherSuites:
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,TLS_ECDHE_RSA_WITH_AES_128_GCM
_SHA256,TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_RSA_WITH_AES_256_GCM
_SHA384,TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305,TLS_ECDHE_ECDSA_WITH_AES_256_GCM
_SHA384,TLS_RSA_WITH_AES_256_GCM_SHA384,TLS_RSA_WITH_AES_128_GCM_SHA256
```

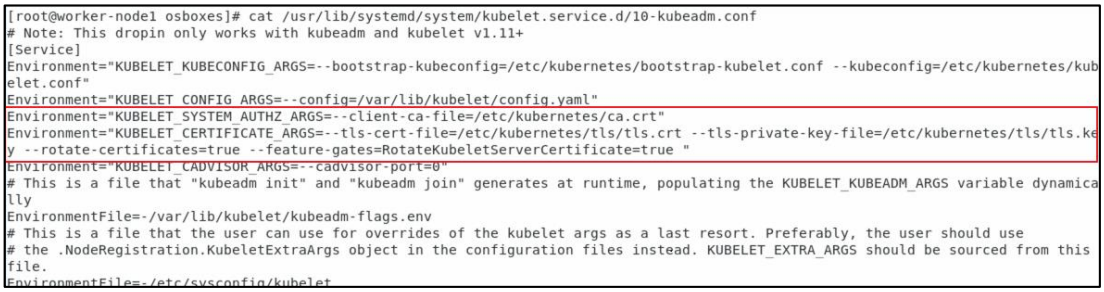
- node hostname 임의 변경 금지

: /usr/lib/systemd/system/kublet.service.d/10-kubeadm.conf 파일 내 아래와 같이 설정

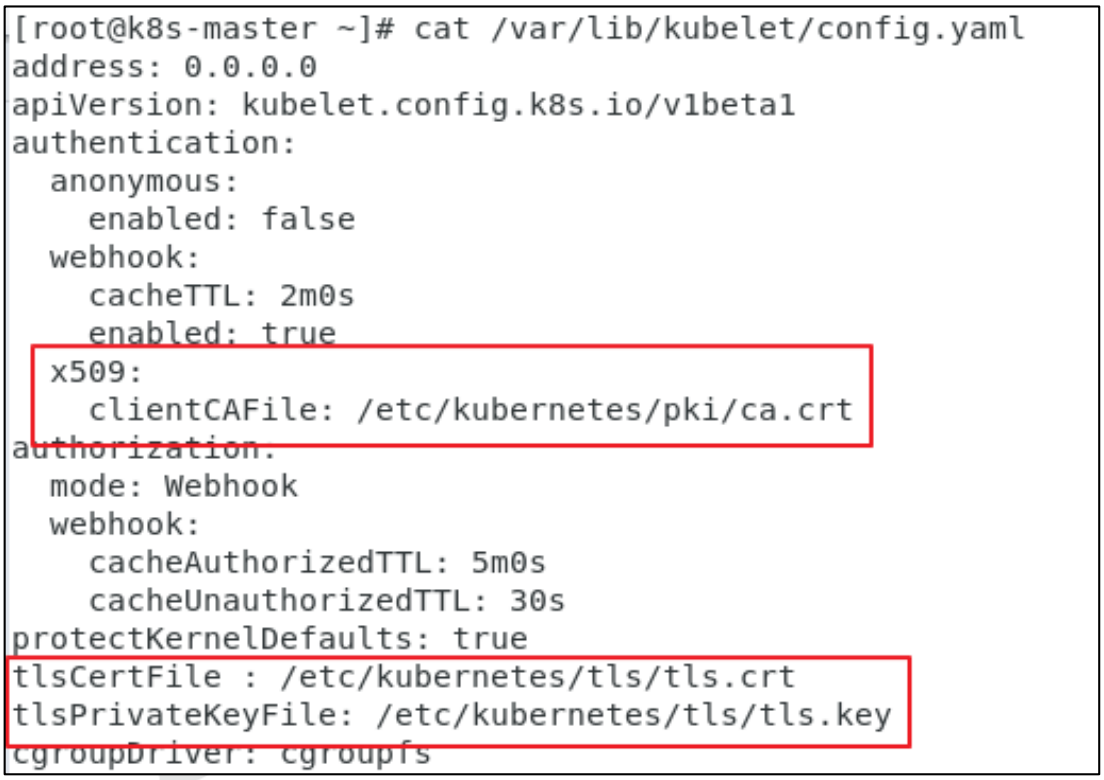
```
KUBELET_SYSTEM_PODS_ARGS 내
    --hostname-override 존재할 경우 제거
```

※ 적용 후 kubelet 서비스 재시작 필요

```
systemctl daemon-reload
systemctl restart kubelet.service
```



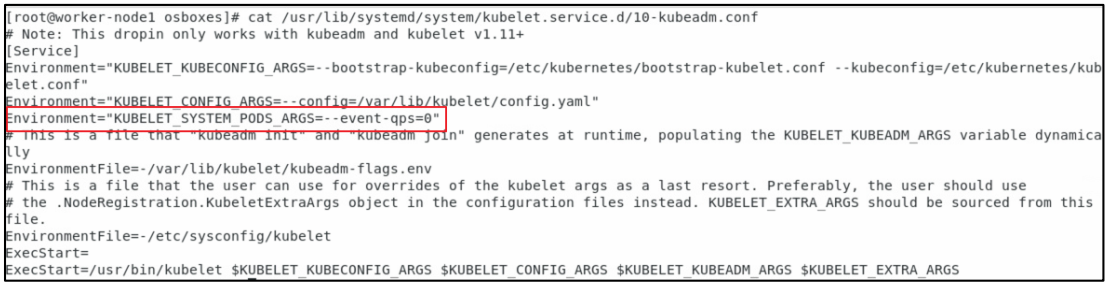
[그림 53] kublet service 파일 내 SSL/TLS 설정 확인



[그림 54] kublet 설정 파일 내 SSL/TLS 설정 확인

비고	중기 적용(적용 시 개발자 및 운영자 협의)

6.4. 로그 관리

분류	Kubelet Configuration	중요도	하
항목명	로그 관리		
항목 설명	Kubelet의 로그 관리를 통해 개별 사용자, 관리자 또는 시스템의 다른 구성 요소에 의해 시스템에 영향을 미친 활동 순서를 문서화해야 합니다. 담당자는 로그 기록을 정기적으로 확인·감독하여 사용자 접속과 관련하여 오류 및 부정행위가 발생하거나 예상되는 경우 즉각적인 보고 조치가 되도록 해야 합니다. 또한 로그 파일이 위·변조되지 않도록 하기 위해 별도 저장 장치에 백업 보관하고, 쓰기 권한을 제한하여 보관하는 것이 바람직하며, 그 외 수정이 가능하더라도 위·변조 여부를 확인할 수 있는 정보(HMAC 값 또는 전자서명 값) 등을 이용한 별도의 보호조치를 취할 수 있습니다.		
진단 방법	**[진단예시]** **- 이벤트 생성** 1-1) kublet service 파일 사용하는 경우 : /usr/lib/systemd/system/kublet.service.d/10-kubeadm.conf 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인 KUBELET_SYSTEM_PODS_ARGS 내 --event-qps 1-2) kubelet config 파일 사용하는 경우 : /var/lib/kubelet/config.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인 eventRecordQPS		
설정 방법	**- 이벤트 생성** 1-1) kublet service 파일 사용하는 경우 : /usr/lib/systemd/system/kublet.service.d/10-kubeadm.conf 파일 내 아래와 같이 설정 KUBELET_SYSTEM_PODS_ARGS 내 --event-qps=0 1-2) kubelet config 파일 사용하는 경우 : /var/lib/kubelet/config.yaml 파일 내 아래와 같이 설정 eventRecordQPS: 0. ※ kubelet 서비스 재시작 필요 systemctl daemon-reload systemctl restart kubelet.service  [그림 55] kublet service 파일 내 로그 설정 확인		

```
[root@k8s-master ~]# cat /var/lib/kubelet/config.yaml
address: 0.0.0.0
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: false
  webhook:
    cacheTTL: 2m0s
    enabled: true
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
authorization:
  mode: Webhook
  webhook:
    cacheAuthorizedTTL: 5m0s
    cacheUnauthorizedTTL: 30s
readOnlyPort : 0
eventRecordQPS: 5
cgroupDriver: cgroupfs
```

[그림 56] kublet 설정 파일 내 로그 설정 확인

비고	단기 적용(적용 시 개발자 및 운영자 협의)

6.5. Kernel 파라미터 설정

분류	Kubelet Configuration	중요도	중
항목명	Kubelet 파라미터 설정		
항목 설명	리눅스의 Kernel 파라미터의 Tuning 작업을 통해 기본적인 IP Forwarding 이나 ip_forward_directed_broadcasts, arp 관련 ip_forward_src_routed 들에 대한 Tuning 과 같은 보안 기능을 적용하여 서비스 거부 공격(DOS)의 경유지에 이용, Spoofing(IP Address 변조 등의 공격)등을 사전에 차단할 수 있습니다. Kubernetes 에서는 기본으로 설정된 Kernel 파라미터가 존재하며 기본적으로 적용되어 있습니다. 만일 이 기본값이 조직 운영 정책과 다른 경우 원하지 않는 커널 기능이 존재하는 포드가 실행될 수 있습니다.		
진단 방법	**[진단예시]** **- default Kernel 파라미터 사용 금지** 1-1) kublet service 파일 사용하는 경우 : /usr/lib/systemd/system/kublet.service.d/10-kubeadm.conf 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인 KUBELET_SYSTEM_PODS_ARGS 내 --protect-kernel-defaults 1-2) kubelet config 파일 사용하는 경우 : /var/lib/kubelet/config.yaml 파일 내 아래 파라미터 존재 및 적절한 인자 값 설정 여부 확인 protectKernelDefaults		
설정 방법	**- default Kernel 파라미터 사용 금지** 1-1) kublet service 파일 사용하는 경우 : /usr/lib/systemd/system/kublet.service.d/10-kubeadm.conf 파일 내 아래와 같이 설정 KUBELET_SYSTEM_PODS_ARGS 내 --protect-kernel-defaults=true" (추가) 1-2) kubelet config 파일 사용하는 경우 : /var/lib/kubelet/config.yaml 파일 내 아래와 같이 설정 protectKernelDefaults: true ※ kubelet 서비스 재시작 필요 systemctl daemon-reload systemctl restart kubelet.service		

```
[root@worker-node1 osboxes]# cat /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf
# Note: This dropin only works with kubeadm and kubelet v1.11+
[Service]
Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kub
elet.conf"
Environment="KUBELET_CONFIG_ARGS=--config=/var/lib/kubelet/config.yaml"
Environment="KUBELET_SYSTEM_PODS_ARGS=--protect-kernel-defaults=true "
# This is a file that "kubeadm init" and "kubeadm join" generates at runtime, populating the KUBELET_KUBEADM_ARGS variable dynamica
lly
EnvironmentFile=-/var/lib/kubelet/kubeadm-flags.env
# This is a file that the user can use for overrides of the kubelet args as a last resort. Preferably, the user should use
# the .NodeRegistration.KubeletExtraArgs object in the configuration files instead. KUBELET_EXTRA_ARGS should be sourced from this
file.
EnvironmentFile=-/etc/sysconfig/kubelet
ExecStart=
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_CONFIG_ARGS $KUBELET_KUBEADM_ARGS $KUBELET_EXTRA_ARGS
```

[그림 57] kublet service 파일 내 default Kernel 파라메터 설정 확인

```
[root@k8s-master ~]# cat /var/lib/kubelet/config.yaml
address: 0.0.0.0
apiVersion: kubelet.config.k8s.io/v1beta1
authentication:
  anonymous:
    enabled: false
  webhook:
    cacheTTL: 2m0s
    enabled: true
  x509:
    clientCAFile: /etc/kubernetes/pki/ca.crt
authorization:
  mode: Webhook
  webhook:
    cacheAuthorizedTTL: 5m0s
    cacheUnauthorizedTTL: 30s
protectKernelDefaults: true
cgroupDriver: cgroupfs
cgroupsPerQOS: true
```

[그림 58] kublet 설정 파일 내 default Kernel 파라메터 설정 확인

비고	중기 적용(적용 시 개발자 및 운영자 협의)

7. Worker Node – Host OS

7.1. 설정 파일 권한 설정

분류	Host OS	중요도	하
항목명	설정 파일 권한 설정		
항목 설명	Kubernetes 설정 파일의 퍼미션이 잘못 설정 된 경우 비인가자가 다양한 방법으로 Kubernetes 설정을 변경하여 침해사고를 일으킬 가능성이 있습니다. 따라서, root 사용자/그룹이 소유권을 가지고 있고 root 외 다른 사용자가 이 파일을 수정할 수 없도록 파일의 권한을 제한하여 파일의 무결성을 유지하여야 합니다.		

진단 방법	**[진단예시]**
	- 설정 파일 권한 설정
	: 파일 확인 후, 소유권 및 그룹 권한 확인
	`# kublet.conf 파일`
	`stat -c %a /etc/kubernetes/kubelet.conf`
	`stat -c %U:%G /etc/kubernetes/kubelet.conf`
	
	`# kublet 서비스`
	`stat -c %a /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf`
	`stat -c %U:%G /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf`
	
	`# 프록시 kubeconfig 파일`
	`stat -c %a 〈proxy kubeconfig file〉`
	`stat -c %U:%G 〈proxy kubeconfig file〉`
	
	`# kubelet 설정 파일`
	`stat -c %a /var/lib/kubelet/config.yaml`
	`stat -c %U:%G /var/lib/kubelet/config.yaml`		

설정 방법	**- 설정 파일 권한 설정**
	: 사용자 및 그룹 권한 적용
	`# kublet.conf 파일`
	`chmod 644 /etc/kubernetes/kubelet.conf`
	`chown root:root /etc/kubernetes/kubelet.conf`
	
	`# kublet 서비스`
	`chmod 755 /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf`		

```
[root@worker-node1 osboxes]#  stat -c %a /etc/kubernetes/kubelet.conf
600
[root@worker-node1 osboxes]# stat -c %U:%G /etc/kubernetes/kubelet.conf
root:root
[root@worker-node1 osboxes]# stat -c %a /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf
644
[root@worker-node1 osboxes]# stat -c %U:%G /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf
root:root
[root@worker-node1 osboxes]# stat -c %a /var/lib/kubelet/config.yaml
644
[root@worker-node1 osboxes]# stat -c %U:%G /var/lib/kubelet/config.yaml
root:root
```

[그림 59] kublet 주요 파일 권한 확인

	chown root:root /usr/lib/systemd/system/kubelet.service.d/10-kubeadm.conf # 프록시 kubeconfig 파일 chmod 644 <proxy kubeconfig 파일> chown root:root <proxy kubeconfig 파일> # kubelet 설정 파일 chown root:root /etc/kubernetes/kubelet.conf chmod 644 /var/lib/kubelet/config.yaml
비고	중기 적용(적용 시 개발자 및 운영자 협의)

7.2. 인증서 파일 권한 설정

분류	Host OS	중요도	하
항목명	인증서 파일 권한 설정		
항목 설명	Kubernetes 는 SSL/TLS 구성을 통한 네트워크상 DATA 보호 및 사용자 인증을 위해 많은 인증서를 사용합니다. 해당 인증서와 인증서가 포함된 디렉터리가 root 사용자/그룹이 소유권을 가지고 있고 root 외 다른 사용자가 이 파일 및 디렉터리에 접근할 수 없도록 파일의 권한을 제한하여 파일의 무결성을 유지하여야 합니다.		
진단 방법	**[진단예시]** **- 인증서 파일 권한 설정** 1) ps -ef | grep kubelet 실행 후 -client-ca-file 인자 사용하는 설정 파일 확인 2) 해당 파일 소유권 및 그룹 권한 확인 stat -c %a 〈client ca인증서 파일〉 stat -c %U:%G 〈client ca인증서 파일〉		
설정 방법	**- 설정파일 권한 설정** : 사용자 및 그룹 권한 적용 chmod 644 〈client ca인증서 파일〉 chown root:root 〈client ca인증서 파일〉		
비고	중기 적용(적용 시 개발자 및 운영자 협의)		

8. 기타

8.1. 네트워크 정책 설정

분류	기타	중요도	N/A
항목명	네트워크 정책 설정		
항목 설명	**- 네트워크 격리 운영** 조직에서 컨테이너를 통해 서비스를 운영하는 경우 때에 따라 외부에 오픈된 웹 서비스, 내부 관리자 서비스와 같이 민감도 수준이 다른 기능을 수행하는 컨테이너들이 존재할 수 있으며, 컨테이너 구축 전 각각 서비스하고자 하는 바를 파악 후 별도 구분하여 서비스 특성에 따라 네트워크 정책 수립 후 격리 운영하는 것을 권고드립니다. **- 일반 트래픽과 관리 트래픽을 분리** 컨테이너의 데이터 트래픽과 클러스터 관리 트래픽을 분리하여 구성할 경우 각각의 특성에 맞게 개별적으로 모니터링 될 수 있으며 다양한 트래픽 제어 정책 및 모니터링에 연결하여 관리하기 용이합니다.		
진단 방법	※ 해당 항목은 체크리스트에 포함하지 않음		
설정 방법	-		
비고	N/A		

8.2. 보안 패치 적용

<table>
<tr><td>분류</td><td>기타</td><td>중요도</td><td>중</td></tr>
<tr><td>항목명</td><td colspan="3">보안 패치 적용</td></tr>
<tr><td>항목
설명</td><td colspan="3">Kubernetes 소프트웨어는 보안 취약점, 제품 버그를 해결한 릴리즈를 제공하고 있습니다.

Docker 업데이트를 최신 상태로 유지하면 소프트웨어 취약성을 줄일 수 있습니다</td></tr>
<tr><td>진단
방법</td><td colspan="3">[진단예시]
- 최신 엔진 업데이트 설치 유무 확인
kubectl version</td></tr>
<tr><td>설정
방법</td><td colspan="3">1. 기간 산정해서 보안 패치 적용(정기 PM 등)
※ 업데이트로 인하여 서비스 장애 등이 발생할 수 있으므로 내부 검토 후 적용 검토

- Kubernetes 최신 Release 현황
<table><tr><th>버전</th><th>Release 일자</th><th>비고</th></tr><tr><td>v1.14.1</td><td>2019-04-18</td><td>-</td></tr></table>
[참고]
- CVE-2019-1002100 : 서비스 거부(2019 년 2 월 26 일)

2. 아래 사이트를 참고하여, 원격 exploit 취약점, 제로 데이 취약점 등 크리티컬 한 취약점 발견 시 즉시 패치 권고

※ Kubernetes Security Update Site
https://kubernetes.io/docs/reference/issues-security/issues/</td></tr>
<tr><td>비고</td><td colspan="3">장기 적용(적용 시 개발자 및 운영자 협의)</td></tr>
</table>

III. References

[1] https://github.com/docker/compliance
[2] https://docs.docker.com/engine/ (Docker Reference: engine)
[3] https://docs.docker.com/network/ (Docker Reference: network)
[4] https://docs.docker.com/storage/ (Docker Reference: storge)
[5] https://docs.docker.com/compliance/ (Docker Reference: compliance)
[6] https://docs.docker.com/compose/ (Docker Reference: compliance)
[7] https://success.docker.com/article/networking (Docker Reference: compliance)
[8] https://success.docker.com/article/docker-ee-best-practices (Docker Reference: best pratices)
[9] https://linuxcontainers.org/ko/lxc/security/ (linux LXC 보안)
[10] http://man7.org/linux/man-pages/man2/seccomp.2.html (linux seccomp)
[11] https://github.com/remotty/documents.docker.co.kr (Docker 한글문서 모음)
[12] https://aws.amazon.com/ko/docker/ (AWS-Docker)
[13] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf
(NIST 800-190 container security)
[14] https://cloud.google.com/containers/security/?hl=ko (구글 컨테이너 시큐리티)
[15] https://aws.amazon.com/ko/blogs/security/tag/docker/ (AWS 컨테이너 시큐리티)
[16] https://www.cisecurity.org/benchmark/docker/ (CIS Docker checklist)
[17] http://manpages.ubuntu.com/manpages/xenial/man1/unshare.1.html (ubuntu namespace)
[18] https://opensource.com/article/18/8/tools-container-security (open source tools for container)
[19] https://kubernetes.io/docs/tutorials/ (쿠버네티스 tutorials)
[20] https://kubernetes.io/docs/reference/ (쿠버네티스 reference))
[21] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-190.pdf
(NIST 800-190 container security)
[22] https://cloud.google.com/containers/security/?hl=ko (구글 컨테이너 시큐리티)
[23] https://www.cisecurity.org/benchmark/kubernetes/ (CIS 쿠버네티스)
[24] https://github.com/dev-sec/cis-kubernetes-benchmark (CIS 쿠버네티스 벤치마크 스크립트)
[25] https://github.com/ahmetb/kubernetes-network-policy-recipe (쿠버네티스 networkpolicy example)
[26] https://github.com/nccgroup/kube-auto-analyzer (Kubernetes Auto Analyzer)
[27] https://unofficial-kubernetes.readthedocs.io/en/latest/admin/kube-apiserver/(쿠버네티스reference)
[28] https://opensource.com/article/18/8/tools-container-security
(open source tools for container security)

EQST INSIGHT

클라우드 보안 가이드 (컨테이너 보안)
– Docker, Kubernetes

2019.06

SK infosec
경기도 성남시 분당구 판교로 277번길 23 4&5층

발행인 : EQST Group