

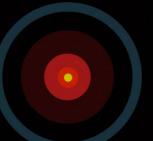
/Rooted^{CON}

Fortificando contenedores en Kubernetes
...like a boss

Rafael Troncoso
@tuxotron

Fran Ramírez
@cybercaronte

CyberHades
@cyberhadesblog



Rafael Troncoso (@tuxotron). Ingeniero de Software, con más de 20 años de experiencia trabajando para el gobierno de los EEUU en proyectos de la DoD/USN, HHS/FDA y DHS/USCIS. Actualmente continúa esa colaboración trabajando como Security Champion en **SAP**.

Co-fundador y escritor del blog de temas geek **www.cyberhades.com** (nick: tuxotron) donde hablamos de Seguridad Informática e Historia de la Informática.

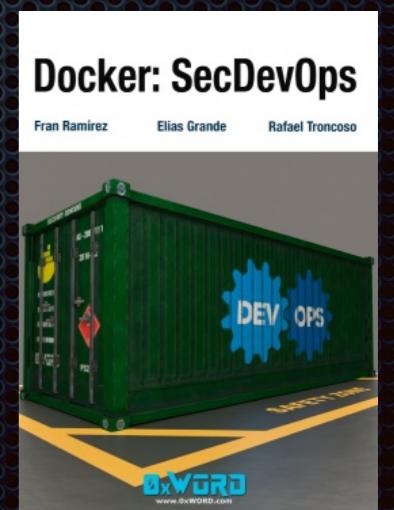
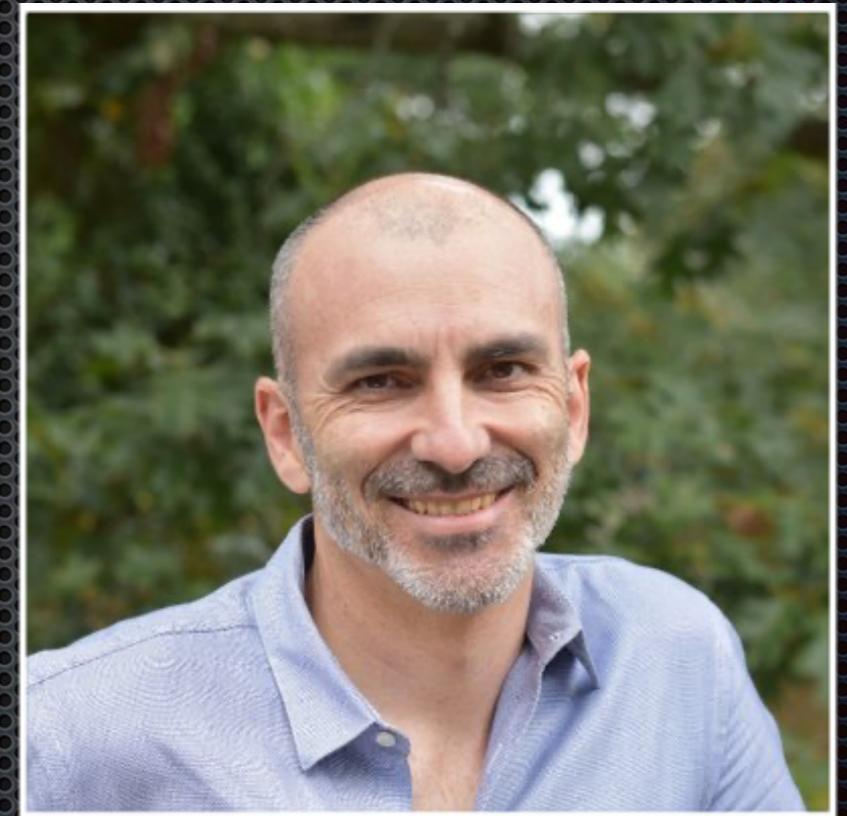
Co-autor de los libros **0xWORD**

- **Microhistorias: Anécdotas y curiosidades de la Informática**
- **Docker: SecDevOps**



My Public
Inbox

<https://www.mypublicinbox.com/RafaelTroncoso>



Fran Ramírez (@cybercaronte), Ingeniero en Informática de Sistemas, Técnico Superior en Electrónica Digital y Máster en Seguridad de las TIC, con más de 15 años de experiencia como administrador de sistemas, realizando múltiples proyectos internacionales en EEUU y Canadá. Actualmente, investigador de seguridad informática en el equipo de **Ideas Locas CDCO** de Telefónica. Colaborador de LUCA (IA, BigData) y ElevenPaths.

Co-fundador y escritor del blog de temas geek www.cyberhades.com (nick: cybercaronte) donde hablamos de Seguridad Informática e Historia de la Informática.

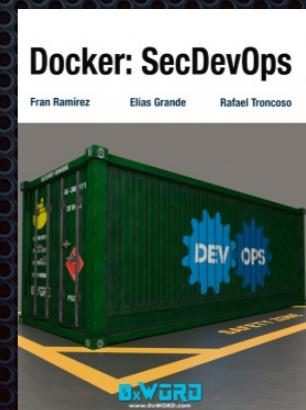
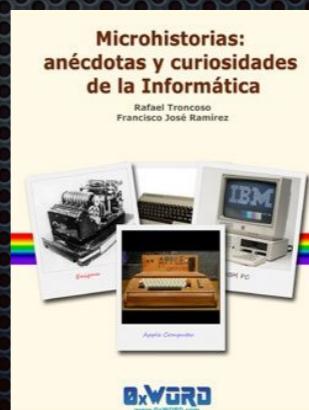
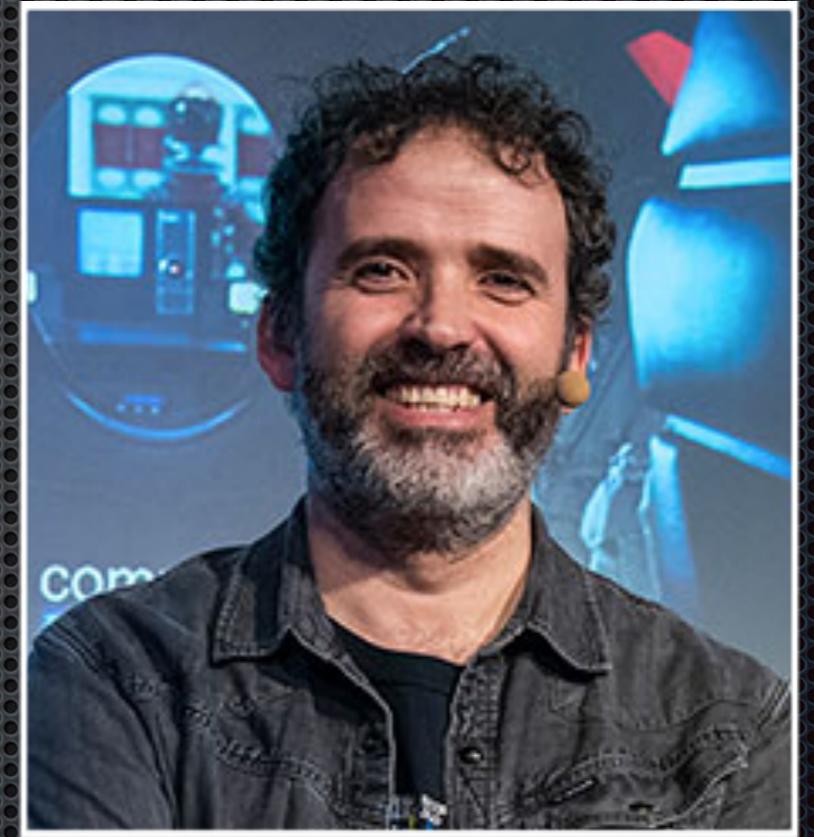
Co-autor de los libros **BxWORLD**

- **Microhistorias: Anécdotas y curiosidades de la Informática**
- **Docker: SecDevOps**
- **Machine Learning aplicado a la Ciberseguridad**



My Public
Inbox

<https://www.mypublicinbox.com/FranRamirez>



Disclaimer

/Rooted®CON

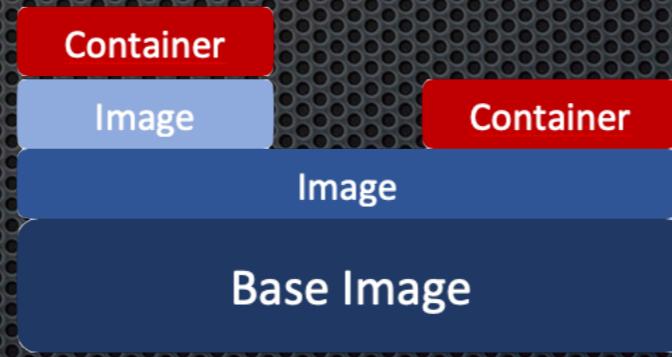
**Las opiniones, comentarios y afirmaciones realizadas
en esta charla son responsabilidad única y
exclusivamente de los ponentes.**

Imagen Base

Imagen Base

/Rooted®CON

- Cadena de provisión (supply chain)
- Escáner
- Imágenes pequeñas (por razones de seguridad)



- https://docs.docker.com/develop/develop-images/dockerfile_best-practices/

Dockerfile

/Rooted[®]CON

```
FROM golang:1.13 as build
```

```
WORKDIR /go/src/app
```

```
COPY main.go .
```

```
COPY go.mod .
```

```
RUN go get -d -v ./... && \
```

```
CGO_ENABLED=0 go install -v ./...
```

```
FROM alpine
```

```
COPY --from=build /go/bin/notes /app
```

```
COPY assets /assets/
```

```
COPY pages /pages/
```

```
RUN mkdir /notes
```

```
EXPOSE 8080
```

```
CMD ["/app"]
```

Dockerfile mejorado

/Rooted®CON

```
FROM golang:1.13 as build
WORKDIR /go/src/app
COPY main.go .
COPY go.mod .
COPY assets /assets/
COPY pages /pages/
RUN go get -d -v ./... && \
    CGO_ENABLED=0 go install -v ./... && \
    groupadd -g 1000 appgrp && \
    useradd -g 1000 -l -m -s /bin/false -u 1000 appuser && \
    mkdir /notes
RUN chown appuser /notes
FROM gcr.io/distroless/base
COPY --from=build /etc/passwd /etc/passwd
COPY --from=build /etc/group /etc/group
COPY --from=build --chown=appuser:appgrp /go/bin/notes /app
COPY --from=build --chown=appuser:appgrp /notes /notes/
COPY --from=build --chown=appuser:appgrp /assets /assets/
COPY --from=build --chown=appuser:appgrp /pages /pages/
USER 1000
EXPOSE 8080
CMD ["/app"]
```

Dockerfile mejorado

/Rooted®CON

```
FROM golang:1.13 as build
WORKDIR /go/src/app
COPY main.go .
COPY go.mod .
COPY assets /assets/
COPY pages /pages/
RUN go get -d -v ./... && \
    CGO_ENABLED=0 go install -v ./... && \
    groupadd -g 1000 appgrp && \
    useradd -g 1000 -l -m -s /bin/false -u 1000 appuser && \
    mkdir /notes
RUN chown appuser /notes
FROM gcr.io/distroless/base
COPY --from=build /etc/passwd /etc/passwd
COPY --from=build /etc/group /etc/group
COPY --from=build --chown=appuser:appgrp /go/bin/notes /app
COPY --from=build --chown=appuser:appgrp /notes /notes/
COPY --from=build --chown=appuser:appgrp /assets /assets/
COPY --from=build --chown=appuser:appgrp /pages /pages/
USER 1000
EXPOSE 8080
CMD ["/app"]
```

Dockerfile mejorado

/Rooted®CON

```
FROM golang:1.13 as build
WORKDIR /go/src/app
COPY main.go .
COPY go.mod .
COPY assets /assets/
COPY pages /pages/
RUN go get -d -v ./... && \
    CGO_ENABLED=0 go install -v ./... && \
    groupadd -g 1000 appgrp && \
    useradd -g 1000 -l -m -s /bin/false -u 1000 appuser && \
    mkdir /notes
RUN chown appuser /notes
FROM gcr.io/distroless/base
COPY --from=build /etc/passwd /etc/passwd
COPY --from=build /etc/group /etc/group
COPY --from=build --chown=appuser:appgrp /go/bin/notes /app
COPY --from=build --chown=appuser:appgrp /notes /notes/
COPY --from=build --chown=appuser:appgrp /assets /assets/
COPY --from=build --chown=appuser:appgrp /pages /pages/
USER 1000
EXPOSE 8080
CMD ["/app"]
```

Dockerfile mejorado

/Rooted®CON

```
FROM golang:1.13 as build
WORKDIR /go/src/app
COPY main.go .
COPY go.mod .
COPY assets /assets/
COPY pages /pages/
RUN go get -d -v ./... && \
    CGO_ENABLED=0 go install -v ./... && \
    groupadd -g 1000 appgrp && \
    useradd -g 1000 -l -m -s /bin/false -u 1000 appuser && \
    mkdir /notes
RUN chown appuser /notes
FROM gcr.io/distroless/base
COPY --from=build /etc/passwd /etc/passwd
COPY --from=build /etc/group /etc/group
COPY --from=build --chown=appuser:appgrp /go/bin/notes /app
COPY --from=build --chown=appuser:appgrp /notes /notes/
COPY --from=build --chown=appuser:appgrp /assets /assets/
COPY --from=build --chown=appuser:appgrp /pages /pages/
USER 1000
EXPOSE 8080
CMD ["/app"]
```

Dockerfile mejorado

/Rooted®CON

```
FROM golang:1.13 as build
WORKDIR /go/src/app
COPY main.go .
COPY go.mod .
COPY assets /assets/
COPY pages /pages/
RUN go get -d -v ./... && \
    CGO_ENABLED=0 go install -v ./... && \
    groupadd -g 1000 appgrp && \
    useradd -g 1000 -l -m -s /bin/false -u 1000 appuser && \
    mkdir /notes
```

```
RUN chown appuser /notes
```

```
FROM gcr.io/distroless/base
COPY --from=build /etc/passwd /etc/passwd
COPY --from=build /etc/group /etc/group
COPY --from=build --chown=appuser:appgrp /go/bin/notes /app
COPY --from=build --chown=appuser:appgrp /notes /notes/
COPY --from=build --chown=appuser:appgrp /assets /assets/
COPY --from=build --chown=appuser:appgrp /pages /pages/
```

```
USER 1000
```

```
EXPOSE 8080
```

```
CMD ["/app"]
```

Contexto de seguridad de Pods

Contexto seguridad Pods

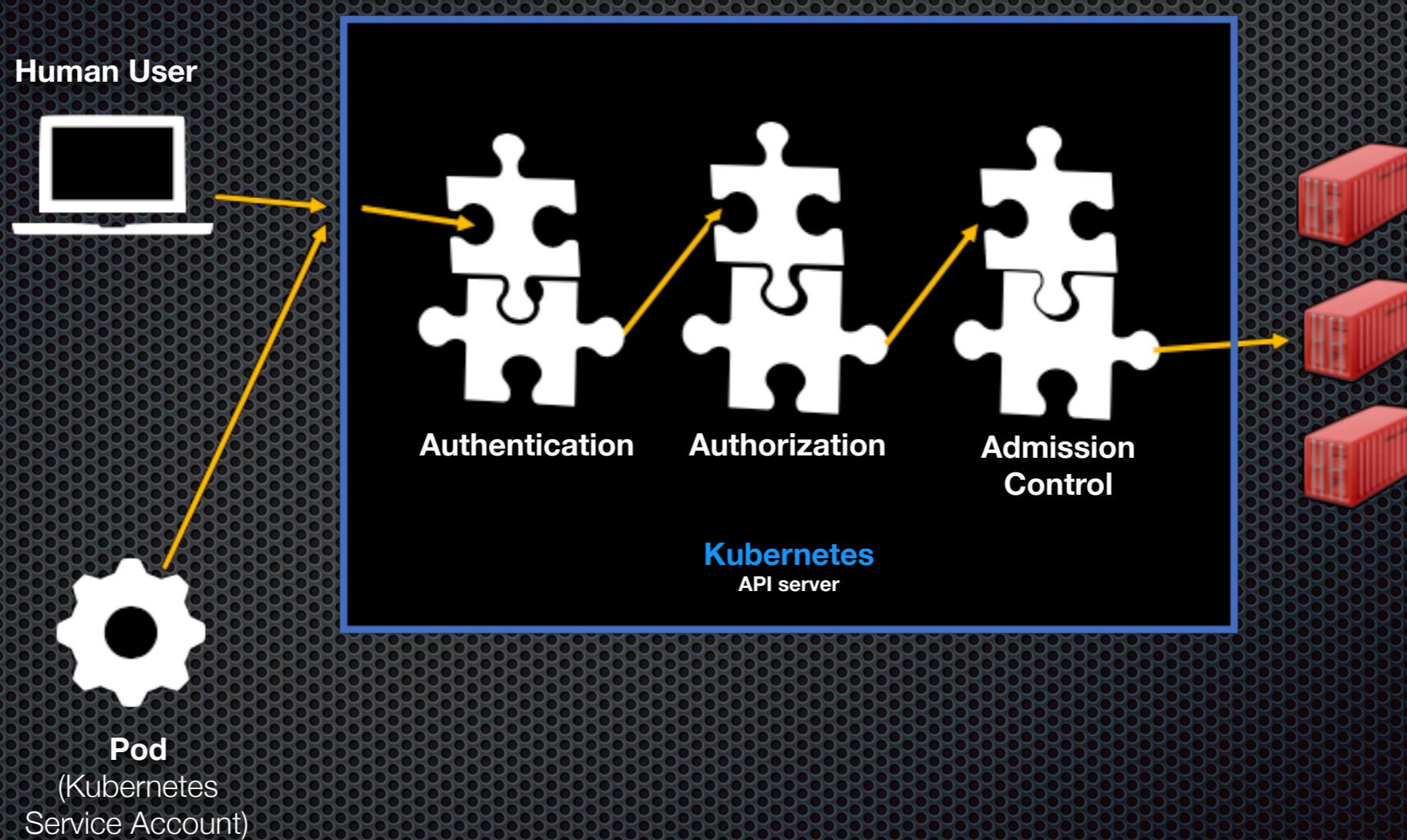
/Rooted[®] CON

- No *privilegios*
- No *root*
 - *RunAsNonRoot* vs *RunsAsUser/RunAsGroup*
- No *escalada de privilegios*
- No *escritura en sistema de ficheros*
- No *capacidades*

Contexto seguridad Pods

Control de Acceso

/Rooted[®] CON



Contexto seguridad Pods

Modo privilegiado

/Rooted[®] CON

- Permite compartir espacio de nombres del *host*
- Permite montar el sistema de ficheros del *host*
- Permite manipular la pila de red del *host*
- Por defecto es falso
- Evade *cgroups*
- Prácticamente el contenedor tiene los mismos privilegios que cualquier otro proceso en el *host*

```
securityContext:  
  allowPrivilegeEscalation: false  
  readOnlyRootFilesystem: true  
  privileged: false  
  runAsUser: 1000
```

Contexto seguridad Pods

/Rooted[®] CON

No root

- **runAsNonRoot**

- Busca USER en la imagen del contenedor
- Si USER no existe o igual a 0 (*root*) no arranca
- Si USER existe, arranca el contenedor con dicho UID y GID
- Si USER ID no existe, UID = USER ID, **GID = 0**

USER 10000

securityContext:
RunAsNonRoot: true

```
→ kubectl exec -it alpine-6df968d877-xkqk4 ash
/ $ id
uid=10000 gid=0(root)
```

Contexto seguridad Pods

/Rooted[®] CON

No root

- **runAsUser / runAsGroup**

- Especifica explícitamente el UID y GID
- Si UID no existe, GID = 0

```
securityContext:  
  runAsUser: 406
```

```
/ $ id  
uid=406 gid=0(root)
```

- Especifica ambos y evita la falsa sensación de seguridad

```
securityContext:  
  runAsUser: 1000  
  runAsGroup: 1000
```

Contexto seguridad Pods

Escalada de privilegios

/Rooted[®] CON

- Evita que un contenedor obtenga nuevos privilegios
- Ejecutables **SETUID**, **SUDO**, etc

```
securityContext:  
  allowPrivilegeEscalation: false
```

Contexto seguridad Pods

Sólo lectura

/Rooted[®] CON

- Los contenedores deberían ser inmutables
- No permitas la escritura en sistema de ficheros
- Ficheros temporales en volúmenes

```
volumeMounts:  
  - name: notes  
    mountPath: /notes  
securityContext:  
  readOnlyRootFilesystem: true
```

```
volumes:  
  - name: notes  
    emptyDir: {}
```

Contexto seguridad Pods

Capacidades

/Rooted® CON

- Sistema granular de permisos (núcleo Linux)
- Contenedores Docker corren por defecto con varias
- Muchas otras disponibles

securityContext:

capabilities:

drop:

- ALL

```
/ # id  
uid=0(root) gid=0(root) groups=0(root),1(bin),  
/ # ping 127.0.0.1  
PING 127.0.0.1 (127.0.0.1): 56 data bytes  
ping: permission denied (are you root?)
```

securityContext:

drop:

- ALL

add:

- NET_RAW

```
/ # id  
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon)  
/ # ping 127.0.0.1  
PING 127.0.0.1 (127.0.0.1): 56 data bytes  
64 bytes from 127.0.0.1: seq=0 ttl=64 time=0.047 ms  
64 bytes from 127.0.0.1: seq=1 ttl=64 time=0.059 ms
```

Cuentas de servicio

Cuentas de servicio

/Rooted®CON

- Gestionadas por *Kubernetes*
- Cada *pod* tiene asociada una cuenta de servicio
- Credenciales de la cuenta (*token*) se monta por defecto:
 - **/var/run/secrets/kubernetes.io/serviceaccount/token**

```
/run/secrets/kubernetes.io/serviceaccount # cat token
eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhYiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC5uYW1LI1YWNjb3VudDpkZWZhdWx0OmR1ZmF1bHQifQ.grmNamXKrMgt4yyYggsvf3pYgF0_JFqpKcWYY7YNyTkVcpMAvS8-NZGytsfm-cbEg-Jp7KCbthErQi2-bfioDXfy9D3yuPuHB-5ADO
```

```
curl -k https://192.168.99.108:8443/api/v1 -H "Authorization: Bearer eyJhbGci..."
```

- Evita montar la credenciales si es posible
 - A nivel de *pod* o cuenta de servicio

```
spec:
  automountServiceAccountToken: false
```

Redes en Kubernetes

Redes en Kubernetes

/Rooted®CON

- Existen 4 problemas de comunicación
 - Contenedor a contenedor
 - Pod a pod
 - Servicio a pod
 - Externo a servicio



Redes en Kubernetes

Comunicación entre *pods*

/Rooted® CON

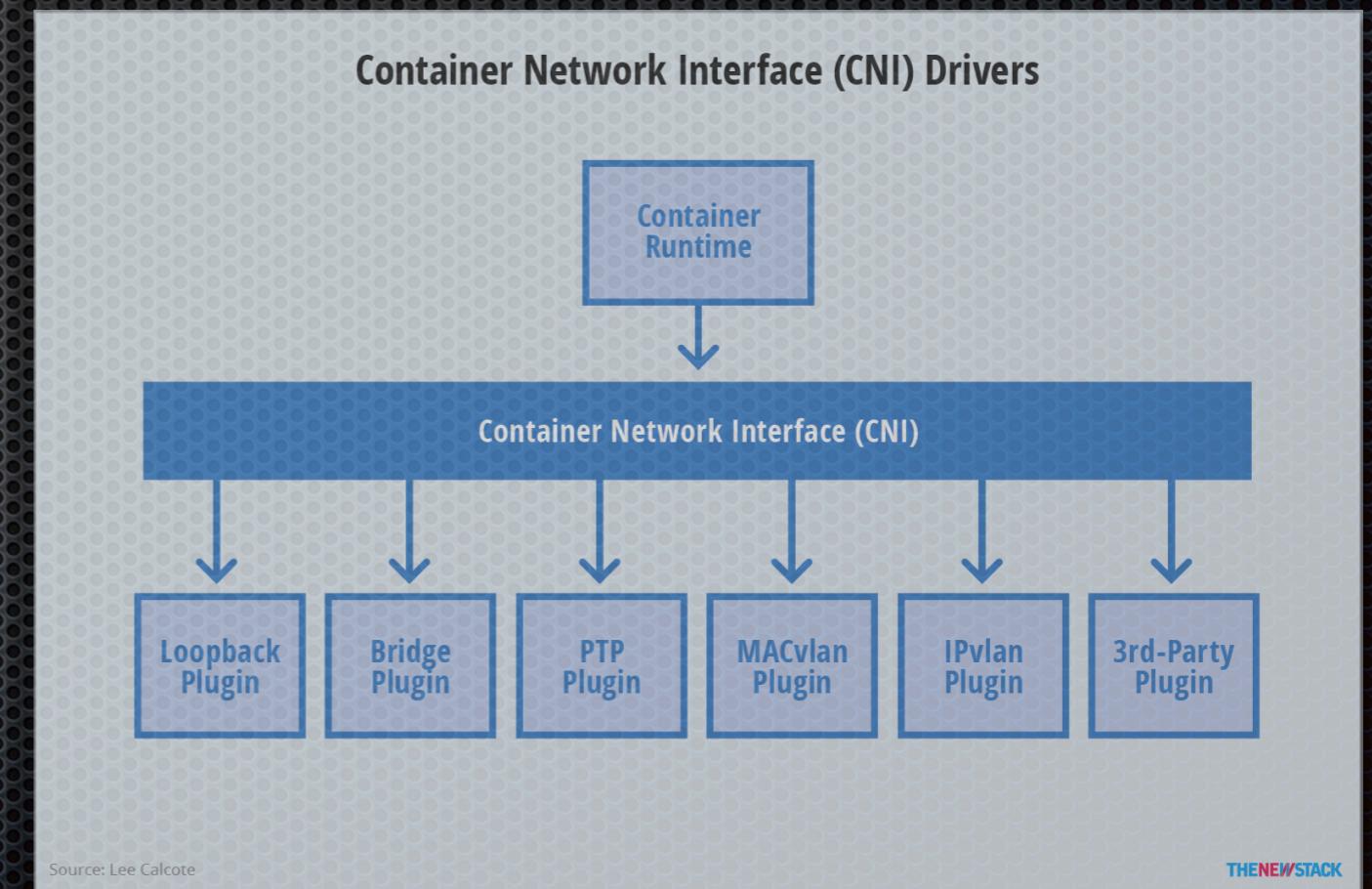
- Los *pods* en un nodo, deben de poder comunicarse con todos los *pods* en todos los nodos **sin NAT**
- Agentes en un nodo (*system daemons*, *kubelet*), deben de poder comunicarse con todos los *pods* de ese mismo nodo
- *Pods* en la red de *host*, deben poder comunicarse con todos los *pods* en todos los nodos **sin NAT**
- Todos los *pods*, **por defecto**, se pueden comunicar entre ellos a través de nombres de espacio o nodos

Redes en Kubernetes

/Rooted®CON

CNI (Container Network Interface)

- Manejan la red de **pod**s
- No todos soportan políticas de red
 - **AWS_CNI**
 - ...
- Otros sí:
 - **Calico**
 - **Cilium**
 - ...



Políticas de red

Políticas de red

/Rooted®CON

- Pertenecen a un nombre de espacio
- Basadas en selectores
- No existen políticas de red por defecto
- Una vez un pod sea afectado por una política de red, ésta se aplica y el resto es denegado
- Son aditivas, no conflictos
- **No es un firewall**



kubernetes
Network Policies

Política de red

Ejemplo

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
    - Ingress
    - Egress
```

```
ingress:
- from:
  - ipBlock:
    cidr: 172.17.0.0/16
    except:
      - 172.17.1.0/24
  - namespaceSelector:
    matchLabels:
      project: myproject
  - podSelector:
    matchLabels:
      role: frontend
  ports:
    - protocol: TCP
      port: 6379
```

```
egress:
- to:
  - ipBlock:
    cidr: 10.0.0.0/24
  ports:
    - protocol: TCP
      port: 5978
```

Políticas de red

Buenas prácticas

/Rooted® CON

- Deniega todo el tráfico por defecto

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  podSelector: {}
  policyTypes:
    - Ingress
    - Egress
```

- Abre el tráfico que necesites
- Etiqueta los *pods* y nombre de espacios

Fortificación (hardening)

Fortificación

Despliegue por defecto vs fortificado

/Rooted[®]CON

```
spec:  
  containers:  
    - name: notes  
      image: notes:v1  
      ports:  
        - containerPort: 8080
```

VS

```
spec:  
  automountServiceAccountToken: false  
  containers:  
    - name: notes  
      image: notes:v2  
      volumeMounts:  
        - name: notes  
          mountPath: /notes  
      securityContext:  
        allowPrivilegeEscalation: false  
        readOnlyRootFilesystem: true  
        privileged: false  
        runAsUser: 1000  
        runAsGroup: 1000  
      capabilities:  
        drop:  
          - ALL
```

Hace 30 minutos...

/Rooted[®]CON

- Explotamos la aplicación consiguiendo:
- Ver contenido de directorios
- Ver contenido de ficheros
- Sobreescribir contenido de ficheros del sistema
- Exfiltrar token de cuenta de servicio de **Kubernetes**
- **RCE**
- Descarga de contenido



Demo

- Fortificar contenedores no arregla vulnerabilidades en la aplicación
- ¡Arregla la aplicación!



- No ayuda con todas las vulnerabilidades (SQLi, client side, etc)
- Ayuda contener y restringir un atacante
- Docker y Kubernetes por defecto sacrifican la seguridad por la facilidad de uso

¿Qué vas a hacer ~~s~~^Xcuando
tu aplicación ~~e~~^Xs sea
comprometida?



/Rooted° CON Gracias



<https://github.com/cyberhades/hardening-k8s-containers>

Rafael Troncoso
@tuxotron

Fran Ramírez
@cybercaronte

CyberHades
@cyberhadesblog

