P8. Asegurando contenedores

A lo largo del curso **no hemos tenido en cuenta para nada la seguridad** pero sin embargo hemos estado trabajando y hemos realizado operaciones como las siguientes:

- Descargar **imágenes de terceros** sin mirar quién era el autor.
- **Entrar como root en el contenedor** sin ningún tipo de problema para instalar y configurar.
- **Ejecutar y publicar servicios como root.**
- **Incluir usuarios y contraseñas en Dockerfiles y ficheros docker-compose.**

Gangster8192 en WikimediaCommons. *Icono de seguridad* (Dominio público)

Si bien es cierto que esto es un curso más orientado a la docencia y al desarrollo, también es cierto que si en algún momento queremos desplegar nuestro código en forma de contenedor en un orquestador como Kubernetes o Docker Swarm tendremos que tomar ciertas medidas. La seguridad es un aspecto global en todas las organizaciones y la seguridad en los contenedores es uno más de la multitud de factores que debemos de tener en cuenta.

En este curso hablaremos de la seguridad en los contenedores desde tres perspectivas:

- **La seguridad en las imágenes docker.**
- **La seguridad en el proceso de build.**
- **La seguridad al arrancar los contenedores.**

8.1 Seguridad en las imágenes

A lo largo de todos los capítulos hemos descargado multitud de imágenes desde DockerHub. Estas **imágenes son el elemento fundamental** sobre el que hemos desarrollado todo el curso pero si no tenemos cuidado **pueden representar un posible riesgo de seguridad** ya que, normalmente son **imágenes realizadas por terceros** y, por lo tanto, no tenemos ningún control sobre ellas.

Este aspecto no es tan importante si las usamos únicamente para el desarrollo pero adquiere mucha más relevancia si tenemos pensado llevar esos contenedores a producción.

Para intentar minimizar los riesgos de usar imágenes hay que seguir una serie de pautas generales:

- Intentar siempre usar una **imagen de un usuario verificado o una imagen publicada** por la propia empresa Docker.
- **Verificar la integridad de la imagen que nos descargamos.**
- Intentar usar **imágenes que tengan lo mínimo.**
- **Restringir los privilegios** de los datos a los que pueden acceder los contenedores.
- En caso de crear nuestras propias imágenes **pagar por el servicio** de DockerHub ***"Vulnerability Scanner"*** o utilizar herramientas que sirve para ese mismo propósito.

USUARIOS VERIFICADOS E IMÁGENES OFICIALES DE LA EMPRESA DOCKER

Aunque no es una garantía al 100%, el hecho de descargar este tipo de imágenes ayuda a reducir los posibles riesgos que puedan existir. Para obtener este tipo de imágenes, cuando hacemos una búsqueda debemos activar los filtros adecuados:

Juan Diego Pérez Jiménez. *Búsqueda de imágenes oficiales o verificadas* (Dominio público)

VERIFICACIÓN DE LAS IMÁGENES

Docker utiliza un **mecanismo de firma digital llamado DCT (Docker's Content Trust) p**ara verificar la **INTEGRIDAD y la AUTORÍA** de las imágenes que nos descargamos desde DockerHub. Por defecto el mecanismo DCT está deshabilitado y cuando hacemos docker pull podemos descargar imágenes sin verificar ni la integridad ni la autoría.

Si habilitamos DCT **solo podré hacer PULL / RUN / BUILD con imágenes *"Trusted"* salvo** que explícitamente haga alguna excepción **añadiendo el flag ---disable-content-trust** a la ejecución de dichas órdenes.

Si quiero habilitarlo tengo que poner la variable de entorno **DOCKER_CONTENT_TRUST=1.** A partir de entonces no podré descargarme imágenes no firmadas, incluso aunque sean mías.En Linux esto se realiza de la siguiente manera:

```
# Añado la siguiente línea al final del fichero .bashrc que sirve para añadir esa variable de entorno

export DOCKER_CONTENT_TRUST=1

# Recargo el .bashrc

> source /home/miusuario/.bashrc
```

IMPORTANTE: El procedimiento en Windows y Mac para establecer variables de entorno es diferente.Si no quiero crear la variable de entorno puedo añadir DOCKER_CONTENT_TRUST=1 delante de la orden docker para conseguir el mismo efecto.

FIRMA DIGITAL DE MIS IMÁGENES

Si quiero firmar digitalmente mis imágenes tengo que seguir los siguientes pasos:

1. **Generar la parejas de claves** público/privada. La privada se coloca en ~/.docker/trust/private
2. **Compartir mi clave pública con el servidor Notary** asociado a DockerHub. Notary es una herramienta que me permite publicar contenidos confiables asegurando la integridad de dicho contenido y la autoría del mismo. Si utilizara otro registro diferente a DockerHub el hecho de tener un servicio Notary es requisito para poder firmar las imágenes en ese registro.**TENGO QUE GENERAR UNA CLAVE PÚBLICA PARA CADA REPOSITORIO (Colección de versiones de la misma imagen).**
3. **Firmar la imagen.** Este proceso firma y a la vez hace un push.

```
# 1. Generar la pareja de claves

>  docker trust key generate miclave.pub

# 2. Comparto mi clave pública con el servidor Notary en el repositorio para las imágenes del respositorio usuario/nombreimagen. En nombre del firmante es miclave y miclave.pub es la clave pública generada en el proceso anterior. Recordad que un mismo repositorio puede contener varias versiones (TAGS) de una misma imagen.

> docker trust signer add --key miclave.pub miclave usuario/nombrerepositorio

# 3. Firmo la imagen
```

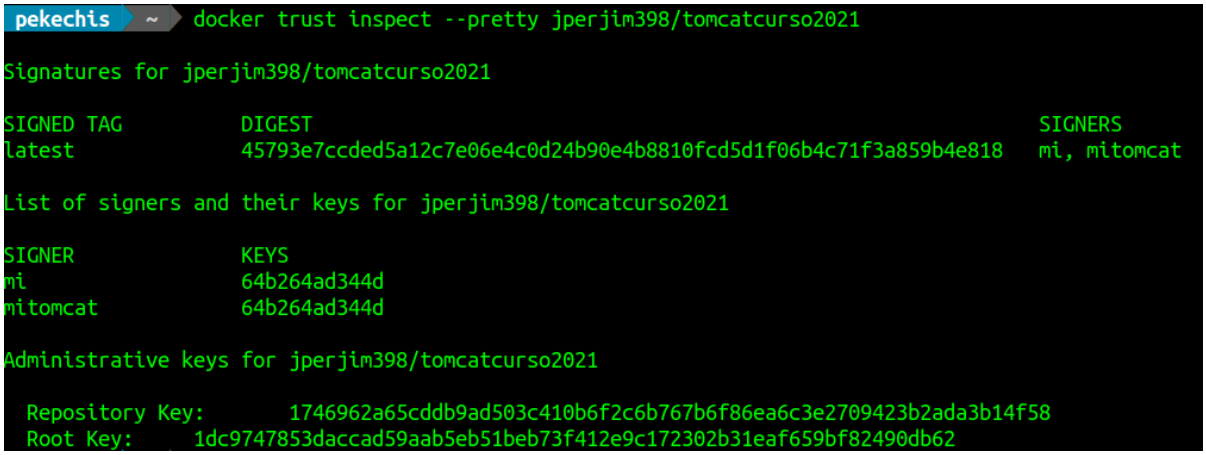
```
> docker trust sign usuario/nombreimagen[:tag]
```

Si quiero ver las firmas almacenadas para un determinado repositorio debo ejecutar la siguiente orden:

```
# 1. Obtener las firmantes y las claves de un repositorio

>  docker trust inspect --pretty nombreusuario/nombrerepositorio
```

Obtendremos una salida similar a la siguiente:



Juan Diego Pérez Jiménez. *Salida de la orden docker trust inspect* (Dominio público)

Para más opciones de la orden docker trust os dejo el enlace directo a la referencia: https://docs.docker.com/engine/reference/commandline/trust/

IMÁGENES MíNIMAS

Cuando estamos construyendo imágenes debemos intentar que sean imágenes de un tamaño mínimo. **¿Qué sentido tiene usar un sistema operativo entero si solo queremos instalar una aplicación o un servicio?.**

Cada aplicación nueva que instalamos añade un potencial punto de inseguridad y nos obliga a estar atentos para actualizar a una nueva versión cuando surja una vulnerabilidad.

Dos buenas prácticas en este sentido son:

- **Si estamos construyendo nuestra propia imagen** usaremos una imagen de base que sea de tamaño mínimo. Un ejemplo es bitnami/minideb. . También es cierto que a lo largo del curso hemos utilizado imágenes de terceros con un único servicio ya instalado. Normalmente estas imágenes ya están pulidas y no tienen mas que lo necesario.
- Usar el fichero ***.dockerignore*** en nuestro proceso de build. Así evitamos que nuestra imagen tenga más tamaño del necesario.

RESTRICCIÓN DE PRIVILEGIOS

Cuando hablamos de restricción de privilegios nos referimos a los **permisos que el contenedor va a tener en el acceso** a los volúmenes montados, bind mounts o a ciertos directorios del propio contenedor.

Si vamos a usar un volumen o un bind mount en el que no queremos que se pueda escribir desde el contenedor podemos hacerlo añadiendo la opción readonly al flag --mount. Por ejemplo:

```
# Realizo un bind mount evitando que desde el contenedor se pueda escribir en mi carpeta:

>  docker run -it --mount type=bind,src=/home/pekechis/pruebaPHP,dst=/var/www/html,readonly -p 8686:80 php:7.4-apache
```

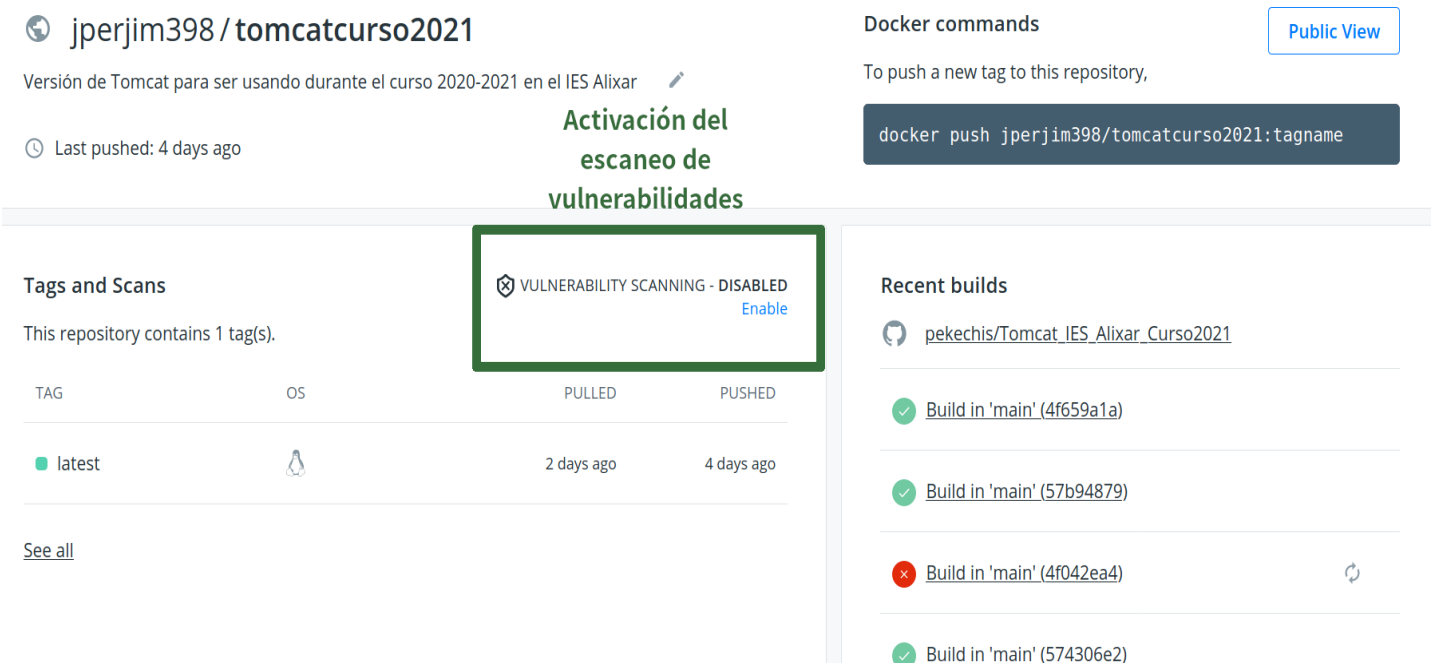
Pero si por el contrario quiero evitar que se escriba en una carpeta propia del contenedor tengo que usar el flag --read-only de la orden docker run. El contenedor no podrá realizar ningún tipo de escritura en su sistema de ficheros. Por ejemplo:

```
# Arranco un contenedor en modo solo lectura.

>  docker run -it --read-only ubuntu:20.04 /bin/bash
```

SERVICIO "VULNERABILITY SCANNER"

Para cuentas de pago DockerHub proporciona un servicio que **escanea las imágenes para encontrar vulnerabilidades** tanto a nivel de sistema operativo como a nivel de las posibles aplicaciones que tengamos instaladas en dichas imágenes. Para activar este servicio tenemos que cambiar de plan y aplicarlo a cada una de las imágenes que queramos escanear desde la sección de administración del repositorio de la imagen:



Juan Diego Pérez Jiménez. *Habilitar el escaneo de vulnerabilidades* (Dominio público)

8.2 Seguridad en el proceso de build

Cuando estamos construyendo nuestras propias imágenes también debemos de tener en consideración ciertos factores a la hora de mejorar la seguridad de las imágenes resultantes. Algunos de los más básicos hacen referencia a las siguientes órdenes del Dockerfile

- **ADD & COPY**
- **CMD & ENTRYPOINT**

ADD & COPY

Como ya vimos en el capítulo 6 **la orden ADD** funciona como una **copia recursiva** desde el origen hasta la carpeta destino en el contenedor, creando los directorios en caso de que no existan. Esta copia recursiva puede ser desde un contenido local o desde una URL y además si le pasamos un archivo comprimido a la orden ADD los descomprime de manera automática. **Pero en caso de que este contenido venga de una URL:**

- **¿Confiamos realmente en los autores?**
- **¿Cómo sabemos que los contenidos no han sido comprometidos a nivel de seguridad en el origen?**
- **¿Nos hemos protegido frente a ataques Man-In-The-Middle?.**

Todo estos aspectos deben de ser tenidos en cuenta.

Al igual que ADD, COPY hace una copia recursiva desde el origen hasta el destino pero no permite ni URL de origen ni descomprime los ficheros. Si copiamos un archivo .zip este permanecerá como un archivo .zip. Esta vez **el problema reside en ambos casos en la recursividad de la copia.**

Cuando copiamos elementos de forma recursiva incrementamos la posibilidad de que suceda lo siguiente:

- **Aumentar el tamaño de la imagen** de manera innecesaria, tal y como dijimos anteriormente.
- **Copiar en el contenedor ficheros sensibles** que contengan claves, tokens de APis etc...

En ambos casos tenemos que tener un **fichero .dockerignore** debidamente configurado y que excluya explícitamente del proceso de copia recursiva archivos con claves como *.ENV, *.pem etc..

El contenido del fichero .dockerignore funciona de manera **análoga al fichero .gitignore**, tal y como hemos comentado a lo largo del curso. Una descripción más detallada de los contenidos que podemos escribir en este archivo la podéis encontrar aquí.

CMD & ENTRYPOINT

Como norma general **los procesos que se ejecutan en un contenedor** (al igual que en otros sistemas) **no deben ejecutarse como root**, en especial aquellos que son servicios expuestos al exterior.

¿Por qué es importante esto?

Desde el punto de vista de seguridad una buena práctica es que las aplicaciones únicamente tengan acceso a los recursos que necesitan para desempeñar sus funciones (PRINCIPIO DEL MÍNIMO PRIVILEGIO). Si no lo hacemos así y ejecutamos aplicaciones como root esa aplicación podrá realizar cualquier operación en nuestro sistema, por ejemplo:

- Realizar operaciones no permitidas como borrar ficheros, parar servicios y transferir información.
- Acceder a ficheros con contenido sensible que hayamos podido colocar en el contenedor, por ejemplo usando un bind mount.

Debemos de tener en cuenta que estamos usando imágenes de terceros y confiando en las aplicaciones que vienen en dicha imágenes. Debemos confiar, pero siempre ser cautos y comprobar si esa aplicación que vamos a usar está actualizada desde el punto de vista de la seguridad.

¿ Cómo evito esto en mi Dockerfile?

Puedo afrontar este tipo de problemas desde un Dockerfile siguiendo este flujo de trabajo cuando lo estoy creando:

1. Crear un usuario y un grupo para ese usuario (RUN).
2. Ejecutar todas las instrucciones del Dockerfile que tengan que ser realizadas como root (RUN, COPY, ADD,WORKDIR etc....).
3. De manera previa a ejecutar el ENTRYPOINT y/o el CMD cambiar el usuario de ejecución de las órdenes al usuario creado previamente (USER).
4. Definir el ENTRYPOINT y/o CMD y que esa órdenes lances procesos pertenecientes al usuario creado.

Un ejemplo de esta estrategia:

```
# Imágen que vamos a usar

FROM XXXXX

....

# Creación del usuario para arrancar el servicio

RUN addgroup -S usuario && adduser -S usuario -G usuario.

.........

# Lista de órdenes que serán de ROOT

.........

# Establezco el usuario que ejecutará las siguientes órdenes.

USER usuario

# Defino el ENTRYPOINT, se ejecutará como usuario.

ENTRYPOINT ..…
```

8.3 Seguridad al arrancar los contenedores

Si queremos aumentar la seguridad al desplegar los contenedores podemos actuar en dos sentidos:

- **Limitar los recursos** que puede usar el contenedor. Esto será de especial utilidad para evitar ataques de DoS.
- **Deshabilitar "capabilities"** del contenedor que voy a arrancar.

LIMITAR RECURSOS

Para limitar recursos hay un serie de flags de docker run que podemos usar, entre ellos los más destacados son:

- **--memory/-m** que establece el límite de memoria que puede llegar a usar un contenedor.
- **--memory-reservation** que es similar al anterior pero es un límite blando. Si se sobrepasa docker intentará reducir la memoria consumida por el contenedor.
- **--cpus** que limita el número de cpus del sistema que un contenedor puede utilizar.
- ....

Por ejemplo:

```
# Arranco un contenedor con un límite de 4GB y 4cpus

> docker runt -it -m 4Gb --cpus=4 httpd .....
```

DESHABILITAR LAS CAPACIDADES

Si además queremos quitar capacidades a nuestros contenedores los haremos con el flag **--cap-drop**, el contrario al flag --cap-add que vimos en el módulo de redes para habilitar iptables usando la capacidad NET_ADMIN.

La lista de las capacidades de los sistemas Linux es larga. Podemos encontrarla en la siguiente dirección: **https://man7.org/linux/man-pages/man7/capabilities.7.html**

Por citar algunas podemos destacar:

- ***AUDIT_WRITE:*** Escribe mensajes en los log de auditoría del Kernel.
- ***CHOWN:*** Para permitir cambios en el UID y GUID de los ficheros.
- ***NET_BIND_SERVICE:*** Permite asociar un socket a un puerto que puede ser accedido desde Internet.
- ***NET_ADMIN:*** Configuración de interfaces, masquerading, enrutamiento, iptables etc...
- ***SETUID:*** Manipulación del UID de los procesos etc...
- **SETGUID:** Manipulación del GUID de los procesos etc..
- ***FOWNER:*** Para manipulación de todo tipo de permisos y ACLs en ficheros.

Un ejemplo de cómo deshabilitar una de estas capacidades sería:

```
# Deshabilitar la escritura de mensajes en el log de auditoría del Kernel

> docker runt -it --cap-drop=audit_write  ubuntu:20.04
```

Habilitar o deshabilitar capacidades requiere un estudio minucioso para cada contenedor. En general debemos limitar al máximo las capacidades cuando llevemos esos contenedores a producción.