Análisis de la seguridad de Docker en servidores Linux

Rafael Cerino Frías, José Ángel Jesús Magaña, Alejandro Hernández Cadena, José Ney Garrido Vázquez, José Manuel Gómez Zea

Tecnológico Nacional de México Campus Villahermosa, Cd. Industrial. Departamento de Postgrado e Investigación. Carretera Villahermosa Frontera, K. 3.5, Cd. Industrial, Villahermosa Tabasco, CP: 86010.

Resumen

En años recientes, el uso de las tecnologías de virtualización ha tenido un espectacular aumento de uso. Esto conlleva a la demanda de soluciones de virtualización que brinden eficiencia y seguridad. En la actualidad, la virtualización tiene dos vertientes principales, la virtualización basada en contenedores y la virtualización basada en hipervisor. De estas dos ramas, la virtualización basada en contenedores puede proporcionar un entorno virtual más ligero y eficiente, pero no sin dejar de preocuparnos por la seguridad. En este artículo se analiza el nivel de seguridad de Docker. En el análisis se toma en consideración dos áreas: la seguridad de la herramienta Docker y cómo interactúa con los módulos que hacen referencia a la seguridad del kernel de Linux, como SELinux y AppArmor.

Abstract

In recent years, the use of virtualization technologies has seen a dramatic increase in use. This leads to the demand for virtualization solutions that provide efficiency and security. Today, virtualization has two main streams, container-based virtualization and hypervisor-based virtualization. Of these two branches, container-based virtualization can provide a more lightweight and efficient virtual environment, but not without security concerns. This article discusses the security level of Docker. The discussion takes two areas into consideration: the security of the Docker tool and how it interacts with modules that refer to Linux kernel security, such as SELinux and AppArmor.

Palabras clave. Docker, Contenedor Docker, Máquina virtual, Virtualización, Computación en la nube.
Keywords. Docker, Docker Container, Virtual Machine, Virtualization, Cloud Computing.

1. INTRODUCCIÓN

La última década ha sido testigo de una explosión de desarrollo en el área de las tecnologías de virtualización, que permiten la partición de un sistema informático en múltiples entornos virtuales aislados. Las tecnologías ofrecen beneficios que han impulsado su desarrollo rápidamente. Una de las razones más comunes para adoptar tecnologías de virtualización es la virtualización de servidores en los centros de datos. Con la virtualización de servidores, un administrador puede crear una o más instancias de sistema virtual en un solo servidor. Estos sistemas virtuales funcionan como servidores físicos reales y se pueden alquilar mediante suscripción, por mencionar: Amazon EC2, Microsoft Azure y Google Cloud son algunas instancias populares de estos proveedores de servicios de centros de datos. Otro uso común es la virtualización de escritorios, consiste en una computadora que puede ejecutar varias instancias del Sistema Operativo (SO). La virtualización de escritorio brinda soporte para aplicaciones que solo pueden ejecutarse en un SO específico.

El crecimiento en el uso de tecnologías de virtualización promueve la demanda de una solución de virtualización que pueda proporcionar entornos de usuario escalables y seguros. Ha aparecido en el mercado una gran cantidad de soluciones de virtualización. Se pueden clasificar en dos ramas principales: virtualización basada en contenedores y virtualización basada en hipervisor. De estas dos clases, la virtualización basada en contenedores puede proporcionar un entorno virtual más ligero y eficiente. Permite que se ejecuten diez veces más entornos virtuales en un servidor físico en comparación con la virtualización basada en hipervisor [1]. Sin embargo, la virtualización basada en contenedores también conlleva problemas de seguridad.

En este artículo, analizamos el nivel de seguridad de Docker [2], un conocido representante del enfoque de virtualización basada en contenedores. Consideramos dos áreas: la seguridad interna de Docker y cómo Docker interactúa con los módulos de seguridad del kernel de Linux, como SELinux y AppArmor, para fortalecer el sistema operativo huésped (host). El análisis examina la seguridad interna de Docker en función del nivel de aislamiento que Docker puede proporcionar a sus entornos virtuales.

2. ENFOQUES DE VIRTUALIZACIÓN

La mayoría de las tecnologías de virtualización se pueden clasificar en dos enfoques principales: virtualización basada en contenedores y virtualización basada en hipervisor. El primero proporciona virtualización a nivel de sistema operativo, mientras que el segundo proporciona virtualización a nivel de hardware. Cada uno de los enfoques tiene sus propias ventajas y desventajas, que se describen a continuación.

La virtualización basada en contenedores es un enfoque de virtualización ligero que utiliza el kernel del host para ejecutar varios entornos virtuales. Estos entornos virtuales a menudo se denominan contenedores. Linux-VServer [3], OpenVZ [4] y Linux Container (LXC) [5] son los tres principales representantes de este enfoque para un SO con kernel de Linux. La arquitectura general de una solución de virtualización basada en contenedores se muestra en la Figura 1. La virtualización basada en contenedores virtualiza en el nivel del sistema operativo, lo que permite que múltiples aplicaciones funcionen sin ejecutar de forma redundante otros núcleos del sistema operativo en el host. Sus contenedores se ven como procesos normales desde el exterior, que se ejecutan en la parte superior del kernel compartido con la máquina host. Proporcionan entornos aislados con los recursos necesarios para ejecutar aplicaciones. Estos recursos pueden compartirse con el host o instalarse por separado dentro del contenedor.

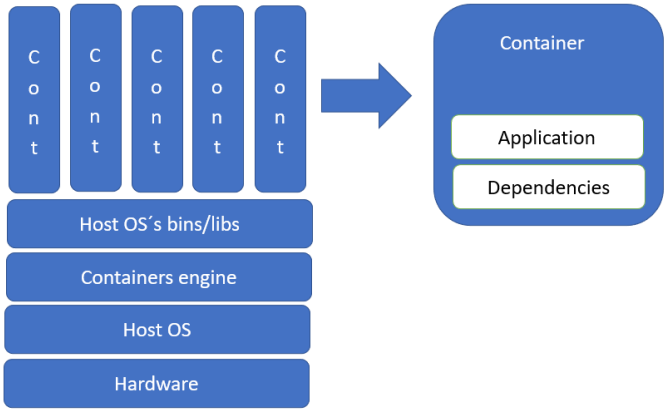

Figura 1. Arquitectura de virtualización basada en contenedores

ISSN: 2007-4786

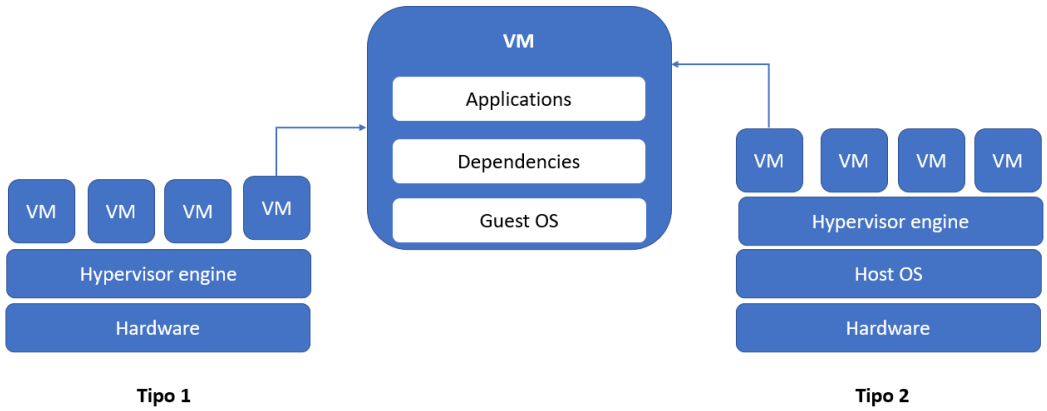

Figura 2. Arquitectura de virtualización basada en hipervisor

Las soluciones de virtualización basadas en hipervisor brindan virtualización a nivel de hardware. A diferencia de la virtualización basada en contenedores, un hipervisor establece máquinas virtuales (VM) completas en la parte superior del sistema operativo host (Figura 2). Cada máquina virtual se compone no solo de una aplicación y sus dependencias, sino también de un sistema operativo completo (invitado) junto con un kernel separado. Hay dos clases de hipervisores: el hipervisor de tipo 1, también conocido como **hipervisor bare metal**, que funciona directamente sobre el hardware subyacente del host y el hipervisor de tipo 2, también conocido como **hipervisor alojado**, que funciona en la parte superior del sistema operativo anfitrión [6]. Xen [7] es un ejemplo del primero, mientras que KVM [8] es del segundo. Dado que el hipervisor de tipo 1 no incluye una capa adicional del sistema operativo host, proporciona un mejor rendimiento que el hipervisor de tipo 2.

Las diferencias en la arquitectura aportan algunos beneficios a la virtualización basada en contenedores sobre la virtualización basada en hipervisores. En primer lugar, la virtualización basada en contenedores puede proporcionar un mayor número de entornos virtuales. Dado que un contenedor no incluye un sistema operativo completo, el tamaño y los recursos necesarios para ejecutar una aplicación en un contenedor son menores que los de una máquina virtual que ejecuta la misma aplicación. Como resultado, se pueden implementar más contenedores que las máquinas virtuales tradicionales en el mismo host. En segundo lugar, la virtualización basada en contenedores también ofrece un mejor rendimiento. Esto ha sido demostrado por experimentos en algunos estudios [9] [10] [11]. Estos estudios muestran que el rendimiento de la virtualización basada en contenedores es mejor que con la virtualización basada en hipervisor en la mayoría de los casos y, es casi tan bueno como las aplicaciones nativas.

3. DESCRIPCIÓN GENERAL DE DOCKER

Docker es una tecnología de contenedores de código abierto con la capacidad de construir, enviar y ejecutar aplicaciones distribuidas [2]. Se ha utilizado en algunas aplicaciones populares, como Spotify, Netflix y PayPal.

Aunque las tecnologías de contenedores han existido durante más de una década, Docker, un candidato relativamente nuevo, es actualmente una de las tecnologías más exitosas ya que viene con nuevas habilidades que las tecnologías anteriores no poseían. En primer lugar, proporciona interfaces para crear y controlar contenedores de forma sencilla y segura. En segundo lugar, los desarrolladores pueden empaquetar aplicaciones en contenedores Docker livianos que pueden operar en casi cualquier lugar sin modificaciones. Además, Docker puede implementar más entornos virtuales que otras tecnologías en el mismo hardware [1].

Por último, pero no menos importante, Docker coopera bien con herramientas de terceros, que simplifican el proceso de administración e implementación de los contenedores de Docker.

Las herramientas de DevOps, como Puppet, Ansible y Vagrant pueden integrarse con Docker, lo que hace que los contenedores de Docker se implementen fácilmente en una nube. Además, muchas herramientas de orquestación, como Docker Swarm, Mesosphere Datacenter y Kubernetes, también admiten contenedores Docker. Estas herramientas proporcionan una capa abstracta de administración y programación de recursos sobre Docker.

Docker consta de dos componentes principales: Docker Engine y Docker Hub. La primera es una solución de virtualización de código abierto, mientras que la segunda es una plataforma de software como servicio para compartir imágenes de Docker. Las siguientes secciones describen estos dos componentes.
Docker Engine

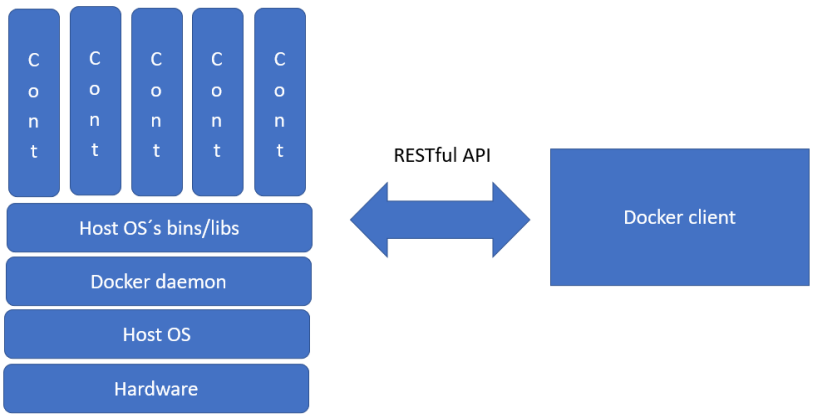

Figura 3. Arquitectura de Docker Engine

Docker Engine es una herramienta de empaquetado ligera y portátil [2] que se basa en la virtualización enfocada en contenedores. Por lo tanto, la arquitectura de Docker Engine (Figura 3) es similar a la de la virtualización basada en contenedores en general. Los contenedores de Docker se ejecutan sobre el demonio de Docker, que se encarga de ejecutar y administrar todos los contenedores de Docker. El cliente de Docker, que proporciona una interfaz de usuario para interactuar con contenedores a los usuarios de Docker, acepta comandos de los usuarios y luego los envía al demonio de Docker a través de las RESTful API. El uso de este método de comunicación permite que el cliente de Docker se ejecute en el mismo host que los contenedores, o incluso en diferentes hosts.

Docker solía utilizar LXC para crear contenedores Docker. Desde la versión 0.9, Docker ha reemplazado LXC con libcontainer [12] su propio formato de virtualización como el entorno de contenedor predeterminado ya que la comunidad de Docker desea no depender de un paquete de terceros. Sin embargo, con LXC o libcontainer, los namespaces, los cgroups, el sistema de archivos de unión (union file system) y las imágenes de Docker siguen siendo las principales tecnologías subyacentes para implementar los contenedores de Docker.

Docker aprovecha las ventajas de dos funciones de Linux, namespaces y cgroups, para crear de forma segura un entorno virtual para sus contenedores. Los cgroups, o grupos de control, proporcionan un mecanismo para contabilizar y limitar los recursos a los que pueden acceder los procesos de cada contenedor. Los namespaces

envuelven los recursos del sistema operativo en diferentes instancias. El uso de estas instancias da a los procesos que se ejecutan dentro de un contenedor la ilusión de que tienen sus propios recursos. Actualmente, Docker utiliza cinco namespaces para proporcionar a cada contenedor una vista privada del sistema host subyacente [13]: montaje, nombre de host, comunicación entre procesos (IPC), identificadores de proceso (PID) y red. Cada uno de ellos trabaja en tipos específicos de recursos del sistema. Los namespaces de red, por ejemplo, aíslan los recursos de red, como direcciones IP y tablas de enrutamiento IP, para proporcionar a cada contenedor una pila de red separada.

Docker lanza sus contenedores a partir de imágenes de Docker. Una imagen de Docker es una serie de capas de datos sobre una imagen base (Figura 4). Cada imagen de Docker comienza a partir de una imagen base, como la imagen base de Ubuntu o la imagen base de CentOS. Cuando los usuarios realizan cambios en un contenedor, en lugar de escribir directamente los cambios en la imagen del contenedor, Docker agrega una capa adicional que contiene los cambios en la imagen. Por ejemplo, si el usuario instala MySQL en una imagen de Ubuntu, Docker crea una capa de datos que contiene MySQL y luego la agrega a la imagen. Este proceso hace que el proceso de distribución de imágenes sea más eficiente, ya que solo es necesario distribuir la actualización.

Para trabajar con múltiples capas de una imagen, como si fuera una sola capa de sistema de archivos, Docker usa un sistema de archivos especial llamado Union File System (UnionFS). Permite que los archivos y directorios de diferentes sistemas de archivos se combinen en un único sistema de archivos coherente.

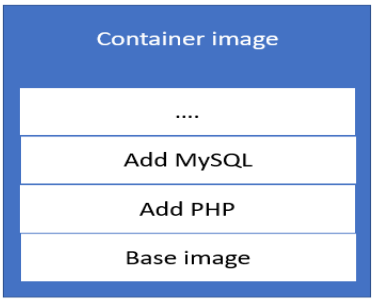

Figura 4. Imagen de contenedor

Docker Hub

Docker Hub [14] es un repositorio central de imágenes (tanto públicas como privadas), a través del cual los usuarios pueden compartir sus imágenes personalizadas. Los usuarios también pueden buscar imágenes publicadas y descargarlas con el cliente Docker. Además, los usuarios pueden verificar la autenticidad e integridad de las imágenes descargadas ya que Docker firmó y verificó las imágenes cuando su propietario las envió a Docker Hub.

4. ANÁLISIS DE SEGURIDAD DE DOCKER

La seguridad es uno de los principales desafíos al ejecutar servicios en entornos virtuales, especialmente en un sistema en la nube de múltiples inquilinos. Se afirma que las máquinas virtuales proporcionadas por técnicas de virtualización basadas en hipervisores son más seguras que los contenedores, ya que agregan una capa adicional de aislamiento entre las aplicaciones y el host. Una aplicación que se ejecuta dentro de una VM solo puede comunicarse con el kernel de la VM, no con el kernel del host. En consecuencia, para que la aplicación escale fuera de una VM, debe pasar por alto el kernel de la VM y el hipervisor antes de que pueda atacar el

kernel del host. Por otro lado, los contenedores pueden comunicarse directamente con el kernel del host, lo que permite que un atacante ahorre una gran cantidad de esfuerzo al irrumpir en el sistema host. Esto plantea una preocupación para la seguridad de los contenedores.

Docker también es una tecnología de virtualización basada en contenedores, por lo que tiene el mismo problema. Nuestro análisis tiene como objetivo descubrir si Docker proporciona un entorno seguro para ejecutar aplicaciones. El análisis considera dos áreas: la seguridad interna de los contenedores Docker y cómo los contenedores Docker interactúan con los sistemas de seguridad adicionales del kernel.

Seguridad Interna de Docker
Examinamos la seguridad interna de Docker según el sistema y el modelo del atacante y los requisitos de seguridad descritos por Reshetova [15] para comparar varias tecnologías de virtualización a nivel de sistema operativo.

El sistema y el modelo del atacante es el siguiente: una sola máquina host ejecuta varios contenedores Docker $A_1 \dots A_n$, en los que un subconjunto $\bar{A}$ de los contenedores está comprometido y el atacante tiene control total sobre ellos, pero el subconjunto restante de los contenedores A todavía están bajo el control de los usuarios legítimos. En este modelo, el atacante puede realizar varios tipos de ataques, como denegación de servicio y escalada de privilegios.

Para enfrentar estos ataques, los autores declararon que una solución de virtualización a nivel de sistema operativo debe satisfacer los siguientes requisitos: aislamiento de procesos, aislamiento del sistema de archivos, aislamiento de dispositivos, aislamiento de IPC, aislamiento de red y limitación de recursos. Las siguientes secciones presentan nuestro análisis sobre cómo Docker cumple con los requisitos.

Aislamiento de procesos
El objetivo principal del aislamiento de procesos es evitar que los contenedores comprometidos utilicen interfaces de gestión de procesos para interferir con otros contenedores. Docker logra el aislamiento de procesos agrupando los procesos que se ejecutan en contenedores dentro namespaces y limitando sus permisos y visibilidad a los procesos que se ejecutan en los otros contenedores y el host subyacente.

Este mecanismo funciona con el apoyo de los namespaces PID, que aíslan el espacio del número de ID de proceso de un contenedor del host. Dado que los namespaces PID son jerárquicos [16], un proceso solo puede ver los otros procesos en su propio namespace o en sus namespaces "secundarios". Como consecuencia, una vez que se crea un nuevo namespace y se asigna a un contenedor, el host puede observar y afectar los procesos dentro del nuevo namespace PID del contenedor, pero los procesos dentro del contenedor no pueden observar ni hacer nada a los otros procesos que se ejecutan en el anfitrión o en otros contenedores. Si el atacante no puede observar otros procesos, es más difícil atacarlos.

Los namespaces PID también permiten que cada contenedor tenga su propio proceso tipo ***init*** (PID 1), lo que hace que todos los procesos en un namespace terminen si se termina. Este proceso ayuda al administrador a cerrar completamente un contenedor cuando se detecta algo sospechoso.

Aislamiento del sistema de archivos
Para lograr el aislamiento del sistema de archivos, los sistemas de archivos del host y los contenedores deben protegerse contra el acceso y la modificación ilegítimos.

ISSN: 2007-4786

Docker usa los namespaces de montaje, también llamados namespaces del sistema de archivos, para aislar la jerarquía del sistema de archivos asociada con diferentes contenedores. Los namespaces de montaje proporcionan a los procesos de cada contenedor una vista diferente del árbol del sistema de archivos y restringen todos los eventos de montaje que ocurren dentro del contenedor para que solo tengan impacto dentro del contenedor.

Sin embargo, algunos de los sistemas de archivos del kernel no tienen espacio de nombres; por ejemplo, los que están debajo de */sys, /proc/sys, /proc/sysrq - trigger, /proc /irq y /proc/bus*, y un contenedor Docker necesita montarlos para poder operar. Esto causa el problema de que un contenedor hereda la vista de estos sistemas de archivos del host y puede acceder a ellos directamente. Docker limita las amenazas que un contenedor comprometido podría hacer al host a través de estos sistemas de archivos con los dos mecanismos de protección del sistema de archivos: eliminar el permiso de escritura para estos sistemas de archivos de los contenedores y no permitir que ningún proceso de un contenedor vuelva a montar ningún sistema de archivos. dentro del contenedor [17]. El segundo mecanismo se logra eliminando el permiso de usuario *CAP_SYS_ADMIN* de los contenedores.

Aislamiento del dispositivo

En Unix, el kernel y las aplicaciones acceden al hardware a través de los nodos del dispositivo, que básicamente son archivos especiales que actúan como interfaces para los controladores del dispositivo. Si un contenedor puede acceder a algunos nodos de dispositivos importantes, como */dev/mem* (la memoria física), */dev/sd∗* (el almacenamiento) o */dev/tty* (la terminal), puede causar graves daños al sistema host. Por lo tanto, es crucial limitar el conjunto de nodos de dispositivos a los que puede acceder un contenedor.

La función *Device Whitelist Controller* de cgroups proporciona un medio para limitar el conjunto de dispositivos a los que Docker permite que acceda un contenedor. También evita que los procesos en contenedores creen nuevos nodos de dispositivo. Además, Docker monta imágenes de contenedor con nodev, lo que significa que incluso si se creó previamente un nodo de dispositivo dentro de la imagen, los procesos en el contenedor que usan la imagen no pueden usarlo para comunicarse con el kernel. De forma predeterminada, Docker no otorga privilegios extendidos a sus contenedores. Por lo tanto, no pueden acceder a ningún dispositivo. Sin embargo, si el operador ejecuta un contenedor como privilegiado, Docker otorga acceso a todos los dispositivos al contenedor [15].

Aislamiento IPC

La IPC (inter-process communication) es un conjunto de objetos para intercambiar datos entre procesos, como semáforos, colas de mensajes y segmentos de memoria compartida. Los procesos que se ejecutan en contenedores deben restringirse para que puedan comunicarse solo a través de un determinado conjunto de recursos de IPC y no se les permite interferir con los de otros contenedores y la máquina host.

Docker logra el aislamiento de IPC mediante el uso de namespaces de IPC, lo que permite la creación de namespaces de IPC separados. Los procesos en un namespace de IPC no pueden leer ni escribir los recursos de IPC en otros namespaces de IPC. Docker asigna un namespace de IPC a cada contenedor, evitando así que los procesos de un contenedor interfieran con los de otros contenedores [15].

Aislamiento de Red

El aislamiento de la red es importante para prevenir ataques basados en la red, como Man-in-the-Middle (MitM) y ARP spoofing. Los contenedores deben configurarse de tal manera que no puedan espiar o manipular el tráfico de red de los otros contenedores ni del host.

ISSN: 2007-4786

Para cada contenedor, Docker crea una pila de red independiente mediante el uso de espacios de nombres de red. Por lo tanto, cada contenedor tiene sus propias direcciones IP, tablas de enrutamiento IP, dispositivos de red, etc. Esto permite que los contenedores interactúen entre sí a través de sus respectivas interfaces de red, que es lo mismo que interactúan con los hosts externos.

De forma predeterminada, la conectividad entre contenedores y la máquina host se proporciona mediante el puente Ethernet virtual [18]. Con este enfoque, Docker crea un puente ethernet virtual en la máquina host, llamado *docker0*, que reenvía automáticamente los paquetes entre sus interfaces de red. Cuando Docker crea un nuevo contenedor, también establece una nueva interfaz Ethernet virtual con un nombre único y luego conecta esta interfaz al puente. La interfaz también está conectada a la interfaz *etho* del contenedor, lo que permite que envíe paquetes al puente.

Limitación de Recursos

La denegación de servicio (DoS) es uno de los ataques más comunes en un sistema de múltiples inquilinos, donde un proceso o un grupo de procesos intenta consumir todos los recursos del sistema, interrumpiendo así el funcionamiento normal de los otros procesos. Para prevenir este tipo de ataque, debería ser posible limitar los recursos que se asignan a cada contenedor.

Los cgroups son el componente clave que Docker emplea para tratar este problema. Controlan la cantidad de recursos, como CPU, memoria y E/S de disco, que cualquier contenedor Docker puede usar, asegurando que cada contenedor obtenga su parte justa de los recursos y evitando que cualquier contenedor consuma todos los recursos. También permiten a Docker configurar los límites y restricciones relacionados con los recursos asignados a cada contenedor. Por ejemplo, una de esas restricciones es limitar las CPU disponibles a un contenedor específico [15].

Sistemas de Seguridad de Kernel

Existen algunos sistemas de seguridad del kernel para fortalecer la seguridad de un sistema host Linux, incluidas las capacidades de Linux y el Módulo de seguridad de Linux (LSM). Las capacidades de Linux restringen los privilegios asignados a cada proceso. LSM proporciona un marco que permite que el kernel de Linux admita diferentes modelos de seguridad. Los LSM que se han integrado en el kernel oficial de Linux incluyen AppArmor y SELinux.

Seguridad de Linux

Tradicionalmente, los sistemas Unix clasificaban los procesos en dos categorías: procesos privilegiados (propiedad de super usuario o root) y procesos sin privilegios (propiedad de usuarios normales). El kernel omitía todas las verificaciones de permisos en los procesos privilegiados, pero realizaba una verificación de permisos completa en los procesos sin privilegios. Sin embargo, el kernel de Linux, desde la versión 2.2, divide los privilegios del super usuario en capacidades, que el kernel puede habilitar o deshabilitar de forma independiente.

Los contenedores Docker se ejecutan en un kernel compartido con el sistema host, por lo que la mayoría de sus tareas pueden ser manejadas por el host. Como resultado, en la mayoría de los casos, no es necesario proporcionar privilegios de *root* completos a un contenedor, por lo que eliminar algunas de las capacidades de *root* de un contenedor no afecta la usabilidad o funcionalidad del contenedor, pero mejora efectivamente la seguridad del sistema. Por ejemplo, la capacidad *CAP_NET_ADMIN*, que proporciona la capacidad de modificar

ISSN: 2007-4786

la red del sistema, se puede eliminar de un contenedor, ya que el demonio Docker puede manejar toda la configuración de red antes de iniciar el contenedor.

Docker permite la configuración de las capacidades que puede utilizar un contenedor. De forma predeterminada, Docker deshabilita una gran cantidad de capacidades de Linux de sus contenedores para evitar que un intruso dañe el sistema host incluso cuando el intruso ha obtenido acceso de *root* dentro de un contenedor [19].

5. DISCUSIÓN

El análisis muestra que Docker proporciona un alto nivel de aislamiento y limitación de recursos para sus contenedores que utilizan namespaces, cgroups y su sistema de archivos de copia en escritura, incluso con la configuración predeterminada. También es compatible con varias funciones de seguridad del kernel, que ayudan a fortalecer la seguridad del host. El único problema que encontramos con Docker estaba relacionado con su modelo de red predeterminado. El puente ethernet virtual que Docker utiliza como modelo de red predeterminado es vulnerable a ataques de suplantación de identidad de ARP y de inundación de MAC, ya que no proporciona ningún filtro en el tráfico de red que pasa a través del puente. Sin embargo, este problema puede resolverse si el administrador agrega manualmente el filtrado, como ebtables [6], al puente, o cambia la conectividad de la red a una más segura, como la red virtual.

También vale la pena destacar que, si el operador ejecuta un contenedor como privilegiado, Docker otorga permisos de acceso completo al contenedor, que es casi el mismo que el de los procesos que se ejecutan de forma nativa en el host. Por lo tanto, es más seguro operar contenedores como no privilegiados.

Además, aunque los contenedores pueden proporcionar una mayor densidad de entornos virtuales y un mejor rendimiento, tienen una mayor superficie de ataque que las máquinas virtuales, ya que los contenedores pueden comunicarse directamente con el kernel del host. Sin embargo, es posible reducir la superficie de ataque manteniendo estas ventajas. Por ejemplo, esto se puede lograr colocando contenedores dentro de máquinas virtuales.

6. CONCLUSIÓN

La virtualización basada en contenedores puede proporcionar entornos virtuales de mayor densidad y un mejor rendimiento que la virtualización basada en hipervisores. Sin embargo, se argumenta que este último es más seguro que el primero. En este documento, realizamos un análisis sobre Docker, que es una de las tecnologías de virtualización basadas en contenedores más populares, para descubrir qué tan seguros son sus contenedores. Nuestro análisis muestra que los contenedores Docker son bastante seguros, incluso con la configuración predeterminada. El nivel de seguridad de los contenedores Docker también podría incrementarse si el operador ejecuta el contenedor sin privilegios de acceso.

En trabajos futuros, se pretende configurar con docker un ambiente para el correcto funcionamiento de un proyecto implementando todas las medidas de seguridad mencionadas en este artículo.

REFERENCIAS

[1] C. Burniske, «ARK Invest,» 21 Octubre 2014. [En línea]. Available: https://ark-invest.com/articles/analyst-research/containers-virtualization/. [Último acceso: 2 Abril 2021].

[2] OpenSource, «opensource,» 2021. [En línea]. Available: https://opensource.com/resources/what-docker. [Último acceso: 1 Abril 2021].

[3] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier y L. Peterson, «Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors,» *EuroSys,* vol. 7, p. 275–287, 2007.

[4] OpenVz, «Open source container-based virtualization for Linux.,» 2021. [En línea]. Available: https://openvz.org/.

[5] Linux Container, «Linux Containers,» 2014. [En línea]. Available: https://linuxcontainers.org/.

[6] D. Merkel, «Docker: Lightweight Linux Containers for Consistent Development and,» *Linux Journal,* p. 238, 2014.

[7] B. R. Anderson, A. K. Joines y T. E. Daniels, «Xen worlds: leveraging virtualization in distance education,» *ITiCSE,* pp. 293-297, 2009.

[8] A. Kivity, Y. Kamay, D. Laor, U. Lublin y A. Liguori, «kvm: the Linux virtual machine monitor,» *Linux symposium,* pp. 225-230, 2007.

[9] M. G. Xavier, M. V. Neves, F. D. Rossi y T. C. Ferreto, «Performance evaluation of container-based virtualization for high performance computing environments,» *IEEE,* pp. 233-240, 2013.

[10] N. Regola y J. C. Ducom, «Recommendations for Virtualization Technologies in High Performance Computing,» *IEEE,* pp. 409-416, 2010.

[11] P. Padala, X. Zhu, . Z. Wang, S. Singhal y K. G. Shin, «Performance Evaluation of Virtualization Technologies for Server Consolidation,» *HP Labs Tec,* 207.

[12] S. Yegulalp, «INFOWORLD TECH WATCH,» 2014. [En línea]. Available: https://www.infoworld.com/article/2607966/4-reasons-why-docker-s-libcontainer-is-a-big-deal.html.

[13] D. J. Walsh, «OpenSource,» 22 Julio 2014. [En línea]. Available: https://opensource.com/business/14/7/docker-security-selinux.

[14] Docker, «Docker,» 2021. [En línea]. Available: https://www.docker.com/products/docker-hub.

[15] E. Reshetova, J. Karhunen, T. Nyman y N. Asokan, «Security of OS-Level Virtualization Technologies,» *Secure IT Systems,* pp. 77-99, 2014.

[16] P. Emelyanov y K. Kolyshkin, «Lwn.net,» 2021. [En línea]. Available: https://lwn.net/Articles/259217/.

[17] D. J. Walsh, «OpenSource,» 3 Septiembre 2015. [En línea]. Available: https://opensource.com/business/14/9/security-for-docker.

[18] Docker, «docs.docker.com,» 2021. [En línea]. Available: https://docs.docker.com/network/.

[19] die.net, «https://linux.die.net/,» 2019. [En línea]. Available: https://linux.die.net/man/7/capabilities.

Correo electrónico autor: ***m20301149@villahermosa.tecnm.mx***