Análisis de la seguridad de Docker

Thanh Bui

Escuela de Ciencias de la Universidad de Aalto

[thanh.bui@aalto.fi](mailto:bui@aalto.fi)

Abstracto

arXiv:1501.02967v1 [cs. CR] 13 ene 2015

En los últimos años, el uso de technolo de virtualización - gies ha aumentado dramáticamente. Esto hace que la demanda de soluciones de virtualización eficientes y seguras ses vuelva más obtuosa- viosa. La virtualización basada en contenedores y la virtualización basada en hipervisores son dos tipos principales de virtualizatien technolo: gies que han surgido al mercado. De estas dos clases, la virtualización basada en contenedores es capaz de proporcionar un entorno virtual más ligero y eficiente, pero no sin problemas de seguridad. En este artículo, analizamos el nivel de seguridad de Docker, un conocido representante de los enfoques basados en contenedores. El análisis considera dos nal áreas: (1) la seguridad internacional de Docker y (2) cómo Docker interactúa con las características de seguridad del kernel de Linux, como SELinux y AppArmor,, para reforzar el sistema host. Además, el documento también analiza e identifica lo que se podría hacer al usar Docker para aumentar su nivel de seguridad.

PALABRAS CLAVE: Contenedores, Docker, Seguridad

1 Introducción

En la última década se ha producido una explosión de desarrollo en el ámbito de las tecnologías de virtualización, que permiten la partición ing de un sistema informático en múltiples en - vironments virtuales aislados.. Las tecnologías ofrecen beneficios sustanciales que han estado impulsando su desarrollo rápidamente. Una de las razones más comunes para adoptar tecnologías de virtualización es la virtualización de servidores en centros de datos. Con server virtualiza- tion, un administrador puede crear una o más instancias de sistema virtual en un único servidor. Estos sistemas virtuales funcionan como servidores físicos reales y se pueden alquilar por suscripción. Amazon EC2, Rackspace y DreamHost son algunas instancias populares de dichos proveedores de servicios de centros de datos. Otro uso común es para la virtualización de escritorios, donde un equipo puede ejecutar varias instancias del sistema operativo. Escritorio- virtualiza- virtualiza tion proporciona soporte para aplicaciones que solo se pueden ejecutar en un sistema operativo específico.

El crecimiento en el uso de tecnologías de virtualización impulsa la demanda de una solución de virtualización que puede proporcionar entornos de usuario densos, escalables y seguros. Un gran número de soluciones de virtualización han surgido al mercado. Se pueden clasificar en dos clases principales: virtualización basada en contenedores y virtualización basada en hipervisor. . De estas dos clases, la virtualización basada en contenedores es capaz de proporcionar una virtual más ligera y eficiente en- vironment. Permite que diez veces más entornos virtuales se ejecuten en un servicio físico r en comparación con los virtues basados en hipervisor -

alización [19]. Sin embargo, la virtualización basada en contenedores también incluye problemas de seguridad.

En este artículo, analizamos el nivel de seguridad de Docker [17], un conocido representante del enfoque de alización de virtubasado en contenedores. We Consideramos dos áreas: (1) la seguridad interna de Docker y (2) cómo Docker interactúa con las características de seguridad del kernel de Linux, como SELinux y AppArmor,, con el fin de reforzar el sistema host. El análisis examinó la seguridad interna de Docker en función del nivel de aislamiento que Docker puede proporcionar a sus entornos virtuales. La interacción entre Docker y las características de seguridad del kernel se estimó en función de cómo Docker sustituye las características. Hasta biencósticos, Docker es una tecnología relativamente nueva, y este es uno de los primeros análisis de este tipo que se centran en sus aspectos de seguridad.

El documento está estructurado de la siguiente manera: La Sección 2 proporciona una vista de alto nivel de las dos clases de soluciones de virtualización. La sección 3 ofrece una visión general de Docker y sus tecnologías subyacentes. La Sección 4 presenta d nuestro análisis de Docker se- curity, y luego en la Sección 5, discutimos el nivel de seguridad de Docker y lo que se hace para aumentar su nivel de seguridad. El documento concluye con un resumen en la Sección 6.

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 virtu- alización a nivel de sistema operativo, mientras que el segundo pro- vides virtualización en el nivel de hardware. Cada uno de los ap- proaches tiene sus propias ventajas y desventajas, que se describen en esta sección.

La virtualización basada en contenedores es un enfoque ligero de virtualización virtual que utiliza el kernel de host para ejecutar varios entornos vir- tual. Estos entornos virtuales a menudo se vuelven a conferir como contenedores. Linux-VServer [31], OpenVZ [11] y Linux Container (LXC) [10] son los tres principales repre- sentatives de este enfoque. La arquitectura general de una solución de virtualización basada en contenedor se ilustra en Fig. 1. La virtualización basadaen contenedor se virtualiza a nivel del sistema operativo, lo que permite que varias aplicaciones funcionen sin ejecutar redundantemente otros kernels del sistema operativo en el host. Sus contenedores parecen procesos normales desde el exterior, que se ejecutan en la parte superior del kernel compartido con la máquina host. Proporcionan entornos aislados con necesidades: recursos sary para ejecutar aplicaciones. Estos recursos se pueden compartir con el host o instalarse por separado dentro del contenedor.

Las soluciones de virtualización basadas en hipervisores proporcionan virtu--

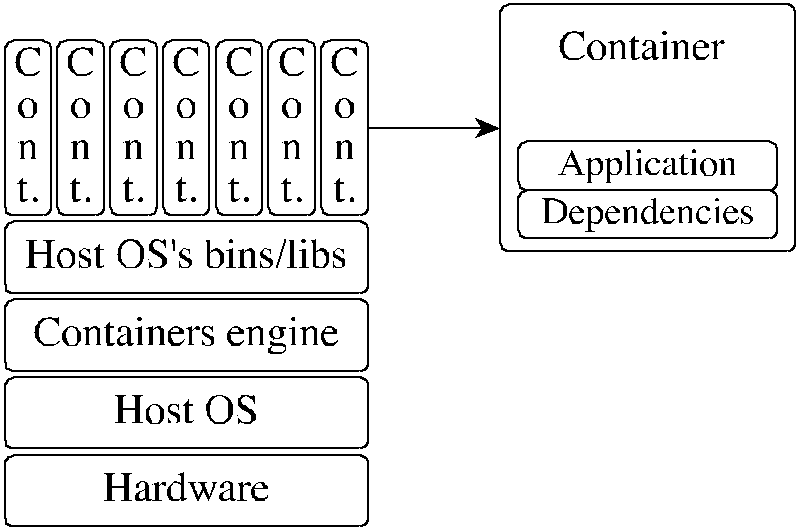
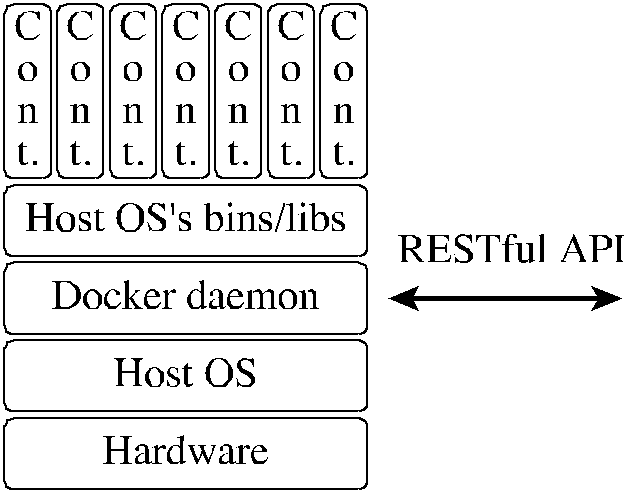


Figura 1: Arquitectura de virtualización basada en contenedores

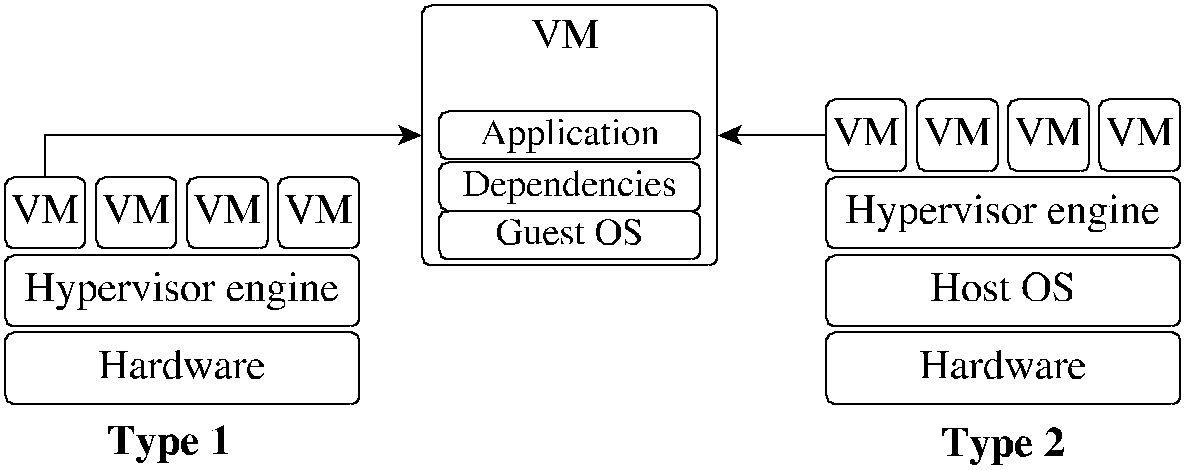


Figura 2: Arquitectura de virtualización basada en hipervisores

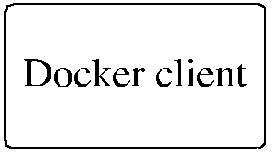
alization a nivel de hardware. A diferencia de la virtualización basada en contenedores, un hipervisor establece ma- chines virtuales (VM) completos en la parte superior del sistema operativo host (Fig. 2). Cada máquina virtual consta no solo de una aplicación y sus dependencias, sino también de un sistema operativo invitado completo junto con un kernel independiente. There Hay dos clases de hipervisores: el hipervisor Tipo 1, también conocido como el hipervisor de metal desnudo, que funciona directamente sobre el hardware subyacente del host, y el hipervisor Tipo 2, también conocido como hipervisor hospedado, que funciona en la parte superior del sistema operativo host [26]. Xen [18] es un ejemplo del primero, mientras que KVM [25] es del segundo. Puesto 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 algunas ventajas a la virtualización basada en contenedores a través del virtu basado en hipervisores: la alización.. En primer lugar, la virtualización basada en contenedores puede proporcionar una mayor densidad de entornos virtuales. Dado que un contenedor no incluye un sistema operativo completo, el tamaño y los orígenes 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 desventajas que las máquinas virtuales tradicionales en el mismo host. tainers En segundo lugar, la virtualización basada en contenedores también ofrece un mejor rendimiento. Esto ha sido demostrado por experimentos en algunos estudios [32, 28, 27, 21]. 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.

Sin embargo, a pesar de todas las ventajas mencionadas, la virtualización basada en contenedores no puede admitir una variedad de entornos en la forma en que lo hace la virtualización basada en hipervisor, ya que todos los entornos de los contenedores deben ser del mismo tipo que el del host. Por ejemplo, los contenedores de Windows no se pueden ejecutar encima de un host Linux.

Figura 3: Arquitectura del motor de Docker

3 Descripción general de Docker



Docker es una tecnología de contenedor de código abierto con el abil- ity "para construir, enviar y ejecutar aplicaciones distribuidas" [17]. Se ha utilizado en algunas aplicaciones populares, como Spotify, Yelp, y Ebay.

did 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 contenedoresde forma sencilla y segura. En segundo lugar, los desarrolladores pueden empaquetar aplicaciones en contenedores de Docker ligeros que pueden funcionar en casi cualquier lugar sin modificaciones. Además, Docker puede implementar más entornos virtuales que otras tecnologías en el mismo hardware [19]. Por último, pero no menos importante, Docker coopera bien con herramientas de terceros, que simplifican el proceso de administración e implementación de contenedores de Docker. Las herramientas de DevOps, como Puppet [13], An- sible [1] y Vagrant [16] pueden integrarse con Docker, lo que hace que los contenedores de Docker se implementen deforma sencilla en una nube. Además, muchas herramientas de orquestación, como Mesos [22], Shipyard [15] y Kubernetes [7], también admiten Docker con- tainers. Estas herramientas proporcionan una capa abstracta de administración y programación de recursos a través de Docker.

Docker consta de dos componentes principales: motor Docker 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. En las secciones siguientes se describen en detalle estos dos componentes.

3.1 Motor Docker

El motor Docker es una herramienta de empaquetado ligera y portátil [17] que se basa en la virtualización basada en contenedores. Por lo general, la arquitectura del motor Docker (Fig. 3) es simi- lar a la de la virtualización basada en contenedores en general. Los contenedores de Docker se ejecutan encima del demonio de Docker, que se encarga de ejecutar y administrar todos los contenedores de Docker. El cliente de Docker, que proporciona un usuario inter- face para interactuar con contenedores a los usuarios de Docker, acepta comandos de los usuarios y, a continuación, lo envía al demonio de Docker a través de las API RESTful. El uso de este método de comunicación munication permite que el cliente de Docker se ejecute en el mismo host que los contenedores, o incluso en hosts diferentes.

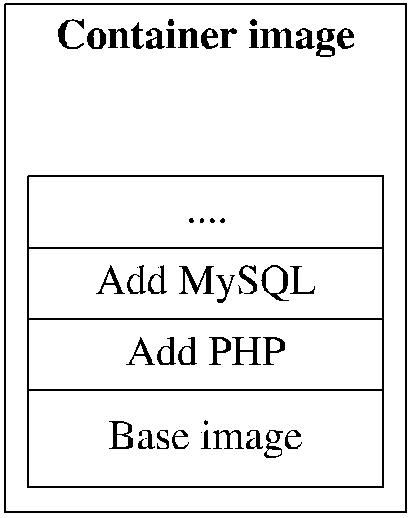


Figura 4: Imagen del contenedor

Contenedor Docker

Docker solía mercantilizar LXC para crear Docker contain- ers. Desde la versión 0.9, Docker ha reemplazado LXC por lib- container [8] - 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 espacios de nombres, cgroups,, el archivo de unión sys- tem, y las imágenes de Docker siguen siendo las principales aplicaciones tecnológicas subyacentes nologies para implementar contenedores Docker.

Docker aprovecha dos características de Linux, nombres: paces y cgroups,, para crear de forma segura un entorno virtual para sus contenedores. Los cgroups,, o grupos de control, proporcionan un mecanismo para la contabilidad y la limitación de los recursos a los que pueden acceder los procesos de cada contenedor. Los espacios de nombres ajustan 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 espacios de nombres para proporcionar a cada con- tainer una vista privada del sistema host subyacente [23]: montaje, nombre de host, comunicación entre procesos (IPC), identificadores de pro- cess (PID) y red. Cada uno de ellos funciona en tipos específicos de recursos del sistema. Los espacios de nombres de red, por ejemplo, aíslan los recursos de red, como los vestidos de anuncio IP y las tablas de enrutamiento IP, con el fin de proporcionar a cada con- tainer una pila de red separada.

Docker inicia sus contenedores a partir de imágenes de Docker. Una imagen de Docker es una serie de capas de datos encima de una base im-age (Fig. 4). Cada imagen de Docker comienza desde una versión de im base, como la imagen base de Ubuntu o la imagen base de OpenSuse. 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 im - age. Por ejemplo, si el usuario instala MySQL en una imagen de Ubuntu, Docker crea una capa de datos que contiene MySQL y, a continuación, se agrega a la imagen. Este proceso hace que la imagen destrite- proceso de bution más eficientemente, ya que solo es necesario distribuir la actualización.

Para trabajar con varias capas de una imagen, ya que era una sola capa de sistema de archivos, Docker utiliza un sistema de archivos especial llamado Union File System (UnionFS). It Permite que los archivos y directorios de diferentes sistemas de archivos se combinen en un único sistema de archivos coherente.

3.2 Docker Hub

Docker hub [4] es un repositorio central de imágenes (tanto públicas como privadas), a través de las cuales los usuarios pueden compartir su im--

Edades. Los usuarios también pueden buscar imágenes publicadas y cargarlas con el cliente de Docker. Además, los usuarios pueden comprobar la autenticidad y la integridad del im descargado desde que Docker firmó y verificó las imágenes cuando su propietario las envió al concentrador.

4 Análisis de seguridad de Docker

La seguridad es uno de los principales desafíos a la hora de ejecutar ser- vices en entornos virtuales, especialmente en un sistema en la nube multiinquilino. Las máquinas virtuales proporcionadas por las técnicas de virtualización basadas en hipervisores se afirman que son más curables que los contenedores, ya que agregan una capa adicional de aislamiento entre las aplicaciones y el host. Una ejecución de la aplicación dentro de una máquina virtual solo puede comunicarse con el kernel de la máquina virtual, no con el kernel del host. Por consiguiente, para que la aplicación se escale fuera de una máquina virtual, debe omitir el kernel de la máquina virtual y el hipervisor antes de que pueda atacar el ker del host - nel. Por otro lado, los contenedores pueden comunicarse directamente con el kernel del host, lo que permite a un atacante ahorrar una gran cantidad de esfuerzo al entrar en el sistema host. Esto plantea una preocupación de seguridad sobre los contenedores.

Docker también es un technolode virtualización basado en contenedores , por, lo que tiene el mismo problema. Nuestro análisis tiene como objetivo eliminar si Docker proporciona un entorno seguro para ejecutar aplicaciones. El análisis tiene en cuenta dos áreas: la seguridad interna de los contenedores de Docker y cómo los contenedores de Docker están en- teract con los sistemas de seguridad adicionales del kernel.

4.1 Docker Internal Security

Examinamos la seguridad interna de Docker en función del modelo del sistema y del atacante y los requisitos de seguridad, tal como los describimos Reshetova et al. [29] por comparar varias de las tecnologías de virtualización de nivel de sistema operativo.

El modelo de sistema y atacante es el siguiente: Una sola máquina host está ejecutando una serie de contenedores Docker c1 ... cn, en el que un subconjunto C de los contenedores se ven comprometidos y el atacante tiene control total sobre ellos, pero el subconjunto restante de contenedores C todavía está bajo el control de los usuarios de fiar- imate. En este modelo, el atacante puede realizar varios tipos de ataques, como denegación de servicio y privilegios de calado..

Con el fin de encontrar con estos ataques, los autores afirmaron que una solución de virtualización de nivel de sistema operativo debe satisfacer los requisitos debajada: aislamiento de procesos, aislamiento del sistema de archivos, aislamiento de dispositivos, aislamiento IPC, aislamiento de red y limitación ing de recursos. En las secciones siguientes se presenta nuestro análisis sobre cómo Docker cumple los requisitos.

Aislamiento de procesos

El objetivo principal del aislamiento de procesos es evitar que los contenedores con errores de compro utilicen interfaces de gestión de procesos para interferir con otros contenedores. Docker logra el aislamiento de los procesos ajustando los procesos que se ejecutan en contenedores en espacios de nombres y limitando sus permisos y visibilidad a los procesos que se ejecutan en los otros contenedores y al host que lo subyenta.

Este mecanismo funciona con la compatibilidad de los espacios de nombres PID, que aíslan el espacio de número de identificador de proceso de un contenedor del del host. Dado que los espacios de nombres PID son jerárquicos [12], un proceso solo puede ver los demás procesos en su propio espacio de nombres o en sus espacios de nombres "secundarios". Como consecuencia, una vez que se crea un nuevo espacio de nombres y se asigna a un contenedor, el host puede observar y afectar a los procesos dentro del nuevo espacio de nombres PID del contenedor, pero los datos profesionales cesses dentro del contenedor no pueden observar ni hacer nada a los otros procesos que se ejecutan en el host o en otros contenedores. Si el atacante no puede observar otros procesos, es más difícil atacarlos.

Los espacios de nombres PID también permiten que cada contenedor tenga su propio proceso init-like (PID 1), lo que hace que todos los procesos de un espacio de nombres se terminen si se termina. Este proceso ayuda al administrador a cerrar por completo 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 estar protegidos contra el acceso y la modificación ilegítimos.

Docker utiliza los espacios de nombres deth montaje, también denominados filesys- espacios de nombres tem, para aislar la jerarquía del sistema de archivos associ- ated wi diferentes contenedores. Los espacios de nombres de montaje pro- vide los procesos de cada contenedor una vista diferente del árbol del sistema de archivos y restringir todos los eventos de montaje que se producen dentro del contenedor para que solo tengan impacto dentro del con- tainer. Sin embargo, algunos de los sistemas de archivos del kernel no tienen espacio de nombres;; por ejemplo, los de /sys, /proc/sys,

/proc/ , sysrq, /proc/irq y /proc/bus, y un

El contenedor de Docker debe montarlos para funcionar. Esto provoca el problema de que un contenedor hereda la vista de estos sistemas de archivos del host y pueda acceder a ellos directamente. Docker limita las amenazas que un con- tainer comprometido podría hacer al host a través de estos sistemas de archivos con los dos mecanismos de protección de sistemas de archivos: (1) eliminación del permiso de escritura a estos sistemas de archivos de contenedores y (2) no permitir que ningún proceso de un contenedor vuelva a montar ningún sistema de archivos dentro del contenedor [24]. El segundo mecanismo se logra mediante la eliminación de la CAP \_SY S\_ADMIN capa- capacidad de los contenedores.

Docker también emplea un mecanismo llamado sistema de archivos de copia en escritura [24]. Como se mencionó anteriormente, Docker crea con- tainers basados en imágenes del sistema de archivos y un contenedor puede escribir contenido en su propia imagen base. Cuando se crean varios contenedores en la misma imagen, el sistema de archivos de copia en escritura reduce cada contenedor para escribir contenido en su archivo específico sys- tem, lo que impide que otros contenedores detecte los cambios que se producen dentro del contenedor.

Aislamiento de dispositivos

En Unix, el núcleo y las aplicaciones acceden al hardware a través de nodos de dispositivos que básicamente son archivos especiales que actúan como interfaces a los controladores de dispositivo. Si un contenedor puede cesss algunos nodos de dispositivo importantes, como /dev/mem (la memoria física), /dev/sd∗ (el almacenamiento) o /dev/tty (el terminal), puede causar daños graves en el sistema host.

Por lo tanto, es crucial limitar el conjunto de nodos de dispositivo a los que puede acceder un con- tainer.

La característica Device Whitelist Controller [3] de cgroups proporciona medios para limitar el conjunto de dispositivos a los que Docker al- lows a container to access. It También impide que los procesos de los contenedores creen nuevos nodos de dispositivo. Además, Docker monta imágenes de contenedor con nodev, lo que significa que incluso si un nodo de dispositivo se creó previamente dentro de la imagen, los procesos del contenedor que utilizan la imagen no pueden utilizarlo 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 exe- cutes un contenedor como "privilegiado", Docker concede acceso a todos los dispositivos al contenedor.

Aislamiento IPC

El IPC (comunicación entreprofesionales) es un conjunto de objetos para intercambiar datos entre procesos, como semáforos, colas de mensajes y segmentos de memoria compartida. Los datos que se ejecutan en contenedores deben restringirse para que puedan comunicarse sólo a través de un determinado conjunto de recursos IPC y no se les permita interferir con los de otros contenedores y la máquina host.

Docker logra el aislamiento de IPC mediante los nombres de IPC, que permite la creación de nombres IPC separados. Los procesos de un espacio de nombres IPC no pueden leer ni escribir los recursos de IPC en otros espacios de nombres IPC. Docker asigna un espacio de nombres IPC a cada contenedor, lo que impide que los procesos de un contenedor interfieran con los de otros contenedores.

Aislamiento de red

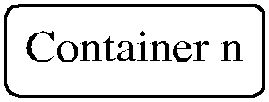
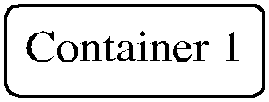
El aislamiento de la red es importante para evitar las conexiones basadas en la red, como Man-in-the-Middle (MitM) y ARP spoof- ing. Los contenedores se deben configurar de tal manera que no puedan espiar o manipular el tráfico de red de los otros contenedores ni del host.

Para cada contenedor, Docker crea una pila de trabajo de red independiente mediante 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 cómo interactúan con hosts externos.

De forma predeterminada, la conectividad entre contenedores y la máquina host se proporciona mediante el puente Ethernet virtual [5] (Fig. 5). Con este enfoque, Docker crea un puente et- ernet virtual en la máquina host, denominado docker0, que reenvía au- tomáticamente paquetes entre sus interfaces de red. Cuando Docker crea un nuevo contenedor, también establece una nueva interfaz Ethernet virtual con un nombre único y, a continuación, conecta esta interfaz al puente. La interfaz también está conectada a la interfaz eth0 ing del contenedor, lo que permite que el contenedor envíe los paquetes al Bridge.

Observamos aquí que el modelo de conectividad predeterminado de Docker es vulnerable a la suplantación de ARP y a la inundación de Mac en- tacks ya que el Puente adelante todos sus paquetes entrantes sin ningún filtrado.

Figura 5: El modelo de red predeterminado de Docker



Limitación de los recursos

of La denegación de servicio (DoS) es uno de los ataques más comunes a un sistema multiinquilino, donde un proceso o un grupo de pro- cesses intentan consumir todos los recursos del sistema, interrumpiendo así el funcionamiento normal de los otros procesos. Para evitar este tipo de ataque, debe ser posible limitar los recursos que se asignan a cada contenedor.

Los grupos C 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 con- tainer de Docker puede usar, lo que garantiza que cada contenedor obtenga su parte justa de los recursos e impida que cualquier contenedor consuma todos los recursos. También permiten a Docker calcular los límites y restricciones relacionados con los recursos allo- cated a cada contenedor. Por ejemplo, una de estas restricciones es limitar las CPU disponibles para un contenedor específico.

4.2 Docker y Kernel Security Systems

Algunos sistemas de seguridad del núcleo existen con el fin de reforzar la seguridad de un sistema host Linux, incluyendo Linux capabili- lazos y Linux Security Module (LSM). Las capacidades de Linux restringen los privilegios asignados a cada proceso. LSM pro- vides un framework que permite que el kernel de Linux para soportar diferentes modelos de seguridad. Los LSM que han sido inte-rallados en el kernel oficial de Linux incluyen AppArmor [20], SELinux [30] y Seccomp [14].

Este documento examina las capacidades de Linux y dos LSM, Ap- pArmor [20] y SELinux [30], que Docker actualmente supusor puertos. Docker también puede colaborar con Seccomp, pero solo si se utiliza LXC; por lo tanto, no lo incluimos en la encuesta. Aunque Docker no admite otros sistemas de seguridad en este momento, no interfiere con ellos. Por lo tanto, estos sistemas se pueden ejecutar independientemente de los contenedores de Docker para pro- tect el host [2].

Linux Capabilities

Como se indica en la página de administración de capacidades de Linux [9], tradition- ally, los sistemas Unix clasificaron los procesos en dos categorías: procesos privilegiados (propiedad de superusuario o raíz) y procesos sin privilegios (propiedad de usuarios normales). El kernel omitió todas las comprobaciones de permisos en los procesos privilegiados, pero llevó a cabo la comprobación completa de permisos en pro- cessessin privilegios . Sin embargo, el kernel de Linux, desde la versión 2.2, divide

CAP \_SET P CAP Modificar capacidades de proceso CAP \_SY S\_MODULE Insertar/Quitar módulos del núcleo CAP \_SY S\_RAW IO Modificar memoria del núcleo

CAP \_SY S\_P ACCT Configure el CAP de la contabilidad del proceso \_SY S\_NICE modificar la prioridad de los procesos CAP \_SY S\_RESOURCE invalidar los límites de recursos CAP \_SY S\_T IME Modificar el CAP del reloj del sistema \_SY S\_T T Y \_CONF los dispositivos IG Configure tty

CAP \_AUDIT \_W RIT E Escribir el registro de auditoría

CAP \_AUDIT \_CONT ROL Configure Audit Subsystem CAP \_MAC\_OV ERRIDE Ignore Kernel MAC Policy CAP \_MAC\_ADMIN Configure MAC Configuration CAP \_SY SLOG Modify Kernel printk behavior CAP \_NET \_ADMIN Configure the network the network

CAP \_SY S\_ADMIN Atrapa a todos

Tabla 1: Algunas capacidades no permitidas en contenedores Docker

[24]

los privilegios del superusuario en capacidades, que el kernel puede habilitar o deshabilitar de forma independiente.

Los contenedores de Docker se ejecutan en un kernel compartido con el sistema host, por lo que el host puede controlar la mayoría de sus tareas. Como resultado, en la mayoría de los casos, no es necesario proporcionar privilegios raíz completos a un contenedor, por lo que la eliminación de algunas de las capacidades raíz de un contenedor no afecta a la facilidad de uso o funcionalidad del contenedor, pero mejora eficazmente la seguridad del sistema. Por ejemplo, la capacidad CAP \_NET \_ADMIN, que proporciona el abil- ity para modificar la red del sistema, se puede quitar de un contenedor, ya que el demonio de Docker puede controlar toda la configuración de red antes de iniciar el contenedor.

Docker permite la configuración de las capacidades que puede usar un con- tainer. De forma predeterminada, Docker deshabilita un gran número de capacidades de Linux de sus contenedores para evitar que un intruso dañe el sistema host incluso cuando el intruso ha obtenido acceso root dentro de un contenedor. Algunas de las capacidades se presentan en la tabla 1, y su de- scription detallado se puede encontrar en la página de comando man de las capacidades de Linux [9].

Selinux

SELinux es una mejora de seguridad para el sistema Linux. Linux viene con el mecanismo estándar de con- trols de acceso discrecional (DAC) (es decir, indicadores de propietario/grupo y permiso de un objeto) para controlar el acceso a un objeto. SELinux proporciona una capa adicional de comprobación de permisos, llamada Control de acceso obligatorio, después de que la DAC estándar se for- formó. En SELinux,, todo está controlado por etiquetas. Ev: el archivo/directorio, el proceso y el objeto del sistema tienen una etiqueta. El administrador del sistema utiliza estas etiquetas para escribir reglas para controlar el acceso entre procesos y objetos del sistema. Estas reglas se denominan directivas. Las directivas de SELinux se pueden dividir en tres clases: aplicación de tipos, aplicación de secude rity varios niveles (MLS) y aplicación de seguridad multiescu (MCS).

of Con el mecanismo DAC, los propietarios tienen total discreción sobre sus objetos, lo que significa que si los propietarios son compro- engañado, el atacante tiene control sobre todos sus objetos. En el modelo SELinux, en cambio, el núcleo gestiona y aplica

todos los controles de acceso sobre los objetos, no sus propietarios. Esto proporciona una separación segura para los contenedores, ya que puede impedir que los procesos, incluso con privilegios raíz, dentro de un contenedor tengan acceso ilegítimo a los objetos fuera de los contenedores.

Docker utiliza dos clases de aplicación de directivas: Type en- forcement y MCS enforcement [24]. La aplicación ment Type protege el host de los procesos en contenedores y la aplicación MCS protects un contenedor de otro con- tainer.

Con la aplicación de tipos, Docker etiqueta todos los elementos de acceso de contenedor con svirt\_lxc\_net\_t tipo y todo el contenido de un contenedor con svirt\_sandbox\_file\_t tipo. Los procesos que se ejecutan con svirt\_lxc\_net\_t tipo solo pueden acceder/escribir al contenido etiquetado con svirt\_sandbox\_file\_t tipo, pero no a ninguna otra etiqueta del sistema. Por lo tanto, los datos que se ejecutan dentro de contenedores solo pueden usar el contenido en contenedores laterales. Sin embargo, solo con esta aplicación de directivas, Docker permite que los procesos de un contenedor tengan acceso al contenido de otros contenedores. La aplicación de MCS es nec- essary para resolver este problema. Cuando se inicia un contenedor, el demonio de Docker elige una etiqueta MCS aleatoria y, a continuación, coloca esta etiqueta en todos los procesos y el contenido del contenedor. El kernel solo permite que los procesos accedan al contenido con la misma etiqueta MCS, evitando así que un proceso comprometido en un contenedor ataque a otros contenedores.

Apparmor

AppArmor es también un modelo de mejora de seguridad para Linux basado en el control de acceso obligatorio como SELinux,, pero re- estrictando su alcance a programas individuales. It Permite al administrador cargar un perfil de seguridad en cada programa, lo que limita las capacidades del programa. AppArmor sup- ports dos modos: modo de aplicación y modo de queja / aprendizaje. El modo de aplicación aplica las directivas definidas en el perfil. Sin embargo, en el modo dequeja/aprendizaje, se permiten las infracciones de las directivas de perfil, pero también se registran. Este registro puede ser útil para desarrollar nuevos perfiles más adelante.

En los sistemas que admiten AppArmor, Docker proporciona una interfaz para cargar un perfil de AppArmor predefinido al iniciar un nuevo contenedor. Este perfil se carga en el contenedor en modo de aplicación para asegurarse de que los procesos en el contenedor están restringidos según el perfil. Si el administrador ng no especifica un perfil al iniciar un contenedor, el nies demonio de Docker carga automáticamente ically un perfil predeterminado en el contenedor, que elimina el acceso a sistemas de archivos importantes en el host, como

/sys/fs/cgroups/ y /sys/kernel/security/.

5 Discusión

El análisis muestra que Docker proporciona un alto nivel de iso- lación y limitación de recursos para sus contenedores mediante nombres- paces, cgroups y su sistema de archivos de copia en, lation escritura, incluso con la configuración predeterminada. It También es compatible con varias características de se- curity del kernel, que ayudan a reforzar 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 su modelo de red predeterminado, es vul- nerable a la suplantación de ARP y los ataques de inundación MAC desde

no proporciona ningún filtro en el tráfico de red que pasa a través del Bridge. Sin embargo, este problema se puede resolver si el administrador agrega manualmente el filtrado, como ebtables [6], al puente, o cambia la conectividad de red a uno más seguro, como la red virtual.

También vale la pena destacar que si el operador ejecuta un con- tainer como "privilegiado", Docker concede 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 "sin privilegios".

Además, aunque los contenedores pueden proporcionar una mayor densidad de entornos virtuales y un mejor rendimiento, tienen una superficie attack más grande que las máquinas virtuales, ya que los con- tainers pueden comunicarse directamente con el kernel del host. Cómo- nunca, es posible reducir la superficie de ataque mientras que la principal- mantener estas ventajas. Por ejemplo, esto se puede lograr colocando contenedores dentro de máquinas virtuales.

6 Conclusión y trabajo futuro

La virtualización basada en contenedores puede proporcionar entornos virtuales de mayor tamaño y un mejor rendimiento que la virtualización basada en hipervisores. Sin embargo, el segundo es ar- gued para ser más seguro que el primero. En este artículo, realizamos un análisis en 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 de Docker son bastante seguros, incluso con la configuración de errores. El nivel de seguridad de los contenedores ers de Docker también podría aumentar si el operador los ejecuta como "sin privilegios" y permite la adiciónde soluciones de protección al en el kernel de Linux, como AppArmor o SELinux.

El trabajo futuro después de este documento podría ser comparar la seguridad de los contenedores de Docker con la de otros sistemas de ización de contenedores o con máquinas virtuales. Estos estudios podrían dar lugar, por ejemplo, a un análisis estático detallado de Docker o a una visión más amplia de la seguridad en los contenedores en general.

Reconocimiento

Este trabajo de investigación es posible gracias a la ayuda y el apoyo de Miika Komu,, Roberto Morabito, Jimmy Kjoll-hombre, y Tero Kauppinen de Nomadiclab.

Referencias

[1] Ansible. [http://www.ansible.com/home/.](http://www.ansible.com/home/) [Consultado el 25 de octubre de 2014].

[2] Contenedores y docker: ¿Qué tan seguros son? [https://blog.docker.com/2013/08/](https://blog.docker.com/2013/08/containers-docker-how-secure-are-they)  [contenedores-docker-how-secure-are-they.](https://blog.docker.com/2013/08/containers-docker-how-secure-are-they) [Accesod 25 de octubre de 2014].

[3] Controlador de lista blanca del dispositivo. [https://www.](https://www.kernel.org/doc/Documentation/cgroups/devices.txt) [kernel.org/doc/Documentation/](https://www.kernel.org/doc/Documentation/cgroups/devices.txt)  [cgroups/devices.txt.](https://www.kernel.org/doc/Documentation/cgroups/devices.txt) [Consultado el 12 de octubre de ber 2014].

[4] Centro de Docker. [https://hub.docker.com/.](https://hub.docker.com/) [Cesssed 30 septiembre 2014].

[5] Docker: configuración de red. [https://docs.](https://docs.docker.com/articles/networking/) [docker.com/articles/networking/.](https://docs.docker.com/articles/networking/) [Cesssed 24 septiembre 2014].

[6] Ebtables. [http://ebtables.netfilter.](http://ebtables.netfilter.org/) [org/.](http://ebtables.netfilter.org/) [Consultado el 25 de octubre de 2014].

[7] Proyecto Kubernetes. [https://github.com/](https://github.com/googlecloudplatform/kubernetes)  [googlecloudplatform/kubernetes.](https://github.com/googlecloudplatform/kubernetes) [Cesssed 10 noviembre 2014].

[8] Proyecto Libcontainer. [https://github.com/](https://github.com/docker/libcontainer)

[docker/libcontainer.](https://github.com/docker/libcontainer) [Consultado el 25 de octubre

2014].

[9] Capacidades de Linux. [http://linux.die.net/](http://linux.die.net/man/7/capabilities)

[hombre/7/capacidades.](http://linux.die.net/man/7/capabilities) [Consultado el 12 de octubre

2014].

[10] LXC. [https://linuxcontainers.org/.](https://linuxcontainers.org/) [Cesssed 30 septiembre 2014].

[11] OpenVZ. [http://openvz.org/.](http://openvz.org/) [Consultado 30

septiembre de 2014].

[12] Espacios de nombres PID en el kernel 2.6.24. [http://lwn.](http://lwn.net/Articles/259217/) [net/Articles/259217/.](http://lwn.net/Articles/259217/) [Consultado el 30 de septiembre- ber 2014].

[13] Títeres. [http://puppetlabs.com/.](http://puppetlabs.com/) [Acceso

18 de octubre de 2014].

[14] SECure COMPuting con filtros. [https:](https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt)

[www.kernel.org/doc/Documentation/ prctl/seccomp\_filter.txt.](https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt) Disponible en: [Consultado el 02 de noviembre de 2014].

[15] Proyecto astillero. [https://github.com/](https://github.com/shipyard/shipyard)

[astillero/astillero.](https://github.com/shipyard/shipyard) [Consultado el 12 de noviembre

2014].

[16] Vagabundo. [https://www.vagrantup.com/.](https://www.vagrantup.com/) [Cesssed 15 noviembre 2014].

[17] ¿Qué es docker? [https://docker.com/](https://docker.com/whatisdocker/)

[whatisdocker/.](https://docker.com/whatisdocker/) [Consultado el 15 de noviembre de 2014].

[18] B. R. Anderson, A. K. Joines, y T. E. Daniels. Mundos Xen: Aprovechando la virtualización en la educación a distancia - tion. En Actas de la 14a Conferencia Anual ACM SIGCSE sobre Innovación y Tecnología en la Educación en Ciencias de la Computación, ITiCSE '09, páginas 293–297, Nueva York, Ny, EE.UU., 2009. ACM.

[19] C. Burniske. Contenedores: La próxima generación de virtu- alization? http://ark-invest.com/webx0/

[21] W. Felter, A. Ferreira, R. Rajamony, y J. Rubio. Una comparación de rendimiento actualizada de máquinas virtuales y contenedores Linux. Technical Informe técnico RC25482 (AUS1407-001), División de Investigación de IBM, julio de 2014.

[22] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker,, y yo. Stoica. Mesos: Una plataforma para compartir recursos en el centro de datos. En Las Actas de la tion 8a Conferencia ence USENIX sobre Diseño e Implementación de Sistemas en Red , NSDI'11, páginas 295–308, Berkeley, CA, EE. UU.,

2011. Asociación USENIX.

[23] D. J. Walsh. ¿Los contenedores acoplables son realmente curativos? [http://opensource.com/business/](http://opensource.com/business/14/7/docker-security-selinux)

[14/7/docker-security-selinux.](http://opensource.com/business/14/7/docker-security-selinux) [Acceso

25 de octubre de 2014].

[24] D. J. Walsh. Traer nuevas características de seguridad a docker.

[https://opensource.com/business/](https://opensource.com/business/14/9/security-for-docker)

[14/9/security-for-docker.](https://opensource.com/business/14/9/security-for-docker) Disponible en: [Consultado el 25 de octubre de 2014].

[25] A. Kivity, Y. Kamay, D. Laor, y U. Lublin. KVM: el monitor de la máquina virtual Linux. En Proceedings of the Linux Symposium, volumen 1, páginas 225–230. 2007.

[26] D. Merkel. Docker: contenedores Linux ligeros para un desarrollo e implementación coherentes. Linux J.,

2014(239), Mar. 2014.

[27] P. Padala, X. Zhu, Z. Wang, S. Singhal, y K. G. Shin.

Evaluación del rendimiento de las tecnologías de virtualización para la consolidación de servidores. LABORATORIOS HP, 2007.

[28] N. Regola y J.-C. Ducom. Recomendaciones para vir- tecnologías de tualización en computación de alto rendimiento - ing. En 2010 IEEE Segunda Conferencia Internacional sobre Tecnología y Ciencia de computación en la nube (Cloud- Com), páginas 409–416, noviembre. 2010.

[29] E. Reshetova, J. Karhunen, T. Nyman, y N. Asokan.

Seguridad de las tecnologías de virtualización de nivel de sistema operativo. En

Actas de la Conferencia NordSec 2014, páginas

77–93, Noruega, 2014.

[30] S. Smalley, C. Vance, y W. Salamon. Implementación

SELinux como módulo de seguridad linux. Informe de NAI Labs

#01-043, Laboratorios NAI, Dic. 2001. Revisado en mayo de 2002.

[31] S. Soltesz, H. Potzl, M. E. Fiuczynski, A. Baviery L. Peterson. Virtualización del sistema operating basada en contenedores: un alternador escalable y de alto rendimiento: tive para los hipervisores. En Actas de la 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, páginas 275–287, EE.UU., 2007. ACM.

[32] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto,

[contenedores-virtualización de próxima generación.](http://ark-invest.com/webx0/containers-next-generation-virtualization) T. Lange, y C. A. F. De Rose. Rendimiento eval-

[Consultado el 22 de noviembre de 2014].

[20] C. Cowan, S. Beattie, G. Kroah-Hartman, C. Pu, P. Wa- gle, y V. Gligor. Subdominio: Seguridad del servidor parsimoniosa. En las Actas del 14o USENIX Confer- ence on System Administration, LISA '00, páginas 355–

368, Berkeley, CA, EE.UU., 2000. Asociación USENIX.

la virtualización basada en contenedores para una alta per-

entornos informáticos de formación. En Actas de la 21a Conferencia Internacional Euromicro sobre Paral- lel, Distributed, y Network-Based Processing, páginas

233–240, Washington, DC, EE.UU., 2013. IEEE Com- puter Society.