Universidad de Buenos Aires

Facultades de Ciencias Económicas, Ciencias Exactas y Naturales e Ingeniería

Carrera de Especialización en Seguridad Informática

Trabajo Final

Seguridad en Virtualización Qemu-KVM

Incluyendo Seguridad en Virtualización Open Source

Autor:
Andrés Villegas

Tutor:
Hernán Segismundo Abbamonte

Entrega: Mayo de 2012

Cohorte 2011

Por medio de la presente, el autor manifiesta conocer y aceptar el Reglamento de Trabajos Finales vigente y que se hace responsable que la totalidad de los contenidos del presente documento son originales y de su creación exclusiva, o bien pertenecen a terceros u otras fuentes, que han sido adecuadamente referenciados y cuya inclusión no infringe la legislación Nacional e Internacional de Propiedad Intelectual.

Andrés Villegas
DNI: 28414686

Resumen

El trabajo emplea una metodología de investigación descriptiva. Se basó en la lectura, análisis y procesamiento del material disponible en la materia, y su posterior composición sumando experiencias propias en el uso de las herramientas en cuestión. Se busca responder qué cuestiones son importantes en la virtualización para asegurar la disponibilidad, integridad y confidencialidad de los datos y los servicios.

El trabajo brinda un baseline sobre cómo implementar de manera segura un entorno de virtualización sobre plataformas Linux Debian Squeeze utilizando el hipervisor Qemu-KVM. Se pretende incluir las mejores prácticas para la implementación y gestión de máquinas virtuales dentro del entorno.

Abarcando todas las posibles vulnerabilidades que generalmente acompañan a la virtualización. Se pretende estudiar todos los puntos relacionados a la seguridad desde el correcto uso de la red, pasando por el control los permisos de las máquinas virtuales, hasta asegurar la correcta publicación de las interfaces para administrar las máquinas virtuales.

A lo largo del trabajo queda evidenciado que para mantener la seguridad en un nivel aceptable en arquitecturas de virtualización Open Source se requiere la implementación de diversas configuraciones y una correcta documentación de los procedimientos para evitar caer en errores relacionados a desconocimiento o distracciones.

Palabras claves: Seguridad, Virtualización, KVM, QEMU, hipervisor, Debian Squeeze.

Índice de contenido

Presentación del tema

La historia de la virtualización se remonta a principios de los años 70 en entornos Mainframe de IBM, en servidores System/360 Modelo 67, en los que se virtualizaba todas las interfaces de hardware del sistema con Virtual Machine Monitor (VMM, hoy conocido como hipervisor), que permitía crear “particiones lógicas” (máquinas virtuales), en las que se ejecutaba una instancia del sistema operativo propietario de IBM. [1]

La evolución de las computadoras, el boom de la arquitectura cliente-servidor y la llegada de los procesadores x86 en la década de los 80, trajo un cambio de paradigmas en el mundo de la informática. Estos procesadores personales, relativamente pequeños encontraron un mercado nuevo, convirtiéndose en estándar de la industria. Los bajos costos de esta arquitectura propuso el uso de un computador para cada servicio, dejando de lado virtualización. Recién a fines de los 90 el crecimiento de los procesadores empezó a evidenciar un desaprochamiento de la capacidad de procesamiento de las computadoras. En los últimos años esto se agudizó con el incremento del número de núcleos y procesadores. Este cambio en las infraestructuras de procesadores x86 impulsó el uso masivo de virtualización en ambientes de servidores independientemente de la plataforma de software y de los sistemas operativos usados.

Debido a la tendencia mundial en el uso de la virtualización, se hace imprescindible el estudio y análisis en profundidad de las posibles brechas de seguridad que esta puede traer aparejada. El presente trabajo se presentarán todas las vulnerabilidades reportadas a la fecha y nuevos riesgos conocidos que han sido introducidos por la plataforma de virtualización KVM corriendo sobre Debian Linux.

¿Qué es Virtualización?

Se podría definir Virtualización como la tecnología que permite ejecutar varios sistemas operativos o máquinas virtuales en una misma máquina física. Esta tecnología puede ser implementada por hardware o por

software, siendo esta última más versátil, flexible y extendida. [1]

En una computadora donde se virtualice van a coexistir dos tipos de sistemas operativos:

- Sistema operativo anfitrión: Es el que manejará el hardware, el que deberá tener los controladores adecuados para interactuar con los componentes instalados. En este se definirá qué recursos serán asignados a cada una de las máquinas virtuales.
- Sistema operativo huésped: Son los sistemas operativos que correrán dentro de las máquinas virtuales definidas en el servidor. Cada uno de estos sistemas operativos interactúa con el hardware físico a través de dispositivos virtuales generado en una capa de virtualización gestionada por el sistema operativo anfitrión. Los recursos de hardware asignados a cada máquina virtual podrán ser modificados según las necesidades, asignándole más memoria, mayor almacenamiento, o bien, quitándole o agregándole interfaces de red. Estas modificaciones podrán realizarse con la máquina virtual apagada o encendida dependiendo de las capacidades de la solución de virtualización implementada.

Teniendo en cuenta que cada sistema operativo asume que tiene el control del hardware sobre el que corre, es evidente que para lograr que los sistemas operativos puedan convivir se requiere de un servidor de virtualización que administre el acceso a la memoria, el uso del CPU y por supuesto las entradas/salidas. Este servidor es una capa de software que interactúa con el sistema operativo anfitrión y con los sistemas operativos huéspedes.

Tipos de Virtualización

Los sistemas de virtualización se pueden clasificar entendiendo tres paradigmas distintos: los sistemas de virtualización completa (Full Virtualization), los sistemas de paravirtualización y los que se conocen como virtualización a nivel de sistemas operativo:

- Full Virtualization: Es el paradigma más utilizado hoy en día. Los sistemas operativos huéspedes desconocen estar siendo virtualizados, por lo tanto no requieren ser modificados. Se utiliza un hipervisor entre las máquinas virtuales y el hardware. Los sistemas operativos huéspedes realizan las peticiones directamente al hardware virtualizado y ante cualquier instrucción que requiera acceso privilegiado el hipervisor la intercepta para su apropiado manejo. Esta arquitectura de virtualización facilita la migración de máquinas virtuales entre diferentes sistemas, e incluso entre diferentes arquitecturas. Esta flexibilidad y versatilidad tiene aparejado una penalización en la performance y el uso de recursos. Ejemplos: VMWare, Qemu-KVM. [3]
- Paravirtualización: Esta tecnología implica una modificación en el sistema operativo huésped que incorpore cambios en los llamados a drivers evitando la necesidad del hipervisor de hacer de proxy entre el sistema operativo huésped y el hardware. En estos sistemas cada máquina virtual actúa con el hardware de manera similar a cualquier otro proceso del sistema. Ejemplos: Xen, LDoms. [1]
- Virtualización a nivel de Sistema Operativo: Este sistema es muy diferente a los dos anteriores. Este tipo de virtualización permite a un sistema operativo crear múltiples entornos de aplicaciones aislados que referencian al mismo kernel. Una ventaja importante que tienen es que su penalización es muy baja. Ejemplos: AIX partitions, Solaris containers. [3]

Procesos y Modos de ejecución

Un proceso es un programa en ejecución. Tradicionalmente, un proceso en Linux tiene dos modos de ejecución: kernel y usuario. El modo usuario (user mode) es un modo de ejecución sin privilegios comúnmente usado por aplicaciones de usuario, o sea cualquier código que no pertenezca al kernel.

El modo kernel (kernel mode) es solicitado cuando una aplicación requiere servicios del kernel, como puede ser escribir a disco. Ante la necesidad de ejecutar una operación de E/S o crear un nuevo proceso hijo, el proceso en modo usuario debe realizar una llamada al sistema para solicitar la acción que requiere ejecutar. [8]

Virtualización Qemu-KVM

El hipervisor Kernel-based Virtual Machine (KVM) es una solución de full virtualization de código abierto que viene incluido con el kernel desde su versión 2.6.20. Está diseñado para hardware x86 que tenga extensiones de virtualización (Intel VT o AMD-SVM); podemos averiguar si disponemos de esas extensiones ejecutando:

```
# egrep '(vmx|svm)' /proc/cpuinfo
```

Si el comando no despliega ninguna salida entonces no se dispone de las extensiones. En algunos casos estas deben ser activadas desde el BIOS para que estén disponibles para usarse.

El requerimiento necesario para convertir el kernel de linux en un hipervisor es simplemente cargar el módulo *kvm.ko*, el cual exporta un dispositivo accesible a través de */dev/kvm.*

Con la carga del módulo, KVM agrega un tercer modo de ejecución llamado huésped (guest mode) el cual tiene dentro, sus propios modos kernel y usuario. Este nuevo modo permite a una máquina virtual tener su propio espacio de direcciones aislada de las del kernel y de las de otras máquinas virtuales.

Cada máquina virtual es un proceso común y corriente del host anfitrión. Este proceso corre en modo huésped permitiendo a la máquina virtual ejecutar todos sus procesos directamente sin interrumpir al kernel del host anfitrión. Y logrando así, acceso directo al CPU mediante las extensiones de virtualización provistas por el hardware. Las únicas operaciones que requieren migrar del modo huésped al modo usuario son las operaciones de E/S requeridas por la máquina virtual.

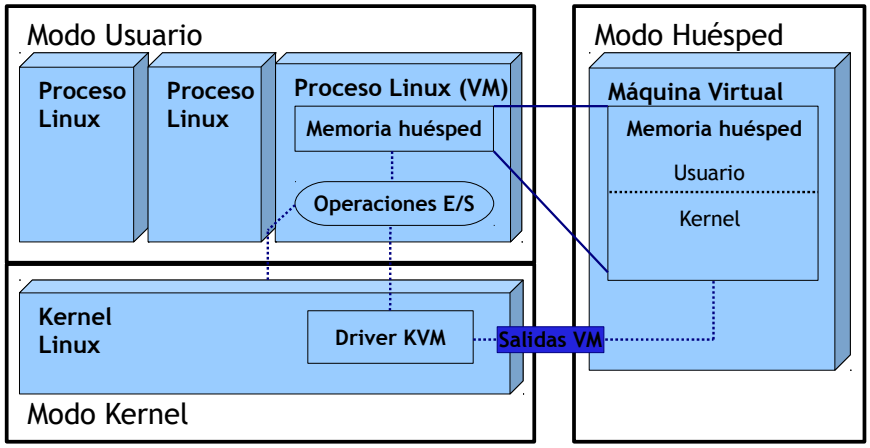

Ilustración 1: Modos de ejecución en máquinas virtuales

Todos los dispositivos del árbol */dev* son comunes para todos los procesos de usuarios. Pero */dev/kvm* es diferente en que cada proceso que la usa ve un mapa de memoria y dispositivos propios, lo que permite asegurar aislamiento entre máquinas virtuales. [2]

Que el hipervisor sea el propio kernel de linux es una gran ventaja, debido a que este se beneficia de las mejoras desarrolladas por la comunidad para el kernel estándar. Por ejemplo, las mejoras relacionadas con el planificador de tareas o con el manejo de memoria beneficiarán tanto al host anfitrión (hipervisor) como al host huésped.

Como se puede ver en el siguiente diagrama, cada máquina virtual es vista desde el host anfitrión como un único proceso:

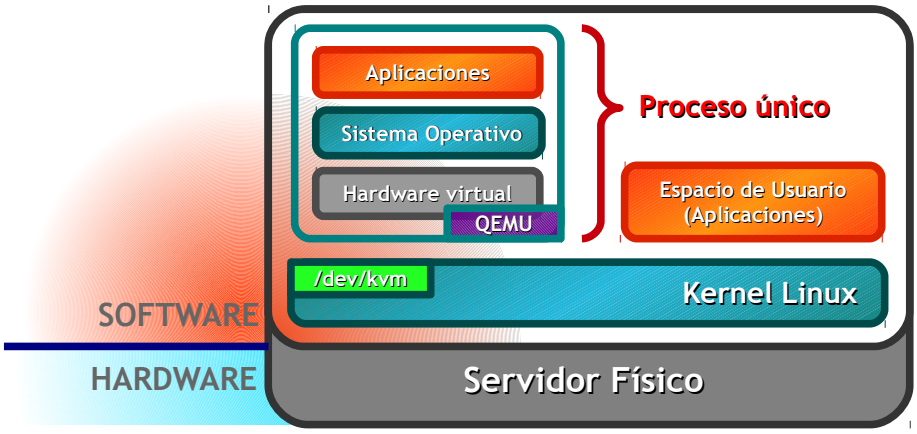

Ilustración 2: Esquema Qemu-KVM

Tipos de control de accesos

El control de accesos es un mecanismos de protección interno que tienen los sistemas operativos multiusuario para proteger los recursos de accesos indebidos.

Unix/Linux históricamente ha usado el esquema de control de acceso discrecional (DAC). Donde generalmente el propietario del recurso o archivo en cuestión, administra los derechos de acceso sobre el mismo y también puede especificar a cual de sus grupos pertenece el recurso. [6]

Sumado a esto, debemos saber que un proceso en ejecución toma los derechos de acceso que tiene el usuario que lo ha invocado. Así un proceso puede modificar o borrar un archivo en particular, si el usuario que lo invoca cuenta con los derechos de acceso suficientes en el recurso. [5]

Siguiendo esta lógica es que el kernel decide si un proceso puede o no leer, escribir o ejecutar un recurso en particular. Aunque debemos saber también que esta lógica es usada para usuarios normales, el super usuario root no sigue esta lógica, ya que es el dueño del sistema operativo y el tiene acceso completo a todos los recursos del sistema.

En contraste con lo anterior existe el control de acceso mandatorio (MAC), implementado en sistemas open sources a través del kernel SELinux. En este tipo de control de acceso cada objeto (archivos o procesos) tiene una etiqueta que representa el nivel de sensibilidad de la información que contiene. Adicionalmente cada usuario tiene asociado un nivel de permisos que indica a que tipos de objetos puede acceder. Estas autorizaciones son especificadas en las políticas SELinux determinado ademas que permisos tendrá al hacerlo. [5]

Seguridad en Virtualización Open Source

La seguridad en virtualización debe ser estudiada desde un punto de vista un poco más amplio que al pensarlo para un servidor sin virtualización. Una brecha de seguridad en una máquina virtual podría traer problemas en otras máquinas virtuales del mismo host físico o incluso podría comprometer el mismo host anfitrión. Así mismo si el host anfitrión es inseguro todas las máquinas virtuales serán vulnerables.

Cuando se analiza la seguridad de un hipervisor se deben tener en cuenta los riesgos introducidos por la plataforma de virtualización, que generalmente estarán relacionados con:

- El nivel de aislamiento entre máquinas virtuales, tanto a nivel de red como de entrada/salida;
- La protección de los accesos a las consolas de las máquinas virtuales, antes física, ahora remota;
- El aseguramiento de los accesos a la gestión y administración del hipervisor.

El presente análisis y sus recomendaciones han sido pensados para escenarios cada vez más comunes en la actualidad, en los que un servidor de virtualización alberga servidores de diferentes zonas de seguridad, por ejemplo: servidores de DMZ, servidores de Intranet y servidores que atienden a Extranets de Partners todos corriendo dentro de un mismo servidor físico.

Como se puede prever la convivencia de diferentes zonas de seguridad aumenta la complejidad de administración y gestión del servidor de virtualización. Por este motivo, todas las configuraciones para aumentar la seguridad deben ser planificadas, analizadas, implementadas y por supuesto documentadas. La documentación de todos los esquemas y configuraciones resultan elementales para una administración cómoda y eficiente.

Además vale destacar que todas las recomendaciones y configuraciones son pensadas para usarse en Linux Debian Squeeze,

aunque estas puede ser adaptadas para usarse en la mayoría de las distribuciones Linux.

Siendo tan amplio y disperso los temas que conciernen al análisis, para abarcar la problemática de seguridad en virtualización en su conjunto, esta se la analizará desde diferentes ángulos:

- La red: se analizará la interacción de las máquinas virtuales y el host anfitrión con el mundo. Se debe tener en cuenta que la proximidad y apareamiento de las máquinas virtuales entre sí, y con el host anfitrión es muy grande. Por lo tanto, el host anfitrión debería ser configurado de modo de aislar las máquinas virtuales de su propia red.
- El almacenamiento: se analizará el manejo de los datos y programas almacenados en medios físicos que usará cada máquina virtual y la propia del host anfitrión. Se debe asegurar el aislamiento de estos datos definiendo exactamente qué puede acceder cada proceso de kvm y restringiendo lo demás, para evitar posibles accesos indebidos.
- El acceso de administración: la creación, modificación y baja de hardware virtual, así como el encendido, migración y apagado de estas máquinas virtuales es una tarea de administración que debe ser asegurada mediante procesos de autentificación y autorización que impidan posibles ataques al hipervisor.
- El acceso a la consola de las máquinas virtuales: Debido a que las terminales de las máquinas huéspedes son consolas virtuales, se debe estudiar la manera mas conveniente de como restringir y controlar el acceso remoto a estas interfaces.

Asegurando la red

A continuación se detallarán todos los temas relacionados con la disponibilidad, integridad y confidencialidad de los datos que viajan por la red en un host anfitrión y por supuestos en sus huéspedes.

Alta disponibilidad de red

Al incrementar la densidad de servicios que dependen de una misma interfaz de red, esta comienza a ser más crítica y se recomienda tomar ciertos recaudos. Esto sucede en la interfaz de red que brinda conectividad a las máquinas virtuales que corren dentro de un mismo host, el aumento de la criticidad de esta interfaz amenaza la disponibilidad de red para todas máquinas virtuales que se albergan. Por este motivo, es recomendable disponer de dos interfaces de red dedicadas para brindar la conexión a todas las máquinas dentro de un mismo equipo.

Para mayor facilidad de administración y una respuesta inmediata de alta disponibilidad lo conveniente es “unir” las interfaces bajo una virtual de modo que esta brinde servicio siempre que, al menos una de las interfaces físicas pueda hacerlo. Esta funcionalidad en linux se llama bonding y para el presente esquema el modo que usaremos es active-backup de modo que sólo una de las interfaces físicas atenderá en un momento y la otra será su backup.

Para poder usar esta configuración deberemos instalar el paquete *ifenslave-2.6* corriendo el siguiente comando:

```
# aptitude install ifenslave-2.6
```

Supongamos que destinaremos las interfaces *eth0* y *eth1* para armar el bond. La configuración de la interfaz bond en */etc/network/interfaces* tendrá que incluir la siguiente definición de la interfaz *bond0*:

```
auto bond0
iface bond0 inet manual
      up ifconfig bond0 0.0.0.0 up
      slaves eth0 eth1
      bond-mode active-backup
      bond-miimon 100
      bond-downdelay 200
      bond-updelay 200
```

Luego debemos dar de alta la interfaz:

```
# ifup bond0
```

Adicionalmente, es recomendable disponer de una tercera interfaz de red conectada a la red de Management o Administración que será utilizada exclusivamente para gestionar la máquinas virtuales de manera remota.

Suponiendo que dicha interfaz es la *eth2*, se recomienda definir las siguientes reglas de firewall para que sumando a las demás prevenciones implementadas incrementen la seguridad en profundidad:

```
# iptables -P INPUT DROP
# iptables -A INPUT -i lo -j ACCEPT
# iptables -A INPUT -m state --state ESTABLISHED,RELATED -j
ACCEPT
# iptables -N admin
# iptables -A admin -p tcp --dport 22 -j ACCEPT
# iptables -A INPUT -i eth2 -s [IP-ADMIN] -j admin
```

El primero de estos comandos define que la política por defecto del firewall para los paquetes de entrada con destino el host físico sean descartados sin enviar ningún aviso de tal acción. El segundo permitirá conexiones desde localhost por la interfaz de loopback, esta regla es necesaria para administrar las maquinas locales usando la terminal de libvirt y poder acceder a la terminal VNC ya sea localmente o a través de un túnel SSH. El tercer comando establece que se acepten los paquetes de conexiones establecidas, por ejemplo respuestas a comunicaciones iniciadas por el host anfitrión. El cuarto y quinto definen una cadena llamada *admin* donde incluiremos todos los servicios que podrán acceder los administradores. Y el sexto deberá repetirse para la IP de cada administrador; y lo que define es que se evalúe la cadena *admin* para todos los paquetes entrantes con destino el host físico desde la dirección IP “IP-ADMIN”.

Para que los cambios en el firewall sean persistentes luego de reiniciar el servidor se deben instalar un paquete llamado *iptables-persistent* almacenar la configuración mediante los siguientes comandos:

```
# aptitude install iptables-persistent
# iptables-save > /etc/iptables/rules
```

Una aclaración al respecto del firewall es que las reglas *iptables* insertadas se han puesto en la cadena *INPUT* y por lo tanto los accesos a las máquinas virtuales no han sido restringidos de ningún modo. Por lo que en caso de ser necesario se debería usar un firewall local en cada máquina virtual.

Más de una zona de seguridad virtualizada

En lo relativo a aislar diferentes zonas de seguridad partiremos del supuesto que la separación de estas, a nivel de red, lo tenemos implementado mediante VLANs. Partimos de esta base debido a que esta configuración es la más común en la actualidad.

Lo primero que debemos hacer es que el servidor de virtualización vea el conjunto de las VLANs de los zonas de seguridad tagueadas en protocolo IEEE 802.1q en ambas interfaces de servicio (*eth0* y *eth1*). Para esto se deben configurar el switch al que están conectadas las interfaces para que en estas bocas se presenten las VLANs correspondientes.

Lo siguiente que se debe hacer es que el servidor se encargue de "desarmar" las VLANs para presentárselas a los diferentes zonas de seguridad unívocamente. Para esto debemos instalar las herramientas que nos permitirán realizar estas tareas, mediante el siguiente comando:

```
# aptitude install vlan bridge-utils
```

Para "desarmar" las VLANs en Linux debemos definir una interfaz por VLAN. Debido que la identificación de las VLANs es numérica si tenemos 2 zonas de seguridad, por ejemplo: DMZ e Intranet con VLANs 10 y 20 respectivamente, definiremos dos nuevas interfaces en */etc/network/interfaces* de la siguiente manera:

```
auto vlan10
iface vlan10 inet manual
      vlan-raw-device bond0

auto vlan20
iface vlan20 inet manual
      vlan-raw-device bond0
```

Luego, debemos dar de alta las interfaces de VLANs:

```
# ifup vlan10
# ifup vlan20
```

Estas interfaces nuevas corresponden cada una a cada VLAN, para poder asociar cada una a las máquinas virtuales a la zona de seguridad que corresponda, debemos crear un bridge por cada zona de seguridad. En Linux los bridge son un concepto similar al de un switch capa 2. Conectan entre si todas las interfaces que sean unidas a cada bridge. Para definir los

bridges debemos incluir en */etc/network/interfaces* las siguientes especificaciones:

```
auto br-dmz
iface br-dmz inet manual
      bridge_ports vlan10
      bridge_fd 9
      bridge_hello 2
      bridge_maxage 12
      bridge_stp off

auto br-intra
iface br-intra inet manual
      bridge_ports vlan20
      bridge_fd 9
      bridge_hello 2
      bridge_maxage 12
      bridge_stp off
```

Seguidamente, debemos dar de alta los bridges ejecutando:

```
# ifup br-dmz
# ifup br-intra
```

A esta altura ya disponemos de una estructura en el servidor de virtualización que de soporte de red a nuestras máquinas virtuales de manera segura. Al momento de definir las interfaces de red que tendrá cada máquina virtual se deberá pegar dicho dispositivo al bridge que le corresponda. Por citar un ejemplo usando *virt-install* para crear una máquina virtual de DMZ, el parámetro network debería tener el siguiente parámetro que indica que la interfaz de la máquina está en DMZ:

```
--network bridge=br-dmz
```

Independientemente de la aplicación usada para crear las máquinas lo importante es saber que todas las especificaciones de la máquina virtual las podemos encontrar en */etc/libvirt/qemu/[nombre-vm].xml*. En este archivo podremos ver qué configuración adoptó la máquina virtual en cuanto a red. Esto es muy importante debido a que la utilización de aplicaciones más sencillas para el usuario muchas veces nos imposibilitan acceder a la granularidad fina de las configuraciones disponibles. Para modificar la configuración fina lo podemos hacer a través de la consola de libvirt:

```
# virsh
virsh # edit [nombre-vm]
```

El comando anterior, abrirá el editor por defecto y podremos ver y modificar cualquier las opciones tanto las especificadas explícitamente y las

tomadas por defecto, incluso dándole un nombre a la interfaz tipo túnel que pertenecerá a la máquina virtual a través del tag <target> dentro de la especificación <interface>:

```
<interface type='bridge'>
      <mac address='52:54:00:f0:78:b4'/>
      <source bridge='br-dmz'/>
      <target dev='tnombrevm'/>
      <model type='virtio'/>
</interface>
```

De este modo cada máquina virtual irá conectada a su zona de seguridad y las zonas de seguridad no tendrán contacto entre sí, salvo que el firewall o router que las una lo permita expresamente. Podemos verificar la construcción de los bridges mediante el siguiente comando:

```
# brctl show
bridge name     bridge id               STP enabled     interfaces
br-dmz          8000.0017a477000c       no              vnet0
                                                        vnet1
                                                        vlan10
br-intra        8000.0017a477000c       no              vnet1
                                                        vnet3
                                                        vlan20
```

Como se ve en la salida del comando anterior *vnet0* y *vnet2* están conectadas a la interfaz *vlan10* en el bridge de DMZ y la interfaces *vnet1* y *vnet3* están conectadas a la interfaz de *vlan20* en el bridge de Intranet.

En la siguiente ilustración se muestra un diagrama aclaratorio de la topología resultante:

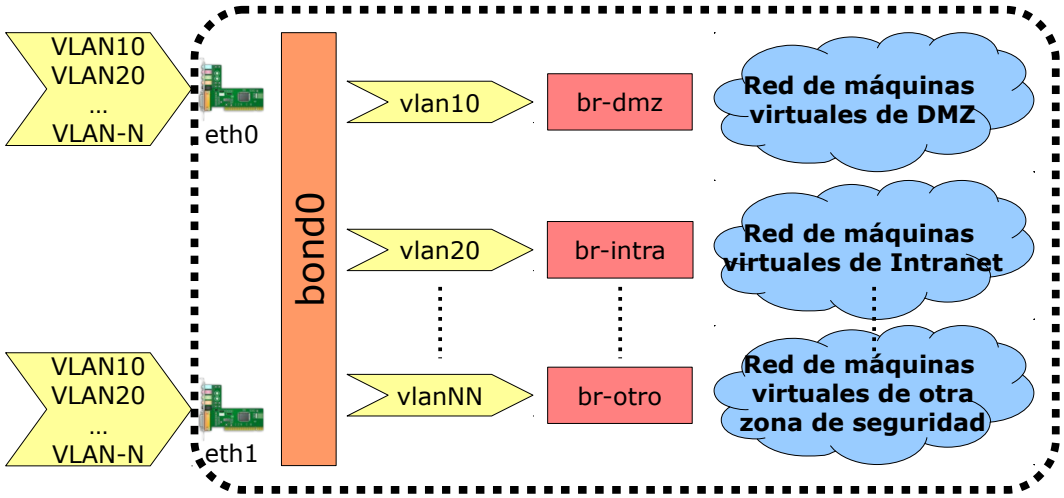

Ilustración 3: Diagrama de redes virtuales

Evitando ataques basados en técnicas de Spoofing ARP

En caso de que una máquina virtual sea comprometida, esta podría ser utilizada para hacer un escalamiento horizontal del ataque. Muchos ataques se basan en el cambio de identidad de host atacante alterando la MAC address de la interfaz de red. Esta técnica es conocida como Spoofing ARP.

Para evitar esto lo que se recomienda hacer es definir reglas de firewall a nivel de capa física (capa 2 del modelo OSI). El objetivo de estas reglas será filtrar todo los paquetes que provengan de una interfaz virtual cuya MAC address no corresponda con la asignada al definir la máquina virtual.

El paquete permite definir este tipo de reglas se llama *ebtables* y no viene instalado por defecto. Por lo tanto, será necesario instalar *ebtables* en el host anfitrión:

```
# aptitude install ebtables
```

Luego suponiendo que tenemos la siguiente definición de red en una máquina virtual:

```
<interface type='bridge'>
      <mac address='52:54:00:92:f4:79'/>
      <source bridge='br-dmz'/>
      <target dev='vnet0'/>
      <model type='virtio'/>
</interface>
```

Instalaremos la siguiente regla:

```
ebtables -A FORWARD -i vnet0 -s ! 54:52:00:92:f4:79 -j DROP
```

Para que estas reglas sean cargadas en cada booteo del host anfitrión debemos configurar el paquete *ebtables*, modificando el archivo */etc/default/ebtables* y definiendo la opción EBTABLES_LOAD_ON_START con el valor “yes”:

```
EBTABLES_LOAD_ON_START=”yes”
```

Y por último, guardaremos las reglas *ebtables* para que sean cargadas en cada inicio del host anfitrión:

```
/etc/init.d/ebtables save
```

Las máquinas virtuales como procesos

Dan Walsh, líder de desarrollo de SELinux en Red Hat, afirma que correr múltiples sistemas operativos en un mismo hardware “es una de las cosas que más miedo dan” desde el punto de vista de la seguridad informática [7]. Previo a la virtualización un host podía ser atacado a través de la red, pero si ahora varios sistemas corren sobre un mismo hardware, la “superficie de contacto” podría ser mucho mayor.

Como se detalla al comienzo del trabajo, Linux ve y trata a cada máquina virtual como un proceso más, este tiene asociado los privilegios del usuario que lo invocó. Ahora, supongamos que se descubre un nuevo bug que posibilita la ejecución de código en el hipervisor desde una máquina virtual. Solamente pensar en las consecuencias que esto podría tener da escalofríos.

Al respecto se pueden tomar algunas medidas que se desarrollan a continuación:

- Un usuario no privilegiado por cada máquina virtual
- sVirt el módulo de SELinux

Un usuario no privilegiado por cada máquina virtual

Debido a que los procesos toman los privilegios del usuario que lo invocó, una posibilidad que de momento es bastante artesanal es crear un usuario por cada máquina virtual a ejecutarse en el servidor. Este nuevo usuario deberá pertenecer al grupo kvm para tener acceso a los recursos asignados y la aceleración que brinda Qemu-KVM a través del módulo */dev/kvm*. Los comandos para crea el usuario son:

```
# useradd [user-vm]
# gpasswd -a [user-vm] kvm
```

Aunque de este modo se obtiene un aislamiento entre máquinas virtuales y se logra que las máquinas virtuales no se ejecuten como root, la inmensa desventaja de esta solución es que se pierde la posibilidad de usar la interfaz libvirt, ya que para correr cada máquina virtual deberemos iniciar

el túnel de red que usará la máquina virtual y pegarla al bridge que corresponda previo levantar la máquina virtual. Esto se debe a que Linux no permite a los usuarios no privilegiados configurar interfaces de red ni modificar los bridges existentes.

Además deberemos iniciar la máquina virtual ejecutando el comando *kvm* y dándole todos los parámetros correspondientes. Debido a la complejidad que esto representa lo recomendable es obtener los parámetros que usa libvirt ejecutando el siguiente comando:

```
# virsh domxml-to-native qemu-argv /etc/libvirt/qemu/[vm].xml
```

Por otro lado, no debemos olvidar que la máquina virtual no sólo debe poder acceder a */dev/kvm* sino también al soporte que será su disco rígido virtual. En muchas ocasiones se utilizará un archivo dentro de alguna partición del host anfitrión, bajo este escenario se deberán ajustar los permisos de ese archivo para que pueda ser leído y modificado por el usuario correspondiente a la máquina virtual y nadie más:

```
# chmod 0600 [discovm.img]
# chown [user-vm]:[user-vm] [discovm.img]
```

En casos más sofisticados, donde el soporte asignado para unidad de disco de la máquina virtual es un Logical Volume del host físico, la situación puede complicarse un poco más. Todos los Logical Volumes por defecto pertenecen a root con grupo disk y permisos de lectura escritura para dueño y grupo (660), lo que excluye de usuarios sin privilegios del uso de los Logical Volumes. Agregar al usuario al grupo disk no sería una solución adecuada debido a que ese usuario podría acceder además a todos los demás Logical Volumes. La solución más apropiada a esta situación es crear una regla del manejador de dispositivos *udev* que asigne los permisos adecuados a cada Logical Volume. Se deberá definir un archivo de regla por cada usuario de cada máquina virtual que use un Logical Volume. Las reglas están ubicadas en */etc/udev/rules.d/*. Por ejemplo, si el usuario es *user-vm1* y el Logical Volume se llama *vm1_dsk* dentro del Volume Group *vms*; el archivo de regla podría llamarse: */etc/udev/rules.d/99-vm1-disk.rules* y contener lo siguiente:

```
SUBSYSTEM=="block",
```

```
KERNEL=="dm-*",
ACTION=="add|change",
ENV{DM_NAME}="vms-vm1_dsk",
OWNER="user-vm1"
```

Luego de modificar las reglas será necesario reiniciar el servicio *udev* y luego desactivar y activar los Logical Volumes del Volume Group:

```
# /etc/init.d/udev restart
# lvchange -an vms
# lvchange -ay vms
```

Este tipo de solución complejiza mucho la administración de las máquinas virtuales. En mi experiencia para poder aplicarla debimos desarrollar una interfaz similar a libvirt que permitiera automatizar y estandarizar todos los procedimientos de gestión del hipervisor.

sVirt el módulo de SELinux

El servicio *sVirt* es un modulo incluido por SELinux para aislar la máquinas virtuales entre sí y brindar un esquema de control de acceso mandatorio (MAC) [4]. Con esta funcionalidad se definen etiquetas únicas, de manera dinámica, para cada proceso que sea una máquina virtual reduciendo los accesos autorizados para cada máquina sólo a sus recursos asignados. De este modo, si una máquina virtual es comprometida, aún pudiendo escalar por sobre la virtualización para ejecutar código en el host anfitrión, las posibilidades de explotar esta situación serán nula.

Lamentablemente esta funcionalidad aún no está madura, ajustada y pulida en Debian Linux, ya que las pruebas realizadas no han sido exitosas. Los inconvenientes encontrados al activar SELinux fueron:

- Presenta un error de falta de permisos al intentar asociar la interfaz túnel al bridge, esto sucede aun iniciando la máquina virtual con el usuario root.
- Luego de iniciada la máquina virtual (sin red por el problema anterior) los discos y el proceso en cuestión, no son asignados con las etiquetas que deberían.

Contrastando los resultados obtenidos con los manuales de Red Hat sobre el tema, es evidente que el desarrollo liderado por Dan Walsh no ha

sido compartido en su totalidad con la comunidad o esta no la ha incorporado aún a los repositorios públicos.

Encriptando la unidad de almacenamiento de los huéspedes

La encripción del disco es una técnica que se suele usar para proteger los datos almacenados en una unidad cuando esta no está en uso o cuando el dispositivo está apagado. Es muy común la encripción de discos en dispositivos portátiles para proteger la confidencialidad de los datos ante un posible extravío.

En el escenario de un hipervisor y sus máquinas virtuales esta técnica podemos usarla para proteger los datos de las máquinas virtuales mientras estas estén apagadas. Supongamos que un atacante logra acceder al hipervisor, si la máquina virtual está apagada los datos de la misma se encontrarán encriptados y por lo tanto inaccesibles. Estos tipos de ataques son conocidos como “Ataques offline” [4].

Para poder encriptar la partición de una máquina virtual vamos a necesitar instalar primero el paquete *cryptsetup*:

```
# aptitude install cryptsetup
```

Luego crearemos la unidad con el espacio requerido que será usada por la máquina virtual. En el ejemplo usamos un Logical Volume, pero podría ser un archivo o una partición física. En este último caso no haría falta crearla sino particionar el disco en cuestión.

```
# lvcreate -L 10G -n vm1-crypt vms
  Logical volume "vm1-crypt" created
```

Usamos *cryptsetup* para darle el formato de encripción a la partición. Nos pedirá una contraseña, la cual debe ser elegida cuidadosamente, debido a que de ella dependerá la seguridad de los datos, y debemos tener en cuenta que la perdida de esa contraseña implicará irremediablemente la perdida de la información contenida en el disco.

```
# cryptsetup luksFormat /dev/vms/vm1-crypt

WARNING!
========
This will overwrite data on /dev/vms/vm1-crypt irrevocably.
```

```
Are you sure? (Type uppercase yes): YES
Enter LUKS passphrase:
Verify passphrase:
```

Ahora debemos abrir el dispositivo encriptado para poder acceder a él, el módulo de Linux, *Device Mapper* generará un nuevo dispositivo a través del cual se podrá acceder a la partición con datos encriptados:

```
# cryptsetup luksOpen /dev/vms/vm1-crypt vm1
Enter passphrase for /dev/vms/vm1-crypt:
```

En este punto ya disponemos de */dev/vms/vm1*, que corresponde al mapeo de la unidad que encuentra desencriptada y permite acceso a la unidad encriptada */dev/vms/vm1-crypt*. Si quisiéramos, después de apagar la máquina virtual, quitar el */dev/vms/vm1* para cerrar el acceso a los datos encriptados, debemos ejecutar:

```
# cryptsetup luksClose /dev/vms/vm1-crypt
```

Esta técnica es recomendable sólo para algunos casos particulares en los cuales sea necesario tener una máquina virtual en la que se hagan tareas esporádicamente y que deba permanecer mayormente apagada. Definiendo para esos casos un procedimiento de encendido y apagado que incluya en cierre de la unidad desencriptada. Un ejemplo de esto podría ser un servidor de Autoridad Certificante de una Infraestructura de Clave Pública, la cual será encendida unicamente para generar nuevas listas de revocación o firmar certificados intermedios.

Para el común de los casos sólo lleva a entorpecer la administración de las máquinas virtuales. La mayor limitante es que no podemos proteger los datos “vivos”, si la máquina virtual está prendida el acceso a los datos es igual que si los datos no estuvieran encriptados.

Por otro lado, el apagado y encendido de las máquinas virtuales deja de ser desatendido. En cada encendido de la máquina virtual deberemos ingresar la contraseña para desencriptar el disco y en cada apagado deberemos cerrar el dispositivo desencriptado. Si luego de apagar la máquina virtual no ejecutamos el *luksClose* la tarea de encriptar los datos habrá sido en vano y los datos permanecerán accesibles. Por este motivo es que se debe documentar muy bien los procedimientos de encendido y apagado de las máquinas virtuales con discos encriptados.

Administración remota del hipervisor

El acceso remoto para administración del hipervisor debe ser analizado en detalle, ya que es un punto crítico dentro de cualquier esquema de virtualización. Su criticidad radica en que, en caso de ser descuidado, un acceso no autorizado amenazaría toda la infraestructura virtualizada.

Dentro de los posibles esquemas de seguridad para el acceso de administración y gestión de máquinas virtuales se detallarán tres formas de acceso remoto [4]:

- Usando túneles SSH
- Usando autentificación SASL y encripción
- Usando TLS con llaves asimétricas

Administración remota a través de túneles SSH

Este modo de acceso remoto aprovecha la seguridad implementada en el protocolo de Secure Shell (SSH). Utiliza la conexión SSH como un túnel, que asegura integridad y confidencialidad dentro de la red, para comunicarse con el hipervisor de manera remota.

Para conectarnos a nuestro servidor de virtualización remotamente y acceder al hipervisor a través de una consola de *virsh*, debemos ejecutar el comando con los siguientes parámetros:

```
$ virsh -c qemu+ssh://root@kvm.master-si.com.ar/system list
root@kvm.master-si.com.ar's password:
 Id Name                 State
----------------------------------
  2 vm1                  running
  4 vm2                  running
 13 vm3                  running
 14 vm4                  running
```

El ejemplo muestra el modo de conexión al usuario root de host kvm.master-si.com.ar y ejecuta el comando *list*. Por supuesto que cualquiera de estos parámetros puede ser modificado. Si no se especifica un comando en particular el programa nos devolverá la consola de *virsh* remota.

Administración remota usando autentificación SASL y encripción

El protocolo SASL es un estándar usado por múltiples aplicaciones que provee autentificación y encripción en conexiones. El servicio define un archivo de base de datos con las credenciales autorizadas. Además puede ser usado en escenarios más complejos usando servicios de autentificación externa como Kerberos o LDAP [4].

Para usar este método de conexión seguiremos los siguientes pasos:

- Paso 1: necesitaremos instalar el paquete de SASL para administrar los usuarios que estarán autorizados a conectarse:

```
# aptitude install sasl2-bin
```

- Paso 2: deberemos ajusta la configuración del servicio libvirt para que escuche en la red, para esto editaremos el archivo de configuración */etc/libvirt/libvirt.conf* y asegurarnos que estén las siguientes opciones:

```
listen_tcp = 1
auth_tcp = “sasl”
```

Adicionalmente será necesario modificar el archivo */etc/default/libvirt-bin* y agregar la el parámetro -l a la siguiente línea:

```
libvirtd_opts="-d -l"
```

Para que los cambios realizados tengan efecto, debemos reiniciar el servicio de libvirt:

```
# /etc/init.d/libvirt-bin restart
```

- Paso 3: definiremos las credenciales del o de los usuarios que tendrán acceso a administrar el hipervisor:

```
# saslpasswd2 -a libvirt admin
Password:
Again (for verification):
```

- Paso 4: configuraremos el firewall para que permita las conexiones entrantes al hipervisor a todos los administradores.

```
# iptables -A admin -p tcp --dport 16509 -j ACCEPT
# iptables-save > /etc/iptables/rules
```

- Paso 5: verificaremos que el servidor quedo correctamente configurado, aceptando y validando las conexiones correctamente:

```
$ virsh -c qemu+tcp://kvm.master-si.com.ar/system list
Please enter your authentication name: admin
Please enter your password:
```

```
 Id Name                 State
----------------------------------
  2 vm1                  running
  4 vm2                  running
 13 vm3                  running
 14 vm4                  running
```

Administración remota usando TLS con llaves asimétricas

Las conexiones TLS (Transport Layer Security) basan su seguridad en encripción basada en certificados x509. Estos certificados son claves asimétricas firmadas por una autoridad certificantes (CA).

Para generar los certificados existen diversas formas, la más básica usando directamente el comando *openssl*. Pero para evitar que los certificados queden en claro en el disco usaremos las aplicaciones *modutil* y *certutil* que brindan un entorno para la generación y almacenado de certificados de manera segura.

A continuación se detallan los pasos a realizar para configurar el servidor de modo que acepte conexiones TLS [4]:

- Paso 1: instalaremos las aplicaciones para administrar los certificados:

```
# aptitude install libnss3-tools openssl
```

- Paso 2: crearemos la base de datos de certificados vacía indicando en qué directorio serán almacenados los certificados:

```
# mkdir ssl-certs
# modutil -create -dbdir ssl-certs/ -force
```

- Paso 3: deberemos inicializar la base de datos, especificando una contraseña:

```
# certutil -N -d ssl-certs/
Enter a password which will be used to encrypt your keys.
The password should be at least 8 characters long,
and should contain at least one non-alphabetic character.

Enter new password:
Re-enter password:
```

- Paso 4: crearemos el certificado de la CA:

```
# certutil -S -d ssl-certs/ -n cacert \
-s "CN=Virt-CA,O=master-si.com.ar,C=AR" \
-t TC,u,u -x -v 12 -1 -2
```

```
Enter Password or Pin for "NSS Certificate DB":

A random seed must be generated that will be used in the
creation of your key.  One of the easiest ways to create a
random seed is to use the timing of keystrokes on a keyboard.

To begin, type keys on the keyboard until this progress meter
is full.  DO NOT USE THE AUTOREPEAT FUNCTION ON YOUR KEYBOARD!

Continue typing until the progress meter is full:

|************************************************************|

Finished.  Press enter to continue:

Generating key.  This may take a few moments...

                0 - Digital Signature
                1 — Non-repudiation
                2 - Key encipherment
                3 - Data encipherment
                4 - Key agreement
                5 - Cert signing key
                6 - CRL signing key
                Other to finish
 > 5
                0 - Digital Signature
                1 — Non-repudiation
                2 - Key encipherment
                3 - Data encipherment
                4 - Key agreement
                5 - Cert signing key
                6 - CRL signing key
                Other to finish
 > 7
Is this a critical extension [y/N]?
y
Is this a CA certificate [y/N]?
y
Enter the path length constraint, enter to skip [<0 for
unlimited path]: >
Is this a critical extension [y/N]?
y
```

- Paso 5: generaremos los certificados para el servidor y el cliente. El Common Name (CN) del servidor deberá coincidir con el nombre de dominio o la IP usada por los clientes para conectarse:

```
# certutil -S -d ssl-certs/ -n servercert \
-s "CN=kvm.master-si.com.ar,O=master-si.com.ar,C=AR" \
-t u,u,u -c cacert -v 12
```

Mientras que el CN del cliente puede ser completamente arbitrario:

```
# certutil -S -d ssl-certs/ -n clientcert \
-s "CN=admin,O=master-si.com.ar,C=AR" -t u,u,u -c cacert -v 12
```

- Paso 6: exportaremos los certificados en formato legible para libvirt:

```
# certutil -L -d ssl-certs/ -n cacert -a > cacert.pem
# certutil -L -d ssl-certs/ -n servercert -a > servercert.pem
# certutil -L -d ssl-certs/ -n clientcert -a > clientcert.pem
# pk12util -d ssl-certs/ -n servercert -o serverkey.p12
Enter Password or Pin for "NSS Certificate DB":
Enter password for PKCS12 file:
Re-enter password:
pk12util: PKCS12 EXPORT SUCCESSFUL
# pk12util -d ssl-certs/ -n clientcert -o clientkey.p12
Enter Password or Pin for "NSS Certificate DB":
Enter password for PKCS12 file:
Re-enter password:
pk12util: PKCS12 EXPORT SUCCESSFUL
# openssl pkcs12 -in serverkey.p12 -nodes \
-nocerts > serverkey.pem
Enter Import Password:
MAC verified OK
# openssl pkcs12 -in clientkey.p12 -nodes \
-nocerts > clientkey.pem
Enter Import Password:
MAC verified OK
root@kvm-libvirt:~# rm -rf serverkey.p12 clientkey.p12
```

- Paso 7: daremos la ubicación definitiva a los certificados:

```
mkdir -p /etc/pki/CA
mkdir -p /etc/pki/libvirt/private
cp cacert.pem /etc/pki/CA/
cp servercert.pem /etc/pki/libvirt/
cp serverkey.pem /etc/pki/libvirt/private/
```

En la máquina cliente deberemos obtener los certificados y ubicarlos:

```
mkdir -p /etc/pki/CA
mkdir -p /etc/pki/libvirt/private
scp kvm.master-si.com.ar:/etc/pki/CA/cacert.pem /etc/pki/CA
scp kvm.master-si.com.ar:clientcert.pem /etc/pki/libvirt
scp kvm.master-si.com.ar:clientkey.pem /etc/pki/libvirt/private
```

- Paso 8: deberemos ajustar la configuración del servidor para que atienda las conexiones TLS. Para esto editaremos los archivos el archivo */etc/libvirt/libvirt.conf* y especificaremos los Distinguished Names (DN) de los certificados que aceptaremos. Luego de modificar el archivo deberían quedar las siguientes opciones:

```
listen_tls = 1
tls_allowed_dn_list = [“C=AR,O=master-si.com.ar,CN=*”]
```

Y también ajustaremos las opciones de inicio del servicio para que abra los puertos de red a la espera de conexiones, esto lo haremos modificando */etc/default/libvirt-bin* y agregando el parámetro -l en la línea:

```
libvirtd_opts="-d -l"
```

Luego, para que los cambios tengan efecto deberemos reiniciar el servicio:

```
# /etc/init.d/libvirt-bin restart
```

- Paso 9: configuraremos el firewall para que permita las conexiones entrantes al hipervisor.

```
# iptables -A admin -p tcp --dport 16514 -j ACCEPT
# iptables-save > /etc/iptables/rules
```

- Paso 10: verificar la conexión desde el cliente:

```
$ virsh -c qemu+tls://kvm.master-si.com.ar/system list
 Id Name                 State
----------------------------------
  2 vm1                  running
  4 vm2                  running
 13 vm3                  running
 14 vm4                  running
```

Administración remota de la consola de las máquinas virtuales

Un punto importante en la virtualización es que desaparecen las consolas puras de las máquinas virtualizadas y estas pasan a ser virtuales también. Qemu-KVM brinda estas interfaces sobre el protocolo VNC para acceso remoto. Por defecto esta opción está deshabilitada y deberá estudiarse en detalle previo a publicar el video de las máquinas virtuales.

El acceso a la consola remota deberá ser protegido de dos maneras: exigiendo una autentificación mediante contraseña y habilitando el acceso al este puerto únicamente desde la estaciones de trabajo de los administradores.

Protegiendo el acceso con contraseña

Como primera medida para asegurar el acceso a la consola de las máquinas virtuales disponemos del uso de una contraseña en el servidor VNC que brinda acceso a dichas terminales. Aunque esto es seguro no es la solución óptima, ya que la limitación del largo de las contraseñas es 8 (ocho) caracteres haciendo que esta contraseña sea potencialmente vulnerable a

ataques de diccionario [4]. Por este motivo debemos elegir una contraseña fuerte que incluya mayúsculas, minúsculas, números y caracteres especiales.

Para configurar esta contraseña debemos editar el archivo */etc/libvirt/qemu.conf* e incluir la siguiente linea:

```
vnc_password = "t!12@uBA"
```

Se debe tener en cuenta que sólo los primeros 8 (ocho) caracteres serán significantes en la definición de la contraseña, los caracteres siguientes serán ignorados tanto aquí como durante al autentificación.

Hasta aquí tenemos un servicio de acceso vía VNC a las máquinas virtuales, al cual sólo se lo puede acceder desde el hipervisor y no remotamente. O osea, en realidad el servicio de acceso está habilitado pero los puertos de VNC escuchan únicamente sobre la interfaz de loopback y significa que sólo pueden ser accedidos desde localhost. Esto podemos verificarlo ejecutando en el hipervisor:

```
# netstat -ltnp | grep kvm
tcp   0   0   127.0.0.1:5900   0.0.0.0:*   LISTEN   2703/kvm
tcp   0   0   127.0.0.1:5901   0.0.0.0:*   LISTEN   3284/kvm
tcp   0   0   127.0.0.1:5902   0.0.0.0:*   LISTEN   3290/kvm
```

La salida del comando anterior muestra que todos los procesos *kvm* escuchan sobre la IP 127.0.0.1, lo que indica que únicamente podrá ser accedida desde localhost. Para acceder desde fuera del servidor existen dos formas: tunelizando VNC sobre SSH o habilitando el acceso remoto. En este último caso es altamente recomendable hacerlo usando TLS y claves asimétricas.

Tunelizando VNC sobre SSH

Para poder conectarse remotamente a un hipervisor que no publica los puertos VNC podemos hacerlo tunelizando el tráfico a través de SSH. Esto podemos hacerlo usando el cliente virt-viewer:

```
$ virt-viewer -c qemu+ssh://root@kvm.master-si.com.ar/system
[nombre-vm]
root@kvm.master-si.com.ar's password:
```

Si no disponemos de un cliente *virt-viewer*, podemos crear el túnel

nosotros mismos. Lo primero que debemos conocer es sobre qué display VNC está la máquina que deseamos ver. Para ello usamos el siguiente comando en el hipervisor:

```
# virsh vncdisplay [nombre-vm]
:2
```

O remotamente:

```
$ virsh -c qemu+ssh://root@kvm.master-si.com.ar/system
vncdisplay [nombre-vm]
:2
```

Conociendo el display VNC de la máquina virtual a cual deseamos acceder, debemos levantar un túnel desde un puerto local al puerto de VNC correspondiente a dicho display. El puerto destino lo obtenemos sumando 5900 al display VNC obtenido. En el ejemplo anterior el display era :2 así que el puerto VNC de la máquina virtual es 5902. Entonces hacemos una conexión SSH que brinde el túnel deseado:

```
$ ssh -L 5900:localhost:5902 root@kvm.master-si.com.ar
root@kvm.master-si.com.ar's password:
Last login: Mon Feb 6 16:26:47 2012 from adm01.master-si.com.ar
root@kvm-libvirt:~#
```

Ahora ya disponemos de un puerto en la PC local que nos permite a través de un túnel encriptado el acceso vía red a un puerto local en el hipervisor. Por lo tanto, podemos conectarnos al puerto local 5900 y estaremos haciendo una conexión remota al hipervisor en su puerto 5902. El puerto 5900 representa el display VNC :0, por lo tanto debemos conectarnos ejecutando:

```
$ vncviewer localhost:0
```

Protegiendo el acceso usando TLS y claves asimétricas

Para habilitar el acceso remoto vía VNC se recomienda hacerlo de modo que este brinde autentificación TLS con certificados x509. Para configurar esto podemos reutilizar los certificados que hicimos para el acceso a la administración del hipervisor. En caso de que quienes administren las máquinas virtuales y el hipervisor sean personas diferentes será importante que los certificados usados sean diferentes. Para generar nuevos certificados se puede seguir los pasos 1 al 6 de creación de

certificados explicado en el apartado "*Administración remota usando TLS con llaves asimétricas*". A continuación se indican los pasos a seguir para configurar este tipo de acceso [4]:

- Paso 1: Copiar el certificado de CA al directorio */etc/pki/libvirt-vnc/*. Si se reutiliza los certificados ya generados podemos hacer un link simbólico al *cacert.pem* instalado anteriormente:

```
# mkdir -p /etc/pki/libvirt-vnc
# cd /etc/pki/libvirt-vnc/
# ln -s ../CA/cacert.pem ca-cert.pem
```

- Paso 2: Copiar certificado del servidor (*server-cert.pem*) y su clave privada (*server-key.pem*) al directorio */etc/pki/libvirt-vnc/*. Si se reutiliza los certificados ya generados podemos hacer un link simbólico a los archivos *servercert.pem* y *serverkey.pem* instalados anteriormente:

```
# cd /etc/pki/libvirt-vnc/
# ln -s ../libvirt/servercert.pem server-cert.pem
# ln -s ../libvirt/private/serverkey.pem server-key.pem
```

- Paso 3: Editar el archivo */etc/libvirt/qemu.conf* para que acepte conexiones VNC en todas las interfaces de red y lo haga usando el protocolo TLS con certificados x509:

```
vnc_listen = "0.0.0.0"
vnc_tls = 1
vnc_tls_x509_cert_dir = "/etc/pki/libvirt-vnc"
vnc_tls_x509_verify = 1
```

- Paso 4: Apagar las máquinas virtuales y reiniciar el servicio libvirt para que los cambios tengan efecto:

```
# virsh shutdown [nombre-vm]
# /etc/init.d/libvirt-bin restart
# virsh start [nombre-vm]
```

- Paso 5: Configurar el firewall para que permita el acceso de los puestos de administración a los puertos necesarios:

```
# iptables -A admin -p tcp --dport 16514 -j ACCEPT
# iptables -A admin -p tcpd --dport 5900:5999 -j ACCEPT
# iptables-save > /etc/iptables/rules
```

- Paso 6: Configurar el cliente. Para comenzar debemos escoger un cliente VNC que soporte autentificación TLS, cada cliente tendrá su propia configuración. En caso de escoger el provisto por *virt-viewer* debemos crear el directorio *~/.pki/CA* y copiar ahí el certificado *ca-*

cert.pem. Si se reutiliza los certificados ya generados podemos hacer un link simbólico al *cacert.pem* instalado anteriormente:

```
$ mkdir -p ~/.pki/CA
$ cd ~/.pki/CA
$ ln -s /etc/pki/CA/cacert.pem ca-cert.pem
```

- Paso 7: Crear el directorio ~/.pki/libvirt-vnc/ y copiar ahí el certificado del cliente (*client-cert.pem*) y su llave privada (*client-key.pem*). Si se reutiliza los certificados ya generados podemos hacer un link simbólico a los archivos *clientcert.pem* y *clientkey.pem* instalados anteriormente::

```
$ mkdir ~/.pki/libvirt-vnc
$ scp root@kvm.master-si.com.ar:/etc/pki/libvirt-vnc/client-
cert.pem ~/.pki/libvirt-vnc/
$ scp root@kvm.master-si.com.ar:/etc/pki/libvirt-vnc/client-
key.pem ~/.pki/libvirt-vnc/
```

- Paso 8: Iniciar la conexión usando *virt-viewer*:

```
$ virt-viewer -c qemu+tls://kvm.master-si.com.ar/system
[nombre-vm]
```

Conclusión

Como se detalló a lo largo del trabajo, las variantes de configuración posibles son muchísimas. Esto exige a los administradores de este tipo de hipervisores un compromiso mayor con la seguridad ya que, como se vio, un error en la configuración podría comprometer la confidencialidad, integridad y disponibilidad de los datos y servicios alojados en el hipervisor y en las máquinas virtuales. También es imprescindible conocer los nuevos riesgos introducidos con KVM para poder minimizarlos o llevarlos a un nivel aceptable para nuestra infraestructura.

Aunque las herramientas a nivel de red y administración remota están preparadas para dar un soporte acorde a las necesidades de seguridad actuales, quedo evidenciado en las pruebas realizadas que todavía a nivel de procesos y privilegios de usuario del hipervisor hay muchos interrogantes por responder. Según la teoría y los manuales de Red Hat Enterprise, SELinux brinda una solución importante a través del modulo sVirt, que daría las respuestas pendientes en Debian Linux. De cualquier modo, un desarrollo más a conciencia y sin grandes cambios del módulo de administración de las máquinas virtuales puede permitirnos correr cada máquina virtual con un usuario diferente y de ese modo obtener un aislamiento más que aceptable sin caer en un modelo de control de acceso mandatorio. Sin dudas es en este punto dónde aún queda mucho camino por recorrer, el presente trabajo no es más que un puntapié para un análisis más en profundidad sobre distintas alternativas de control de acceso y aislamiento de procesos que ayuden a minimizar los nuevos riesgos.

El sistema de virtualización estudiado es muy “joven” y está en pleno desarrollo, hoy en día impulsado por ser el sistema de virtualización recientemente elegido por Red Hat, desplazando su anterior solución Xen. Debido a esto la comunidad que desarrolla Open Source posiblemente será altamente beneficiada del código que pueda brindar Red Hat.

Más allá de los esfuerzos de los desarrolladores de la comunidad y de Red Hat, la responsabilidad de velar por la seguridad de los servidores que administremos es nuestra. Por este motivo, es muy importante mantener los

hipervisores actualizados al día y seguir atentamente los reportes de seguridad. En este caso, DSA (*Debian Security Announce*) es el más indicado y en el que encontraremos todas las novedades al respecto.

Por experiencia propia puedo asegurar que además de las configuraciones iniciales recomendadas a lo largo del trabajo, para maximizar la seguridad es fundamental la correcta documentación y revisión de los procedimientos para adaptarlos a este nuevo paradigma.

Bibliografía específica

[1] Antonio Ángel Ramos, Jean Paul García-Morán, Fernando Picouto, Jacinto Grijalba, Maikel Mayan, Ángel García, Eduardo Inza y Carlos Alberto Barbero, Instala, administra, securiza y virtualiza ENTORNOS LINUX,. Editorial Alfaomega Ra-Ma, 2009

[2] Discover the Linux Kernel Virtual Machine IBM, http://www.ibm.com/developerworks/linux/library/l-linux-kvm/ (consultada el 11/10/2011)

[3] Evi Nemeth, Garth Snyder, Trent R. Hein y Ben Whaley, Unix And Linux System Administration Handbook (Fouth Edition), Editorial Prentice Hall, 2011.

[4] IBM Blueprint: KVM Security First Edition (December 2011), http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaat/liaatsecurity_pdf.pdf (consultada el 25/09/2011)

[5] IBM Blueprint: KVM Tunning KVM for performance (November 2010), http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaat/liaattuning_pdf.pdf (consultada el 25/09/2011)

[6] Kernelthread.com: A Taste of Computer Security by *Amit Singh* http://www.kernelthread.com/publications/security/ac.html (consultada el 01/04/2012)

[7] LinuxCon: Secure virtualization with sVirt (consultada el 02/12/2011)

[8] Qumranet: KVM: Kernel-based Virtualization Driver Whitepaper, http://www.linuxinsight.com/files/kvm_whitepaper.pdf (consultada el 11/10/2011)

Bibliografía general

- @sec: KVM Security Comparison, http://www.redhat.com/f/pdf/rhev/kvm_security_comparison.pdf (consultada el 11/10/2011)
- Best Practices for KVM and Red Hat Enterprise Linux on NetApp Storage, http://media.netapp.com/documents/tr-3848.pdf (consultada el 11/10/2011)
- Creating and Controlling KVM Guests using libvirt, heprc.phys.uvic.ca/sites/heprc.phys.uvic.ca/files/reports/vliet-wtr.pdf (consultada el 27/11/2011)
- IBM Blueprint: Best practices for KVM First Edition (November 2010), http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaat/liaatbestpractices_pdf.pdf (consultada el 25/09/2011)
- IBM Blueprint: First Steps with Security-Enhanced Linux (SELinux) http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaai/selinux/liaaiselinux_pdf.pdf (consultada el 01/04/2012)
- KVM as learning tool Intel, http://www.fceia.unr.edu.ar/lcc/jcc/2009/kvm-as-a-learning-tool.pdf (consultada el 22/10/2011)
- KVM Security - Where Are We At, Where Are We Going, Klaus Heinrich Kiwi, IBM en LinuxCon Brazil 2010 presentation, http://blog.klauskiwi.com/archives/91 (consultada el 22/10/2011)
- Red Hat Enterprise Linux 6.2 Beta Security Guide (2011), http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6-Beta/pdf/Security_Guide/Red_Hat_Enterprise_Linux-6-Beta-Security_Guide-en-US.pdf (consultada el 22/10/2011)
- Red Hat Enterprise Linux 6.2 Beta Security-Enhanced Linux (2011), http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6-Beta/pdf/Security-Enhanced_Linux/Red_Hat_Enterprise_Linux-6-Beta-Security-Enhanced_Linux-en-US.pdf (consultada el 22/10/2011)
- Red Hat Enterprise Linux 6.2 Beta Virtualization Administration Guide (2011), http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6-

Beta/pdf/Virtualization_Administration_Guide/Red_Hat_Enterprise_Linux-6-Beta-Virtualization_Administration_Guide-en-US.pdf (consultada el 22/10/2011)

- sVirt: Hardening Linux Virtualization with Mandatory Access Control, James Morris, Red Hat Security Engineering (2009), http://namei.org/presentations/svirt-lca-2009.pdf (consultada el 22/10/2011)