Kernel Virtual Machine (KVM)

KVM security

Kernel Virtual Machine (KVM)

KVM security

Note

Before using this information and the product it supports, read the information in "Notices" on page 49.

Second Edition (April 2012)

Contents

KVM security

You can protect and secure the Kernel-based Virtual Machine (KVM) environment by deploying KVM security features, such as configuring network isolation, securing storage devices, configuring secure remote management, isolating virtual machines with the sVirt service, preventing denial-of-service situations with control groups (cgroups), and protecting data at rest with disk-image encryption.

Notes:

- Although all examples shown in this document were tested in a Red Hat Enterprise Linux 6.2 environment, some tools used are also available in SUSE Linux Enterprise Server 11 SP1.
- The paths used in the examples might vary between distributions.
- Red Hat Enterprise Linux 6.2 uses Yum as a package manager, and SELinux as its default Linux security module (LSM).
- SUSE Linux Enterprise Server 11 SP1 uses YaST as a package manager, and AppArmor as its default LSM.
- For more information about specific tools and paths, see your distribution documentation.

Related information:

KVM overview

Host security

Learn about the trusted computing base (TCB), how to configure the network to isolate the host and guest operating systems, and how to customize the storage location for storage devices.

Trusted computing base

The *trusted computing base (TCB)* is the combination of hardware and software in a computer system that enforces a unified security policy.

The TCB usually contains components that are critical to the security of the system, such as hardware, firmware, a security policy, and other components. The TCB controls and authenticates access to system resources and verifies system integrity. In a KVM environment, the overall TCB includes the host TCB, KVM, and QEMU.

The type of the hypervisor does not influence the security quality of the hypervisor. A type 1 hypervisor can be more secure than a type 2 hypervisor and a type 2 hypervisor can be more secure than a type 1 hypervisor. Instead, the TCB of the hypervisor determines the security quality of the hypervisor. More specifically, the size, complexity, design, and implementation of the TCB determine the security quality of the hypervisor. For example, a large hypervisor with a quality design can be more secure than a small hypervisor with a poor design. However, as the size and complexity of the TCB increases, the difficulty of determining the quality of the design and implementation also increases. This difficulty increases exponentially with the amount of code that must be trusted. Therefore, to achieve maximum security, most operating systems reduce the size and complexity of the TCB as much as possible.

The size of the TCB directly affects the security quality of the hypervisor. The larger the TCB, the more bugs the TCB likely has and the less secure the hypervisor. To further reduce the size of the TCB in KVM, you can minimize the amount of code that runs in the host operating system. For example, you can disable any network-facing daemons that run in the host operating system.

Another reason to reduce the size and complexity of the TCB is to aid the feasibility of formal certification, such as the Common Criteria certification. The size of the TCB directly affects the cost of the

TCB assurance process. The larger the TCB, the more costly the process. Currently, Red Hat is pursuing Common Criteria at Evaluation Assurance Level (EAL) 4+ for Red Hat Enterprise Linux 6. This effort includes certifying the KVM hypervisor on both Red Hat Enterprise Linux 5 and Red Hat Enterprise Linux 6.

Related information:

Red Hat Enterprise Linux 6, KVM to Pursue Security Certification

Host network configuration

Learn how to isolate the host from the guest operating systems and how use the network functions of KVM to isolate the guest operating systems from each other.

Host network isolation

You can increase network security by configuring one network interface for the host and a separate network interface for the guest operating systems.

Typically, tasks that you perform from the host, like starting or stopping virtual machines, require a high security clearance. Generally, there are a small number of trusted users with a high security clearance. Tasks that you perform from guest operating systems require a lower security clearance. Generally, there are a large number of users with a lower security clearance.

To increase the security of the host, configure one network interface for the host and a separate network interface for the guest operating systems. In this configuration, the network traffic for the host travels on a different subnet than the network traffic for the guest operating systems. This configuration increases security in the following ways:

- Helps isolate the host from the guest operating systems.
- Helps prevent malicious users with a lower security clearance from breaching a guest operating system and attacking the host or other guest operating systems.

Network isolation options

A standard network security practice is the use of firewalls to block unwanted traffic between two different networks. These firewalls are often applied at the borders of networks as a way to filter potential security threats coming from untrusted sources. However security threats can also originate from behind the firewall when a trusted host is compromised.

Networked environments can be protected from internal threats in various ways: from applying firewalling internally to using VLAN segmentation and intrusion detection systems. A virtualized environment is susceptible to internal threats as any physical environment would be, but with a fundamental difference: shared network ports.

One of the key points in virtualization is sharing resources, and network ports are no exception. From the network security and management standpoint, this means having additional active network elements inside each virtualization host that can share its network ports.

Linux networking stack implementation allows the KVM host to act as a simple layer 2 bridge (that is, Ethernet switch), a forwarding or NAT router, a stateful firewall, or any combination of those roles.

This section covers the most common practices for secure virtual network interface sharing. It also covers network filtering with libvirt.

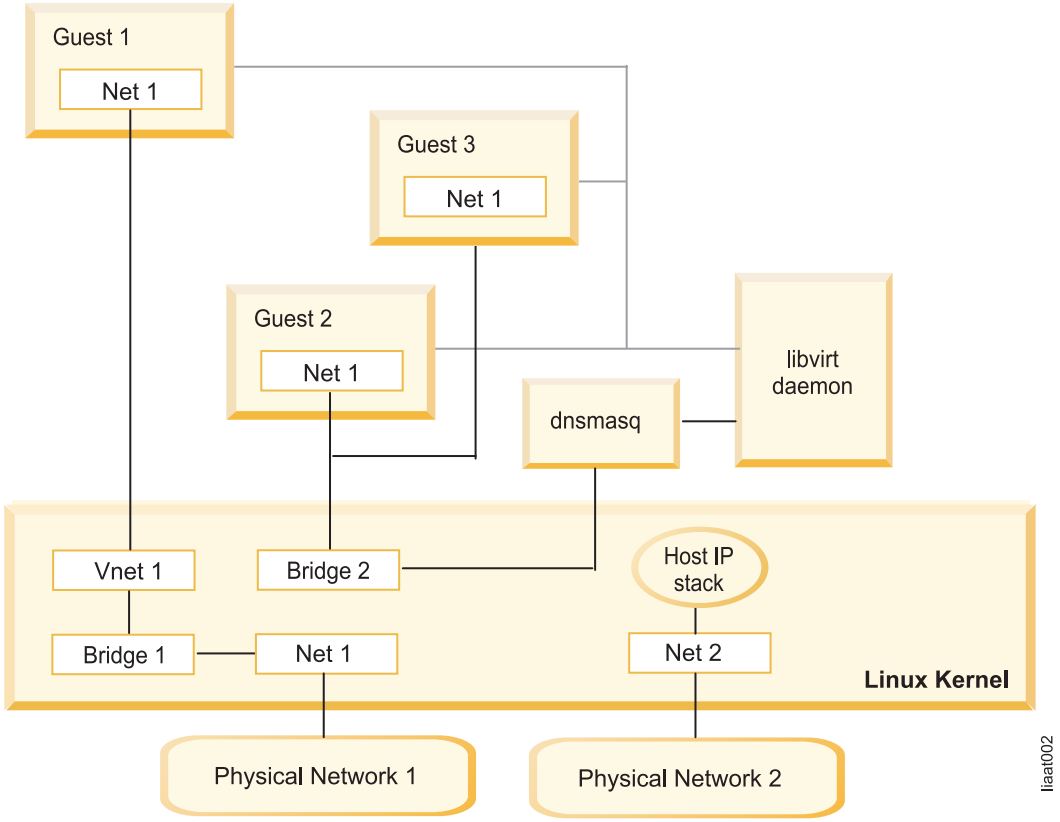
Network port sharing with Ethernet bridges:

The most straightforward way of sharing a network connection, even in the physical world, is by extending the number of ports in this network using an Ethernet hub or switch (a bridging device). In fact, bridging devices are at the core of bus-topology networks such as Ethernet.

After implementing network port sharing with Ethernet bridges, assign a separate IP address to the guest operating system for its network connection to work.

Overview of Ethernet bridges:

Networking in a virtualized environment such as Qemu and KVM can be seen as a number of independent physical and virtual network interfaces. In the Linux KVM Host, a *bridge* is a pseudo-network interface that forwards layer 2 packets back and forth.



This means that a physical network connection can act as an uplink to a virtual Ethernet switch inside the KVM host system. Each virtual-machine guest and the host should also have different MAC addresses and different IP addresses. Having different MAC addresses and IP addresses means that guests rely on external entities (in the physical network) to act as gateways and DNS servers (and optionally DHCP servers). Valid IP addresses must be assigned to guest interfaces and are seen in the physical network the same way that physical hosts are.

For more information about configuring a Linux bridge in your KVM host, see the "(Optional) Setting up a network bridge in a host" section of the *Quick start guide to installing and configuring KVM* blueprint at http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaai/kvminstall/liaaikvminstallbridge.htm.

Bridged networking and Netfilter:

It is important to note that, by default, Linux enables Netfilter processing even in bridged traffic.

Netfilter, although allowing the use of iptables to create layer 3 filtering rules, is also seen as a security risk for guest isolation because Netfilter processing sometimes occurs on a *global context* (without distinction of a source layer 2 port). In practice, with Netfilter processing enabled, an attacker in a

compromised guest might mix malicious IP traffic with legitimate ones from other guests. This processing can also result in a minor performance impact. Therefore it is recommended that Netfilter processing is disabled.

Configuring KVM guests to use Linux bridge:

After you configure your Linux bridge, attach virtual machines to the bridge and share the physical network port by editing any configured guests to use the configured bridge interface as a source bridge.

Before you start, ensure that each guest operating system has an IP address or fully qualified domain name.

To configure KVM guests to use a Linux bridge, complete the following steps:

1. Edit your guest domain definition using the **virsh edit** command:

```
# virsh edit guest01

    <disk type='file' device='cdrom'>
      <driver name='qemu' cache='none'/>
      <target dev='hdc' bus='ide'/>
      <readonly/>
    </disk>
    <interface type='bridge'>
      <mac address='54:52:00:5b:1a:cd'/>
      <source bridge='br0'/>
    </interface>
    <serial type='pty'>
      <source path='/dev/pts/1'/>
      <target port='0'/>
    </serial>
    <console type='pty' tty='/dev/pts/1'>
      <source path='/dev/pts/1'/>
      <target port='0'/>
    </console>
  </devices>
</domain>
```

   Note: It is important that each KVM guest has a unique MAC address. Otherwise, it might cause traffic disruptions to all hosts in the same subnet. To generate a new, random MAC address for a guest network interface, edit the guest XML definition, removing the **mac** tag for that network interface.

   For more information about managing KVM guests using an XML definition file, see "Creating an XML definition file for your KVM" in the *The developer's approach to installing and managing KVM* blueprint at http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaai/kvmadv/kvmadvguestxml.htm.

   The following is the original network configuration:

```
# virsh dumpxml guest01

<interface type='network'>
  <mac address='54:52:00:5b:1a:cd'/>
  <source network='default'/>
</interface>
```

2. Restart each modified guest for changes to take effect and test whether the network is working properly.

Restricting MAC address spoofing:

The MAC address of the guest network device can be changed by the root user in the guest operating system. MAC address spoofing is used when attacking an external network switch that is vulnerable to MAC flooding. The **ebtables** package can be used in the KVM host to enforce a single MAC address for each guest network device.

Before you start, ensure that each guest operating system has an IP address or fully qualified domain name.

To restrict MAC address spoofing in KVM environments, complete the following steps:

1. Verify that the network going out is working properly by pinging another host:

   ```
   # ping 132.8.234.138
   PING 132.8.234.138 (9.8.234.138) 56(84) bytes of data.
   64 bytes from 132.8.234.138: icmp_seq=1 ttl=255 time=0.509 ms
   64 bytes from 132.8.234.138: icmp_seq=2 ttl=255 time=0.458 ms
   ```

2. Add the device name of the network tunnel (TUN) device and identify the MAC address used by the guest domain.

   Note: Specify a target device that does not start with "vnet" or "vif", because those prefixes conflict with libvirt automatic naming scheme and are ignored.

   ```
   $ virsh edit guest01
   <interface type='bridge'>
     <mac address='54:52:00:5b:1a:cd'/>
     <source bridge='br0'/>
     <target dev='virtnet'/>
   </interface>
   ...
   ```

3. Use the **ebtables** command to prevent frames leaving the TUN device (virtnet1, in this example), from having an unknown MAC address by running the **ebtables -A FORWARD -i <*TUN device name*> -s ! <*MAC address*> -j DROP** command:

   ```
   $ ebtables -A FORWARD -i virtnet1 -s ! 54:52:00:5b:1a:cd -j DROP
   ```

   If the MAC address of the guest network device is modified, all outgoing traffic is dropped in the KVM host.

4. Save the new **ebtables** rule:

   ```
   $ service ebtables save
   Saving Ethernet bridge filtering (ebtables):              [  OK  ]
   ```

5. Verify that the **ebtables** rule is working:

   a. Log in to your KVM guest. Run the **ifconfig <*device*> hw ether <*fake MAC address*>** command to spoof its MAC address:

      ```
      # ifconfig eth0 hw ether 54:52:12:33:44:55
      ```

   b. Spoofing is working if the **ifconfig** command returns the fake MAC address instead of the real MAC address of your device:

      ```
      #ifconfig eth0
      eth0      Link encap:Ethernet  HWaddr 54:52:12:33:44:55
                inet addr:9.8.234.237  Bcast:9.8.234.255  Mask:255.255.255.128
                inet6 addr: fe80::5652:ff:fedd:f5c6/64 Scope:Link
                UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
                RX packets:1412 errors:0 dropped:0 overruns:0 frame:0
                TX packets:475 errors:0 dropped:0 overruns:0 carrier:0
                collisions:0 txqueuelen:1000
                RX bytes:91671 (89.5 KiB)  TX bytes:62849 (61.3 KiB)
                Interrupt:11 Base address:0x4000
      ```

   c. Pinging from your guest will cease to work because ebtables is in action:

```
# ping 132.8.234.138
PING 132.8.234.138 (9.8.234.138) 56(84) bytes of data.
From 132.8.234.237 icmp_seq=2 Destination Host Unreachable
From 132.8.234.237 icmp_seq=3 Destination Host Unreachable
From 132.8.234.237 icmp_seq=4 Destination Host Unreachable
```

Network port sharing with 802.1q VLANs:

Learn about 802.1q VLANs and how to configure 802.1q VLANs in KVM environments.

Virtual LANs:

A virtual LAN (VLAN), as specified by the IEEE 802.1q standard, is a method for segregating network traffic within a bridged LAN infrastructure.

VLANs were originally designed specifically for physical environments to allow two logically separated networks to use the same physical medium. VLAN segmentation, also called 802.1q tagging, works by appending a tag identifying the VLAN ID to each Ethernet frame.

VLAN usage in the KVM virtualization scenario can be seen as an extension to the simple bridge interface sharing. The difference lies in which interface participates in the bridge set. In the standard mode of operation (as seen in the examples in "Network port sharing with Ethernet bridges" on page 2), the physical interfaces (such as *eth0*, *eth1*...) are bound to the bridge, which is used by each guest. These interfaces carry unmodified packets coming externally or being generated internally, with or without a VLAN ID tag.

It is possible to filter out every package not carrying a particular VLAN ID by creating subinterfaces. These subinterfaces become part of the VLAN defined by a specific VLAN ID.

Applying this concept to the bridged interface sharing method involves replacing the bound physical interface by a subinterface that is part of a particular VLAN segmentation. This way, every virtual machine guest with interfaces bound to this bridge is part of that particular VLAN. Like in the simple Ethernet bridge environment, the network provided is transparent.

Configuring 802.1q VLANs:

Complete the steps to configure VLAN segmentation for KVM guests.

Before you start, perform the following tasks:

- Ensure that each guest operating system has an IP address or fully qualified domain name.
- Ensure that the host and guest operating systems are connected to a VLAN-capable network switch and infrastructure.

In enterprise environments, most of the existing network infrastructure supports 802.1q VLANs. Its use in the KVM virtualization environment is a good compromise for the following reasons:

- Low setup cost in the KVM host – no need for firewall or routing between guests
- Use of existing network access controls, methods, and processes – after the KVM Host administrator assigns a virtual machine guest to a VLAN, access control can be made external through the traditional network administrator role

1. Identify the VLAN IDs that to assign to each guest.
2. Explicitly configure the external network infrastructure to allow traffic from those VLANs to the KVM host:
   a. Configure the network switch connected to the KVM Host.
   b. Qualify the physical port on the host as a *trunk* (carries multiple VLANs) and a *tagged* (accepts tagged frames) port.

c. Allow traffic to necessary VLAN IDs.

3. Create the virtual bridge in the KVM Host. Avoid mixing different VLANs in a single bridge.
4. Create a file named **ifcg-<*name*>** in the `/etc/sysconfig/network-scripts` directory to create a permanent bridge configuration, where *<name>* is the bridge interface name. The following example specifies a br_v19 bridge with a file named `/etc/sysconfig/network-scripts/ifcfg-br_v19`:

```
DEVICE=br_v19
TYPE=Bridge
BOOTPROTO=static
STP=yes
ONBOOT=yes
DELAY=0
```

Note: The ONBOOT=yes line assures that the bridge will be available automatically after each boot. No IP address is to this bridge and Spanning Tree Protocol (STP) is enabled (STP=yes).

5. If there are multiple guests participating in the same VLAN ID (even if they use separate bridges), disable Netfilter processing in bridging devices by appending the following lines to the `/etc/sysctl.conf` file:

```
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.bridge.bridge-nf-call-arptables = 0
```

6. Reload the kernel parameters with the **sysctl** command:

```
# sysctl -p
net.ipv4.ip_forward = 0

. . .
net.bridge.bridge-nf-call-ip6tables = 0
net.bridge.bridge-nf-call-iptables = 0
net.bridge.bridge-nf-call-arptables = 0
```

7. Configure one or more subinterfaces from the main, physical network interface (the trunk). The following example configures the subinterface eth0.19 that is assigned to VLAN ID 19. The file name is `/etc/sysconfig/network-scripts/ifcfg-eth0.19`.

```
#VLAN 19 in trunk eth0
DEVICE=eth0.19
VLAN=yes
ONBOOT=yes
BRIDGE=br_v19
```

Note: The value of DEVICE (that is, the subinterface name), is in the form of *<interface-name>.<VLAN-ID>*, where *interface-name* is the name of the physical interface that this virtual bridge attaches to (*eth0* in this example) while the *VLAN-ID* is directly taken from the subinterface name (*19* in this example).

The bridge strips the VLAN tags from ingress traffic and assign tags to egress packets.

As ONBOOT is set to yes, the subinterface of the virtual bridge opens automatically on every reboot, but the administrator can also bring them up manually by using the **ifup** command:

```
# ifup br_v19
# ifup eth0.19
# brctl show
bridge name bridge id    STP enabled  interfaces
br0   8000.00145ed87f4a  yes   eth0
virbr0  8000.000000000000  yes
br_v19  8000.00145ed87f4a  yes   eth0.19
```

This example also shows the use of the **brctl** command to list all of the virtual bridges and their assigned interfaces.

8. With the bridge interface running, adjust each guest configuration, assigning interfaces to their respective bridge or VLAN (as described in "Configuring KVM guests to use Linux bridge" on page 4).

```
<interface type='bridge'>
  <mac address='54:52:00:5b:1a:cd'/>
  <source network='br_v19'/>
  <target dev='vnet0'/>
</interface>
```

9. Restart the modified guests for changes to take effect.
10. Assign a separate IP address to the guest operating system for its network connection to work.

The libvirt network filter driver:

The libvirt network filter driver provides configurable network filtering by using ebtables and iptables. Filters are defined in XML format. Filters are associated with individual guest NICs in libvirt XML domain definitions.

The following shows an example of a filter rule definition:

```
<filter name='mysecfilter' chain='root'>
  <rule action='drop' direction='out' priority='500'>
    <tcp dstportstart='25' dstportend='25'/>
  </rule>
  <rule action='drop' direction='in' priority='500'>
    <tcp dstportstart='23' dstportend='23'/>
  </rule>
  <filterref filter='clean-traffic'/>
</filter>
```

The filter element name attribute identifies the filter rule. The chain attribute groups filter rules, similar to iptables chains. One or more rule elements define the network controls. Zero or more filterref elements include additional filter rules in the filter.

Inside a rule element, you can define the action to be taken when the rule is matched, the direction of traffic, and the priority of this rule compared to others. You can also define the specific criteria to match the rule, including protocol and protocol-specific parameters.

You can also use predefined network filter rules. You can display them with the **virsh nwfilter-list** command. In Red Hat Enterprise Linux 6.2, the following predefined network filter rules are available:

```
# virsh nwfilter-list

UUID                                   Name
----------------------------------------------------------------
dfc0da09-9562-4103-4cfb-592afbdacfea  allow-arp
f4143aa2-0288-143c-3839-0b971737ddeb  allow-dhcp
dfbec98d-9aad-93a3-70ab-58efe32b76be  allow-dhcp-server
c3a5fd24-6f69-dc91-9f2d-f4943161e3e8  allow-incoming-ipv4
8bcf098d-8586-c4e8-ed3c-1c0186c0429e  allow-ipv4
334e22d3-a9a8-5dcf-e5a0-ae79eabc5a90  clean-traffic
f88f1932-debf-4aa1-9fbe-f10d3aa4bc95  no-arp-spoofing
c4c5c411-3217-761d-19a4-9bfb24e2951c  no-ip-multicast
3bca6dc0-61f2-d0cd-f4c2-c44a11f9cbb0  no-ip-spoofing
07714833-c1d0-ede7-713e-31ff9688840d  no-mac-broadcast
4b3c542f-ddd4-9795-2bf0-f8b34868a5d6  no-mac-spoofing
b620b0ec-f59c-3d0a-ac39-d7146498badf  no-other-l2-traffic
bd624d16-9a5b-dc7b-4e92-8337a2a92a8a  no-other-rarp-traffic
fb990e75-07a5-f304-5a4d-3ff287427133  qemu-announce-self
22bc6be0-9319-63be-7fd8-ae93ad1bab83  qemu-announce-self-rarp
```

You can use the no-mac-spoofing rule to obtain the same effect as if you were following the instructions in the Restricting MAC address spoofing topic.

The clean-traffic rule is an aggregate of best-practice rules, like no-mac-spoofing, no-ip-spoofing, and no-arp-spoofing. You can show the rules aggregated by the clean-traffic rule by using the **virsh nwfilter-dumpxml** command:

```
virsh nwfilter-dumpxml clean-traffic
```

For more information about the libvirt network filter driver, see the "The network filter driver" section of *Firewall and network filtering in libvirt* at http://libvirt.org/firewall.html.

Network filtering with libvirt:

You can filter undesired network packets with the libvirt network filter driver.

Complete the following steps:

1. Define a network filter rule with the filtering characteristics you want in a file named `mysecfilter.xml`. For example:

```
<filter name='mysecfilter' chain='root'>
  <rule action='drop' direction='out' priority='500'>
    <tcp dstportstart='25' dstportend='25'/>
  </rule>
  <rule action='drop' direction='in' priority='500'>
    <tcp dstportstart='23' dstportend='23'/>
  </rule>
  <filterref filter='clean-traffic'/>
</filter>
```

2. Add the rule to the libvirt environment with the **virsh nwfilter-define** command.

```
# virsh nwfilter-define mysecfilter.xml
```

3. Associate the network filter named mysecfilter with a network interface by including a **filterref** element in the interface of a libvirt guest.

```
# virsh edit guest01

...

<interface type='network'>
    <mac address='54:52:00:5b:1a:cd'/>
    <source network='default'/>
    <target dev='vnet0'/>
    <model type='virtio'/>
    <filterref filter='mysecfilter'/>
</interface>
```

Securing storage devices

This topic provides information about the default location for virtual disk images for Red Hat Enterprise Linux 5.6 and 6.2, and how to change the location.

Default location for virtual disk images

This topic provides information about where libvirt stores virtual disk images (properly tagged with SELinux labels) in Red Hat Enterprise Linux 5.6 and 6.2, and about the default permissions of the images.

Note: Although AppArmor is the default LSM for SUSE Linux Enterprise Server 11 SP1, SELinux can also be used.

By default, libvirt stores images in the `/var/lib/libvirt/images` directory. The root user owns the `/var/lib/libvirt/images` directory. Therefore, only the root user can create files, delete files, and view a listing of the files in the `/var/lib/libvirt/images` directory.

The `/var/lib/libvirt/images` directory has the following permissions:

```
drwx--x--x. root root system_u:object_r:virt_image_t:s0 .
drwxr-xr-x. root root system_u:object_r:virt_var_lib_t:s0 ..
-rw-------. root root system_u:object_r:virt_image_t:s0 filename
```

where *filename* is the name of a virtual disk image. For example: `f14-live.iso`

The SELinux label of the `/var/lib/libvirt/images` directory has the `virt_image_t` type. Image files not currently in use also have the `virt_image_t` type.

When you start a guest operating system by using libvirt, sVirt relabels the image to reflect the process labels of the guest operating system. In addition, libvirt makes "qemu" the user and group owner of the image. For example:

```
-rw-------. qemu qemu system_u:object_r:svirt_image_t:s0:c408,c776 /var/lib/libvirt/images/f14-live.iso
```

This example shows that libvirt made "qemu" the user and group owner of the image. Also, sVirt relabeled the `f14-live.iso` image:

- With the `svirt_image_t` type, because processes of guest operating systems run in the svirt_t domain
- With the `c408,c776` categories to match the process labels of the guest operating system

Storing virtual disk images in a customized location

You can store virtual disk images in a location other than the `/var/lib/libvirt/images` directory.

The following information applies to KVM environments that are running Red Hat Enterprise Linux 5.6 or 6.

To store virtual disk images in a customized location, complete the following steps:

1. Create the directory in which you want to store the virtual disk images by running the **mkdir** command as follows:

   ```
   # mkdir directory
   ```

   where *directory* is the directory in which you want to store the virtual disk images. For example: `/mnt/raid/images`
2. Move the virtual disk images to the directory that you created in Step 1 by running the **mv** command as follows:

   ```
   # mv /var/lib/libvirt/images/image directory
   ```

   where:
   - *image* is a virtual image that you want to move from the `/var/lib/libvirt/images` directory to the directory that you created in Step 1. For example: `rh-5.5.qcow2`
   - *directory* is the directory that you created in Step 1. For example: `/mnt/raid/images`
3. Update the `domain.xml` file with the libvirt **virsh** utility:
   a. Edit the file by typing the following command:

      ```
      $ virsh edit domainID
      ```

      where *domainID* is the ID of the domain of the guest operating system whose virtual disk images you want to store.
   b. Change the value of `source file` to reflect the directory that you created in Step 1. For example,

      ```
      <source file='/mnt/raid/images/rh-5.5.qcow2'/>
      ```
4. Change the SELinux context for files by running the following command:

   ```
   # semanage fcontext -a -t virt_image_t "directory(/.*)?"
   ```

   where *directory* is the directory that you created in Step 1. For example: `/mnt/raid/images/`
5. Apply the context changes by running the following command:

```
# restorecon -R directory
```

where *directory* is the directory that you created in Step 1. For example: `/mnt/raid/images/`

6. Start the guest operating system.

Storing virtual disk images on an NFS mount

You can store virtual disk images on a Network File System (NFS) instead of the `/var/lib/libvirt/images` directory.

The following information applies to KVM environments with the SELinux and the sVirt services enabled.

Before you move the virtual disk images of a guest operating system to an NFS mount, be aware that NFS does not support file labels. This means that virtual disk images that are stored on an NFS mount cannot be labeled. Therefore, when using dynamic labeling, you must prevent the sVirt service from relabeling the virtual disk images. When using static labeling, you cannot manually label the virtual disk images.

To store virtual disk images on an NFS mount, complete the following steps:

1. Move the virtual disk images to an NFS mount.
2. Prevent the sVirt service from attempting to relabel the virtual disk images that are stored in NFS by running the following command:

```
# setsebool -P virt_use_nfs on
```

3. Start the guest operating system.

Related concepts:

"The sVirt service" on page 28
Learn about how the sVirt service isolates virtual machines, how to optionally create static sVirt labels, and how to verify dynamic and static sVirt labeling.

Remote management

Like many other open source software bundles, KVM-based virtualization favors modularity, flexibility, and open-standard interfaces where the functionality is separated into smaller, self-contained packages. Modularization offers benefits in terms of code reusability, maintainability, and flexibility. However, in terms of security, this design creates more entry-points of concern compared to the monolithic approach. This section contains information about how you can secure the KVM solution but still enjoy the benefits of having small, understandable, interconnected pieces.

By default, management of KVM guests cannot be performed remotely. This default condition exists because the libvirtd daemon does not create any listening sockets, and the qemu-kvm VNC console accepts connections only from the local host. You have, however, several choices for secure remote management configuration.

Overview of remote management

Packages providing virtualization support include the libvirtd daemon, and the management clients of the libvirtd daemon, including **virsh**, **virt-viewer**, and **virt-manager**.

Typical Red Hat Enterprise Linux 6.2 installation source has the following packages, which are part of the 'virtualization' package group:

qemu, qemu-img
: the user space process that performs machine virtualization, namely **qemu**, or **qemu-kvm**. The **qemu-img** utility is used to create virtual machine disk images.

libvirt, python-virtinst
the user space management layer, including the libvirtd daemon (a system service responsible for coordinating virtual machine management), **virsh** (a full-featured, command-line virtual machine management utility that connects to the libvirtd daemon), and **virt-install** (a command-line utility that facilitates virtual machine installations).

virt-viewer
a tool that displays the graphical (VNC) console of a virtual machine.

virt-manager
a desktop tool that manages virtual machines.

The libvirtd daemon is responsible for managing all things related to virtualization. It stores virtual machine guest configuration, creates the user space qemu-kvm processes that are responsible for machine emulation, and assigns resources such as virtual networks, disk images, or pass-through devices to the guests. Except for the VNC console (which is provided by the qemu-kvm process directly), all management of the guests is done through libvirtd daemon clients such as **virsh**, **virt-viewer**, and **virt-manager**. These clients can be initiated remotely through various methods including local UNIX domain sockets, TCP sockets, SSH tunnels, or secure TLS (Transport Layer Security) connections. The latter two are discussed. Authentication can also be handled using SASL, which is a library and a protocol designed to add pluggable authentication support for connection-based services.

Using the VNC console, management can be partially performed outside of the libvirtd daemon control because VNC clients connect directly to the qemu process.

The following table summarizes the management clients and the security methods they use.

Table 1. Management clients and security methods

	SSH + QEMU	SASL + QEMU	TLS + QEMU
virsh	X	X	X
virt-viewer/virt-manager	X		X
Other VNC viewer	X		

To allow incoming connections through the KVM host firewall, you are asked to open the following ports:

- 22 for SSH
- 16509 for SASL
- 16514 for TLS
- 5900 + VNC screen number for VNC connections

Figure 1 offers a simplified view of the architecture of a KVM virtualization host. It illustrates how each of the components described connect to each other and their external connection points.

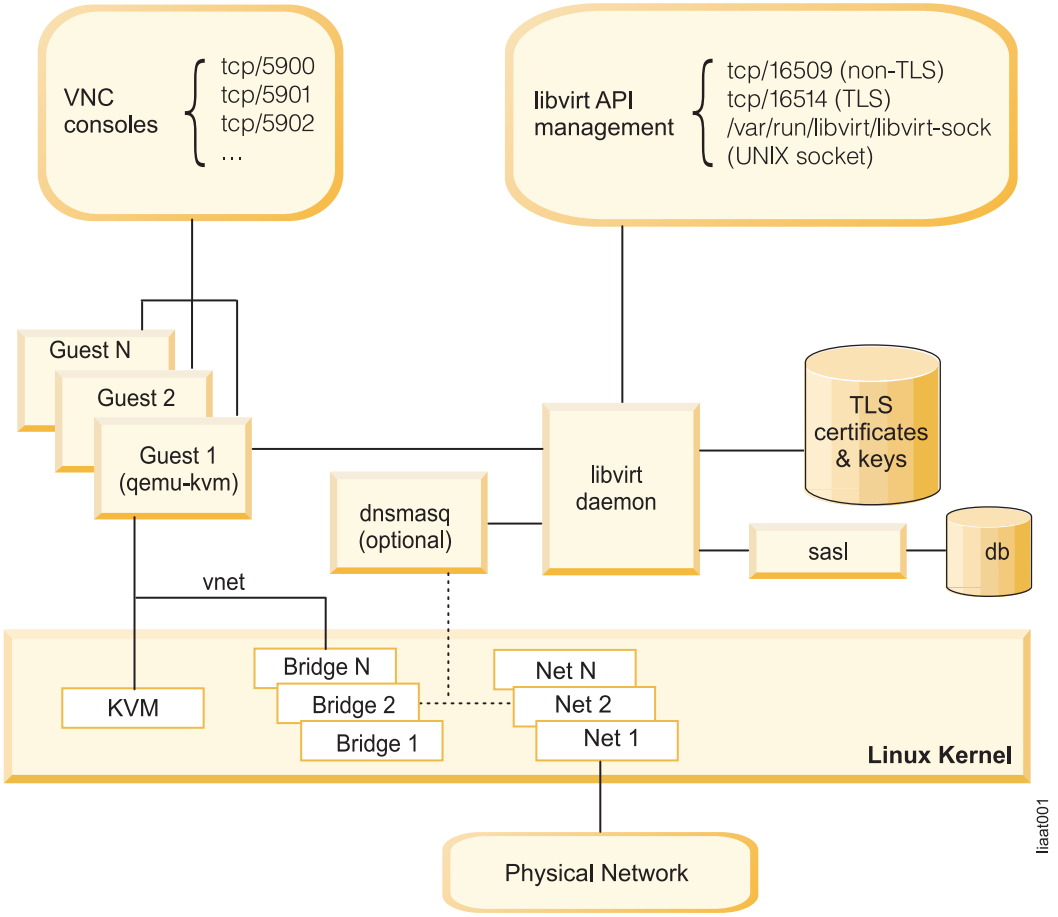


Figure 1. The architecture of a KVM virtualization host

Remote management using SSH tunnels

The most direct (while still secure) way of performing remote management of the libvirtd daemon plus qemu-kvm pair is by using standard SSH sessions as tunnels for communicating libvirt management data. The libvirt API can perform management through an SSH tunnel, so most management software built over libvirt is also capable of using this feature.

No special configuration or setup is needed if the root user, or any other user that is given permission to manage virtual machine guests, is allowed to log on the KVM host using a standard SSH session. Connections through an SSH tunnel are seen as local from the libvirtd daemon point of view. Through the SSH tunnel, you can remotely manage KVM guests using the **virsh** command, or access your KVM guest graphical consoles using the VNC viewer of your choice.

Managing KVM guests remotely with the virsh command

You can manage your remote KVM hosts by connecting to their local libvirtd daemon instance using the SSH method and running **virsh** commands. You must already be able to connect using SSH from your management station to the remote KVM host.

Install the libvirt package to make the **virsh** command available on your management station (client).

- Red Hat Enterprise Linux 6.2

  ```
  # yum install libvirt
  ```

- SUSE Linux Enterprise Server 11 SP1

```
# yast2 -i libvirt
```

To connect to a remote KVM host using an SSH tunnel, pass the **-c** flag followed by a valid URI to the **virsh** command. The following command connects to the libvirtd daemon running on *kvmhost.company.org* and lists the running qemu-kvm guests:

```
# virsh -c qemu+ssh://root@kvmhost.company.org/system list
root@kvmhost.company.org's password:
 Id Name                 State
----------------------------------
  1 guest01              running
```

The Uniform Resource Identifier (URI) is similar to a common HTTP Uniform Resource Locator, or URL. The communication protocol is specified as *qemu+ssh*, and the host to connect is *kvmhost.company.org*. For more information about libvirt URI format, see the libvirt online documentation at http://libvirt.org/uri.html.

Note: In these instructions, *kvmhost.company.org* is used as the domain name of the KVM host. Replace this domain name with the domain name of your KVM host when you are using these instructions in your environment. Make sure that the KVM host SSH daemon is accessible from your management station.

You can issue other **virsh** subcommands to manage your remote KVM guests by including the **-c** flag followed by a valid URI on any **virsh** command invocation. However, **-c** and its argument are an option on the **virsh** command, as opposed to **virsh** subcommands like *list*, *create*, or *reboot*. Thus, order is important: the **-c** flag must come before any **virsh** subcommand. For more information about how to use the **virsh** tool to manage your KVM guests, see *The developer's approach to installing and managing KVMs* at http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaai/kvmadv/kvmadvstart.htm.

Displaying the remote KVM VNC console using virt-viewer

The **virt-viewer** command can use SSH tunnels to display the VNC graphical console of your remote guests. However, you must be cautious when using this method due to security limitations.

The SSH tunneling method pipes arbitrary data through a secure path, and libvirt can automatically do the same with the VNC connection. However, the SSH tunneling method requires Qemu to accept insecure connections from the local host. This means that regular users in the KVM host system can attempt to connect to the VNC console locally. Even if access is secured by a VNC password, its limitation in size (eight characters) might be seen as a potential target for a dictionary attack.

To display the remote KVM VNC console, specify which KVM host to connect to through the **-c** flag followed by a valid URI. This example connects to *kvmhost.company.org* to display the guest01 VNC graphical console:

```
# virt-viewer -c qemu+ssh://root@kvmhost.company.org/system guest01
root@kvmhost.company.org's password:
```

If this configuration is successful, a virt-viewer-provided VNC session of your remote KVM guest graphical console opens.

Displaying the remote KVM VNC console using any VNC client

You can tunnel the VNC console connection manually using a standard SSH session if needed (for example, when the **virt-viewer** utility is unavailable on the management station).

To tunnel the VNC console connection manually, complete the following steps:

1. Use the **virsh** command to query the VNC graphical console screen number for the guest you would like to connect to:

   ```
   # virsh -c qemu+ssh://root@kvmhost.company.org/system vncdisplay guest01
   root@kvmhost.company.org's password:
   :1
   ```

2. Open an SSH session forwarding the remote VNC port to a local port. To find out which VNC port is being used by your remote KVM guest, add 5900 (the known base VNC port number) to the screen number from the previous step. In this example, the remote VNC port is 5901. The choice of local port is arbitrary. In this example, assume port 5910 (VNC Port for screen number 10) is free in the management station. The following command therefore forwards port 5901 of kvmhost.company.org to port 5910 of the host you are on (management station):

```
# ssh -L 5910:localhost:5901 root@kvmhost.company.org
 root@kvmhost.company.org's password:
 Last login: Tue Apr 6 15:28:14 2010 from otherclient.company.org
```

3. At another terminal, use your VNC client of choice to connect to the local port 5910, where the KVM host's 5901 port is forwarded to. SSH forwards the port and treats the remote port as if it were a local connection attempt:

```
# vncviewer localhost:10
```

If this configuration is successful, a VNC session of your remote KVM guest graphical console opens.

Remote management using SASL authentication and encryption

SASL provides secure authentication and data encryption while allowing integration with traditional or external authentication and authorization services.

In its simplest form, SASL can be used to define a database of credentials for authorization. In more complex scenarios, it can work with external authentication services such as Kerberos or LDAP to authenticate users. Either way, the libvirtd daemon guarantees confidentiality by requiring GSSAPI as the SASL method if remote management requests are not running on top of a secured TLS connection. The libvirtd daemon can use DIGEST-MD5 instead of GSSAPI as the SASL method. However, MD5 hashes are considered unsafe and should not be used. These variations of SASL enable encryption of the data being pushed through. For simplicity, the example below uses DIGEST-MD5 as the SASL method.

To configure remote management using SASL in the simplest scenario (no external authentication or TLS security), complete the following steps:

1. Log in to the KVM host.
2. Save a copy of the `/etc/libvirt/libvirtd.conf` file and the `/etc/sysconfig/libvirtd` file.
3. Edit the `/etc/libvirt/libvirtd.conf` file and make the following changes:
   a. Disable the `listen_tls` configuration directive (that is, set to `0`) because no TLS certificates are configured. Otherwise the libvirtd daemon fails to start.
   b. Ensure that the configuration directive `listen_tcp` is enabled (that is, set to `1`).
   c. Set the `auth_tcp` configuration directive to `sasl` to enable SASL authentication over TCP.

   The following example shows these parameters in contrast with the stock `libvirtd.conf` file, with changes highlighted for deletions (-) and additions (+):

```
--- libvirtd.conf.orig  2012-01-04 11:28:32.000000000 -0600
+++ libvirtd.conf       2012-01-04 11:34:02.000000000 -0600
@@ -19,7 +19,7 @@
 # using this capability.
 #
 # This is enabled by default, uncomment this to disable it
-#listen_tls = 0
+listen_tls = 0
 # Listen for unencrypted TCP connections on the public TCP/IP port.
 # NB, must pass the --listen flag to the libvirtd daemon process for this to
@@ -30,7 +30,7 @@
 # DIGEST_MD5 and GSSAPI (Kerberos5)
 #
 # This is disabled by default, uncomment this to enable it.
-#listen_tcp = 1
+listen_tcp = 1
```

```
@@ -143,7 +143,7 @@
 # Don't do this outside of a dev/test scenario. For real world
 # use, always enable SASL and use the GSSAPI or DIGEST-MD5
 # mechanism in /etc/sasl2/libvirt.conf
-#auth_tcp = "sasl"
+auth_tcp = "sasl"
 # Change the authentication scheme for TLS sockets.
 #
```

4. Edit the `/etc/sysconfig/libvirtd` file and enable the *--listen* parameter so that the libvirtd daemon listens to TCP/IP connections:

```
--- libvirtd.orig       2012-01-04 11:41:37.000000000 -0600
+++ libvirtd    2012-01-04 11:31:33.000000000 -0600
@@ -3,7 +3,7 @@

 # Listen for TCP/IP connections
 # NB. must setup TLS/SSL keys prior to using this
-#LIBVIRTD_ARGS="--listen"
+LIBVIRTD_ARGS="--listen"

 # Override Kerberos service keytab for SASL/GSSAPI
 #KRB5_KTNAME=/etc/libvirt/krb5.tab
```

5. Restart the libvirtd daemon for the changes to take effect:

```
# /etc/init.d/libvirtd restart
Stopping libvirtd daemon:                                  [  OK  ]
Starting libvirtd daemon:                                  [  OK  ]
```

6. Now that the libvirtd daemon is accepting TCP connections, add some users to the SASL database. The following example uses the **saslpasswd2** command to add the admin user to the libvirt credential database.

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

Note:

- The expected name of the SASL database for the libvirtd daemon authentication domain is `libvirt`. Do not use any other name for this database.
- Keep these credentials safe as every user in this database is authorized to log on and perform remote virtual machine administration.

7. If the KVM host is running a firewall, ensure that it allows incoming traffic through the libvirtd daemon TCP listening port. By default, the listening port is 16509.
   - Red Hat Enterprise Linux 6.2

     ```
     system-config-firewall-tui
     ```

   - SUSE Linux Enterprise Server 11 SP1

     ```
     yast2 firewall
     ```

8. When the security level configuration tool opens, choose **Other Ports** and add TCP port 16509 as an allowed network port.
9. Verify that the setup was successful by using SASL authentication to instruct libvirt enabled applications to connect using the TCP transport. The following example runs the **virsh** command from a remote management station and logs in as the newly created user admin to start the guest02 instance in the *kvmhost.company.org* system:

```
# virsh -c qemu+tcp://kvmhost.company.org/system start guest02
Please enter your authentication name:admin
Please enter your password:
Domain guest02 started
```

Remote management using TLS

TLS (Transport Layer Security) connections are made secure by digital signature verification when peers exchange certificates that were previously signed by a recognized certificate authority (CA).

In the most common scenario (for example when web browsers connect to web servers), the client application is configured to trust a certain list of CAs. Each server must prove its identity to the client by presenting a certificate signed by those trusted CAs with a matching Subject Name, usually the fully qualified domain name (FQDN) of the server.

In other less common scenarios that need improved security, the server also requires the client to present a digitally signed certificate to prove its identity.

The example given in this section demonstrates how to create a local CA, use the local CA to digitally sign server and client certificates, and distribute the certificates for use. The **openssl** command is used to create private keys and certificates that can be used directly by libvirt.

For more information about using the **openssl** command, run the **man openssl** command, or see http://www.openssl.org/docs/apps/openssl.html.

Step 1. Create a CA key and certificate in your KVM host

You can create a CA key and certificate in your KVM host. All certificates created are signed by a 2048-bit RSA key and a sha256 hash algorithm.

Complete the following steps:

1. Log in to your KVM host.
2. Create a temporary directory to keep the files, and change into it:

   ```
   # mkdir cert_files
   # cd cert_files
   ```

3. Using the **openssl** command, create a 2048-bit RSA key:

   ```
   openssl genrsa -out cakey.pem 2048
   ```

4. Use the key to create a self-signed certificate to your local CA:

   ```
   openssl req -new -x509 -days 1095 -key cakey.pem -out cacert.pem -sha256 \
           -subj "/C=US/L=Austin/O=IBM/CN=my CA"
   ```

5. Check your CA certificate:

   ```
   # openssl x509 -noout -text -in cacert.pem

   Certificate:
       Data:
           Version: 3 (0x2)
           Serial Number:
               b8:fd:55:0d:39:d8:48:fc
           Signature Algorithm: sha256WithRSAEncryption
           Issuer: C=US, L=Austin, O=IBM, CN=my CA
           Validity
               Not Before: Jan  9 02:11:58 2012 GMT
               Not After : Jan  8 02:11:58 2015 GMT
           Subject: C=US, L=Austin, O=IBM, CN=my CA
           Subject Public Key Info:
               Public Key Algorithm: rsaEncryption
                   Public-Key: (2048 bit)
                   Modulus:
                       00:9d:df:d0:3e:e0:ec:55:8e:e0:e5:b2:44:da:94:
                       a5:d7:28:4d:18:2f:ea:eb:f1:a2:e3:fa:ee:a5:7c:
                       b0:cb:db:6f:9c:4f:8a:14:ff:19:51:c3:8b:f3:ac:
                       ce:8a:d2:c4:1d:16:03:b5:6d:94:32:8b:25:b9:75:
                       6a:d5:eb:f9:9f:72:68:af:02:03:47:1d:e8:07:87:
                       fa:1a:64:d8:84:ee:62:fb:41:bb:5d:25:c7:67:8c:
                       ad:89:89:bf:3e:bf:b3:4c:42:27:4e:44:68:cf:48:
   ```

```
                23:6e:f3:8d:3b:62:a1:a6:e5:d2:a0:db:8b:4d:0e:
                0b:5f:02:aa:52:06:49:ec:7f:ea:cd:00:6d:d2:eb:
                0e:71:b3:70:98:e1:c1:81:7e:d5:1d:e4:7d:d7:e2:
                22:79:24:3f:d1:0b:46:56:29:ce:ee:49:82:84:74:
                a8:b6:da:e0:11:35:57:46:f2:cb:11:0a:4c:c7:51:
                11:be:c0:d4:73:99:fe:d2:22:e1:f4:e2:53:4a:c4:
                da:b0:62:b4:3d:ad:a7:63:7f:72:f2:f6:6f:b8:cc:
                fc:59:c5:95:50:4e:58:01:42:6f:0b:a8:ad:09:74:
                b6:75:e9:d1:bc:3d:76:2c:8f:7f:a9:a8:3b:b7:38:
                39:6b:b1:67:ca:b8:fc:76:c1:d2:c9:2e:be:70:8d:
                d3:71
            Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                0E:B0:0D:46:DF:6D:1E:05:5A:27:AF:58:AA:BC:57:57:C3:9C:83:00
            X509v3 Authority Key Identifier:
                keyid:0E:B0:0D:46:DF:6D:1E:05:5A:27:AF:58:AA:BC:57:57:C3:9C:83:00

            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: sha256WithRSAEncryption
        5a:9f:28:9f:72:b1:46:f4:c7:e8:cd:c6:d4:55:80:b0:55:2b:
        04:dc:b3:2b:7b:46:da:0b:55:88:34:c3:3a:90:1f:e7:b9:0b:
        c3:f2:82:48:eb:4e:69:da:c3:ac:df:36:18:67:4d:88:85:1c:
        c0:d1:bd:60:d5:8a:a6:66:d3:c7:5e:d8:ba:e5:9b:cd:37:2a:
        7b:e5:a0:ed:77:72:f8:4f:15:0b:3f:47:5e:cc:e0:a2:5d:71:
        bd:f0:f6:e0:7b:1b:19:93:cb:84:6b:6f:75:f0:ef:3b:c7:c8:
        ac:d4:e2:e8:9d:a5:21:1c:c3:89:22:00:db:94:fb:0e:00:4a:
        87:18:c0:dd:11:1b:a6:91:b5:90:24:24:6e:5e:a1:b6:94:31:
        3e:40:b7:de:34:62:e0:a1:a3:e2:1c:c0:3c:2d:a6:3a:3f:60:
        75:15:0c:51:cc:19:3e:64:37:62:cc:1f:5f:08:8b:89:ec:f0:
        a5:a7:7c:1d:4b:da:88:8b:f8:c0:a7:a5:9b:c7:7f:d2:a4:58:
        b9:63:92:c4:14:22:3f:6b:8d:d4:28:88:0a:b3:e6:60:c6:ca:
        8b:eb:3a:af:a2:0d:fe:7a:7b:2e:95:dc:ef:1c:f0:9d:61:55:
        4d:a3:65:5a:aa:40:b0:06:b9:c5:2e:95:84:cc:ef:52:d3:0d:
        81:6b:78:ec
```

Step 2. Create the client and server keys and certificates in your KVM host

You can create the client and server keys and certificates in your KVM host.

Note: In order to create both client and server certificates, you must create a certificate signing request.

A certificate signing request (csr) is a message used to apply for a certificate to a CA. It usually contains identification information (the certificate subject) for the certificate owner (for example, country, organization, country). It is signed by the applicant, using its private key.

The format used by the certificate signing request generated by **openssl** is described by the PKCS#10 standard.

Complete the following steps:

1. Create the keys:

   ```
   # openssl genrsa -out serverkey.pem 2048
   # openssl genrsa -out clientkey.pem 2048
   ```

2. Create a certificate signing request for the server. Remember to change the kvmhost.company.org address (used in the server certificate request) to the fully qualified domain name of your KVM host:

   ```
   # openssl req -new -key serverkey.pem -out serverkey.csr \
             -subj "/C=US/O=IBM/CN=kvmhost.company.org"
   ```

3. Create a certificate signing request for the client:

   ```
   # openssl req -new -key clientkey.pem -out clientkey.csr \
             -subj "/C=US/O=IBM/OU=virtualization/CN=root"
   ```

4. Create client and server certificates:

```
# openssl x509 -req -days 365 -in clientkey.csr -CA cacert.pem -CAkey cakey.pem \
          -set_serial 1 -out clientcert.pem
# openssl x509 -req -days 365 -in serverkey.csr -CA cacert.pem -CAkey cakey.pem \
          -set_serial 94345 -out servercert.pem
```

5. Check the keys:

```
# openssl rsa -noout -text -in clientkey.pem
# openssl rsa -noout -text -in serverkey.pem
```

6. Check the certificates:

```
# openssl x509 -noout -text -in clientcert.pem

Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 9434242 (0x8ff482)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, L=Austin, O=IBM, CN=my CA
        Validity
            Not Before: Jan 10 19:44:06 2012 GMT
            Not After : Jan  9 19:44:06 2015 GMT
        Subject: C=US, O=IBM, OU=virtualization, CN=root
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c1:ef:30:8e:b3:73:3a:d6:72:a3:c5:44:1f:a2:
                    63:23:20:2b:b9:34:04:2a:1c:12:18:8e:e5:87:ec:
                    ff:28:ec:b1:62:e6:5e:ec:bb:67:cd:e9:18:68:c5:
                    51:f6:f6:fa:83:d0:0c:74:bd:72:f2:ac:a5:35:ce:
                    8c:84:1e:dc:a2:3d:bb:90:32:a8:14:48:2b:57:ae:
                    d5:91:14:5e:92:ad:85:78:92:35:81:02:d0:73:9f:
                    4e:68:52:d3:a9:24:d5:c0:0d:1f:2f:0d:c3:57:67:
                    42:a3:50:b7:9b:1e:c3:25:9e:f0:35:13:f8:9c:d5:
                    76:5e:c4:eb:a0:d2:42:01:0c:17:f1:59:78:0d:1c:
                    0a:b1:3d:61:3d:89:85:7c:cd:9a:a3:07:bc:79:e3:
                    05:5d:97:65:51:e7:9e:26:09:d8:6d:a9:86:03:13:
                    bd:36:af:66:fc:a7:7b:12:9a:cc:38:0d:d1:b4:a1:
                    9a:e7:13:50:9e:c2:b5:8e:df:b4:7c:74:6e:bb:07:
                    75:ef:07:8f:04:d3:2a:ee:e1:4b:ce:51:65:59:02:
                    3c:15:d9:d2:30:0a:0e:44:10:30:97:13:df:57:cf:
                    1e:df:5f:34:02:bf:8d:b8:ef:ba:25:3b:86:db:ec:
                    6b:d6:01:0c:09:e7:da:07:5f:47:af:27:fd:e1:a3:
                    58:2d
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha1WithRSAEncryption
        94:9c:05:53:39:7f:ae:3c:e9:14:b0:31:98:3f:df:af:05:dc:
        67:73:10:bc:e5:7d:bd:20:38:af:1f:56:86:8f:e1:64:fb:ca:
        df:94:80:7d:78:ec:f8:bb:4e:09:10:7e:d1:2d:50:04:dc:ea:
        6d:db:e0:fb:02:da:07:67:e2:06:28:fe:10:ac:9b:37:a6:8d:
        f3:45:07:61:18:d5:84:75:66:60:d8:fc:8d:8c:38:ce:c3:59:
        d0:11:d7:9e:d0:a6:eb:1c:e2:ff:5d:6b:61:bd:30:fe:6f:61:
        ff:2a:25:be:32:b0:31:91:be:3d:92:60:59:57:ec:9e:fd:20:
        98:38:4f:6d:53:da:ce:2c:22:cd:61:de:6d:51:5b:b4:f0:91:
        05:c6:e3:fe:e9:aa:43:45:a0:a8:ec:ed:4b:db:c1:fb:d0:13:
        47:42:cb:38:6a:b0:10:60:ce:a7:80:ef:4b:ab:e8:0a:7e:d8:
        40:7e:b4:3f:74:b3:de:d0:9c:97:31:dd:11:47:df:35:63:9f:
        17:2c:e0:d7:f2:17:e1:44:50:e1:80:41:f3:54:00:3f:fe:fe:
        7e:cf:c4:25:26:8a:ae:34:99:75:d6:90:52:4d:ac:ef:ea:74:
        e9:f6:f0:42:35:b0:eb:1f:34:6d:a3:a7:f2:bc:5c:02:10:f0:
        b8:e0:6a:3a

# openssl x509 -noout -text -in servercert.pem

Certificate:
    Data:
        Version: 1 (0x0)
```

```
        Serial Number: 94345 (0x17089)
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C=US, L=Austin, O=IBM, CN=my CA
        Validity
            Not Before: Jan  9 03:27:30 2012 GMT
            Not After : Jan  8 03:27:30 2015 GMT
        Subject: C=US, O=IBM, CN=kartoffel.stglabs.ibm.com
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:c4:fb:0d:92:48:ae:d1:fb:e1:50:c3:32:8f:4d:
                    fd:de:83:07:a7:cf:02:ef:10:be:3c:ad:44:cd:df:
                    b7:52:97:fd:c2:ce:47:39:cc:e5:7d:50:4e:16:06:
                    48:c0:7f:12:35:b0:da:80:a9:67:7f:72:b2:c8:27:
                    65:f6:36:54:e1:3c:9c:2d:ac:6d:a1:a3:c1:ae:7f:
                    96:e1:9d:aa:56:05:85:ff:07:f5:09:29:27:d4:34:
                    99:3a:0b:f2:35:3a:36:dd:b0:f2:78:ca:cf:4c:21:
                    cb:79:bd:8b:23:d6:f6:62:4f:d4:44:67:62:e5:60:
                    47:da:05:ae:00:02:03:84:5e:ad:e6:12:ed:ef:27:
                    99:72:59:46:38:f1:b9:65:fa:47:7a:29:90:1d:14:
                    47:06:52:da:bd:5b:91:be:42:b3:36:79:de:b2:e6:
                    6a:4d:01:89:51:d1:a9:3c:7e:c4:7c:37:64:2f:76:
                    5b:7b:26:08:d8:cc:77:07:20:02:43:53:10:c4:02:
                    58:f8:53:7e:51:93:66:17:68:b7:35:85:fd:58:34:
                    5c:3e:1d:0e:74:cf:9c:4e:28:86:1e:b0:b7:16:98:
                    5c:8b:a8:4e:56:e1:46:f6:dc:66:b9:76:5f:33:dd:
                    0a:4e:ef:f2:d7:e6:c8:a9:3e:76:50:37:03:95:c3:
                    4b:c3
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha256WithRSAEncryption
        1a:a0:91:19:56:10:da:7c:9c:13:2a:2a:da:ae:12:15:60:71:
        33:3a:2b:e0:84:f0:48:d8:d2:f7:f6:ba:08:f3:f9:9d:d8:50:
        fd:54:c0:ee:60:99:8d:0b:7b:21:6a:d1:9a:aa:71:df:f8:69:
        dd:44:96:74:2c:85:e8:b0:54:b2:7b:25:c6:06:1f:67:86:45:
        0e:c6:6f:80:55:a7:43:d1:51:97:ab:80:17:16:a4:2b:ee:a1:
        2b:ba:5c:7b:05:54:83:78:10:dd:42:30:68:40:7b:1c:7d:df:
        60:9d:85:6e:16:ea:dc:74:3e:c6:c6:2b:17:30:0f:9c:37:bb:
        c2:3c:f8:ed:ea:ca:1b:b4:a4:66:30:ad:a7:85:7a:f9:94:28:
        b6:a5:f0:d8:af:80:5d:3a:3d:00:ee:32:6e:88:15:97:fa:ce:
        ba:75:70:38:d9:30:91:a3:6e:c0:52:20:a3:4e:38:bf:5a:97:
        60:f6:22:4d:46:a3:a0:f1:2b:99:40:ab:c0:b3:67:6e:47:5f:
        0b:40:c7:85:b5:6f:a7:76:1c:0d:d3:dd:7e:02:b4:c4:cb:e6:
        8a:35:f9:c2:10:6e:13:a7:c3:c3:ec:87:b2:cd:c5:a1:d9:8e:
        b5:53:5c:d1:bd:d6:6d:19:44:f1:01:c3:7c:0d:a3:14:24:7e:
        3e:b9:d3:f5
```

Step 3. Distribute keys and certificates to the server (KVM host)

When certificates and keys for both the server and the client are in a format readable by libvirt, distribute and configure them to be used by TLS.

To distribute keys and certificates to the server (KVM host), complete the following steps:

1. Copy the CA certificate `cacert.pem` file to `/etc/pki/CA/cacert.pem`.

   ```
   # cp cacert.pem /etc/pki/CA/cacert.pem
   ```

2. Create the `/etc/pki/libvirt` directory. Copy the `servercert.pem` server certificate file to `/etc/pki/libvirt/servercert.pem`. Create the `/etc/pki/libvirt/private` directory. Copy the `serverkey.pem` server key file to `/etc/pki/libvirt/private/serverkey.pem` directory. Make sure that only the root user is able to access the private key.

   Note: If the keys or certificates are named incorrectly or copied to the wrong directories, the authorization fails.

```
# mkdir /etc/pki/libvirt
# cp servercert.pem /etc/pki/libvirt/.
# mkdir /etc/pki/libvirt/private
# cp serverkey.pem /etc/pki/libvirt/private/.
# chmod -R o-rwx /etc/pki/libvirt/private
```

3. Verify that the files are placed correctly:

```
# find /etc/pki/CA/*|xargs ls -l
-rw-r--r-- 1 root root  821 Apr  9 15:10 /etc/pki/CA/cacert.pem
# ls -lR /etc/pki/libvirt
/etc/pki/libvirt:
total 16
drwxr-x--- 2 root root 4096 Apr  9 16:35 private
-rw-r--r-- 1 root root  751 Apr  9 15:11 servercert.pem

/etc/pki/libvirt/private:
total 8
-rw-r----- 1 root root 1040 Apr  9 15:11 serverkey.pem
```

Step 4. Distribute keys and certificates to clients (management stations)

For every configured management station, repeat the following steps. Place a copy of the client certificate (`clientcert.pem`) in the `/etc/pki/libvirt/` directory and the key (`clientkey.pem`) in the `/etc/pki/libvirt/private/` directory. As usual, restrict access to the client key to the root user.

Restricting the client key to root access results in root-only access to the server using this certificate/key pair. This situation is acceptable if the management console requires root access to manage remote environments. Otherwise you can disable client certificate verification in the server configuration and stack another layer of authentication (using SASL) on top of TLS.

1. Log in to the management station.
2. Copy the CA certificate (`cacert.pem`) from the KVM host to the management station `/etc/pki/CA/` directory. Do not change the file name.

   ```
   # scp kvmhost.company.org:/tmp/cacert.pem /etc/pki/CA/
   ```

3. Copy the client certificate (`clientcert.pem`) to the `/etc/pki/libvirt/` directory, and the client key (`clientkey.pem`) to the `/etc/pki/libvirt/private/` directory. Use the default file names and make sure that only the root user is able to access the private key.

   Note: If the keys or certificates are named incorrectly or copied to the wrong directories, authorization fails.

   ```
   # cp clientcert.pem /etc/pki/libvirt/.
   # mkdir /etc/pki/libvirt/private
   # cp clientkey.pem /etc/pki/libvirt/private/.
   # chmod -R o-rwx /etc/pki/libvirt/private
   ```

4. Verify that the files are placed correctly:

   ```
   # ls -lR /etc/pki/libvirt/
   /etc/pki/libvirt/:
   total 8
   -rw-r--r-- 1 root root  767 2010-04-09 13:54 clientcert.pem
   drwxr-xr-- 2 root root 4096 2010-04-09 14:00 private

   /etc/pki/libvirt/private:
   total 4
   -rw-r--r-- 1 root root 1044 2010-04-09 13:55 clientkey.pem
   ```

Step 5. Edit the libvirtd daemon configuration

Make sure that the libvirtd daemon is listening to network connections and that the `libvirtd.conf` file specifies allowed subjects and client certificates.

To edit the libvirtd daemon configuration, complete the following steps:

1. Log in to the KVM host.

2. Make a copy of the /etc/sysconfig/libvirtd file and the /etc/libvirt/libvirtd.conf file.
3. Edit the /etc/sysconfig/libvirtd file and ensure that the **--listen** argument is passed to the libvirtd daemon. This step ensures that the libvirtd daemon is listening to network connections. The following example shows the changes from the original file:

```
--- libvirtd.orig       2012-01-04 11:41:37.000000000 -0600
+++ libvirtd    2012-01-04 11:31:33.000000000 -0600
@@ -3,7 +3,7 @@

 # Listen for TCP/IP connections
 # NB. must set up TLS/SSL keys prior to using this
-#LIBVIRTD_ARGS="--listen"
+LIBVIRTD_ARGS="--listen"

...
```

4. Edit the /etc/libvirt/libvirtd.conf file and configure a set of allowed subjects using the *tls_allowed_dn_list* directive in the libvirtd.conf file. The following example shows the changes from the original file. It restricts acceptable client certificates to certificates with the "O=IBM,OU=virtualization" values, while the country (C) and common name (CN) might be assigned any value. The fields in the subject must be in the same order as was used when creating the certificate.

```
--- libvirtd.conf.orig  2012-01-04 11:28:32.000000000 -0600
+++ libvirtd.conf       2012-01-10 11:33:02.000000000 -0600
@@ -210,7 +210,7 @@
 #
 # By default, no DN's are checked
 #tls_allowed_dn_list = ["DN1", "DN2"]
-
+tls_allowed_dn_list = ["C=*,O=IBM,OU=virtualization,CN=*"]

...
```

5. Restart the libvirtd daemon service for changes to take effect:

```
# /etc/init.d/libvirtd restart
Stopping libvirtd daemon:                                  [  OK  ]
Starting libvirtd daemon:                                  [  OK  ]
```

Step 6. Change the firewall configuration

Allow TLS-authenticated connections through the firewall of the KVM host by opening its TCP port 16514.

To allow TLS-authenticated connections through the firewall of the KVM host, complete the following steps:

1. Access the security level configuration:
   - Red Hat Enterprise Linux 6.2

     ```
     # system-config-firewall-tui
     ```

   - SUSE Linux Enterprise Server 11 SP1

     ```
     # yast2 firewall
     ```

2. Click **Other Ports** and add TCP port 16514 as a trusted port.

Step 7. Verify that remote management is working

Clients with a valid certificate and subject can connect to the KVM host libvirtd daemon to perform virtual machine guest administration.

To verify that clients are able to connect, run the following commands in your management station (client):

```
# virsh -c qemu+tls://kvmhost.company.org/system list --all
 Id Name                 State
----------------------------------
```

```
  - guest01              shut off
  - guest02              running

# virsh -c qemu+tls://kvmhost.company.org/system start guest01
Domain guest01 started
```

Securing remote VNC console with TLS

The VNC console that produces the graphical output for the virtual machine is not part of the libvirt API. It requires a different method than securing libvirt communication for virtual machine guest management demonstrated earlier. An exception is when using VNC through SSH tunnels.

For more information about using VNC through SSH tunnels, see "Remote management using SSH tunnels" on page 13.

Although not part of the libvirt API, the libvirtd daemon can be configured to instruct Qemu to accept only VNC-over-TLS connections while enforcing client certificate checks.

TLS security for the VNC console requires private keys and certificates for both the client (any TLS-enabled VNC client) and for the server (the Qemu process). The keys and certificates from "Remote management using TLS" on page 17 can be reused.

Step 1. Server (KVM host) configuration:

You can configure the server (KVM host) by copying certificate and key files, or by using symbolic links to reuse them.

Complete the following steps:

1. Copy the CA certificate (`cacert.pem`) to the `/etc/pki/libvirt-vnc/` directory and rename it to `ca-cert.pem`. If you are reusing the certificate from the libvirt TLS configuration, you can use a hard link instead:

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

2. Change the certificate file ownership to the "qemu" user:

   ```
   # chown qemu ca-cert.pem
   ```

3. Restrict access to the certificate file:

   ```
   # chmod 440 ca-cert.pem
   ```

4. Copy the `servercert.pem` client certificate file to `/etc/pki/libvirt-vnc/server-cert.pem`. Copy the `serverkey.pem` key file to `/etc/pki/libvirt-vnc/server-key.pem`. Hard links can also be used in case of reusing the certificates and keys from the libvirt TLS configuration:

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

   Note: The expected server and client key and certificate names in the VNC environment are different from the simple TLS environment. Be sure that they are named correctly or the authorization fails.

5. Change the file ownership to the "qemu" user:

   ```
   # chown qemu server-cert.pem
   # chown qemu server-key.pem
   ```

6. Restrict access to these files:

   ```
   # chmod 440 server-cert.pem
   # chmod 440 server-key.pem
   ```

7. Verify that the files are where they belong:

```
# ls -lR /etc/pki/libvirt-vnc/
/etc/pki/libvirt-vnc/:
total 12
-r--r----- 1 qemu root 16 Apr 12 14:00 ca-cert.pem
-r--r----- 1 qemu root 25 Apr 13 00:01 server-cert.pem
-r--r----- 1 qemu root 32 Apr 13 00:02 server-key.pem
```

8. By default, the libvirtd daemon instructs Qemu to accept only local connections. Enable TLS client certificate checking by editing the `/etc/libvirt/qemu.conf` file to set the *vnc_listen* value to 0.0.0.0 to accept connections from any interface, and the *vnc_tls* and *vnc_tls_x509_verify* values to 1. The following example shows the updated configuration in contrast to the original `qemu.conf` file:

```
--- qemu.conf.orig      2010-02-11 12:10:05.000000000 -0600
+++ qemu.conf   2010-02-11 12:26:20.000000000 -0600
@@ -9,7 +9,7 @@
 # NB, strong recommendation to enable TLS + x509 certificate
 # verification when allowing public access
 #
-# vnc_listen = "0.0.0.0"
+vnc_listen = "0.0.0.0"


 # Enable use of TLS encryption on the VNC server. This requires
@@ -20,7 +20,7 @@
 # It is necessary to setup CA and issue a server certificate
 # before enabling this.
 #
-# vnc_tls = 1
+vnc_tls = 1


 # Use of TLS requires that x509 certificates be issued. The
@@ -46,7 +46,7 @@
 # Enabling this option will reject any client who does not have a
 # certificate signed by the CA in /etc/pki/libvirt-vnc/ca-cert.pem
 #
-# vnc_tls_x509_verify = 1
+vnc_tls_x509_verify = 1
```

9. For each existing and new virtual machine guest, make sure that the domain configuration accepts external connections. Use the **virsh edit** command to start changing a guest domain definition:

```
# virsh edit rhel62
```

The following example shows the differences in the before and after versions of the domain XML configuration file. In the new version, external VNC connections are allowed by listening to all hosts (0.0.0.0) instead of the local host (127.0.0.1):

```
--- rhel62.xml.orig     2012-01-11 13:02:22.000000000 -0600
+++ rhel62.xml  2012-01-11 13:02:59.000000000 -0600
@@ -46,7 +46,7 @@
     <target port='0'/>
    </console>
    <input type='mouse' bus='ps2'/>
-   <graphics type='vnc' port='5900' autoport='yes' listen='127.0.0.1' keymap='en-us'/>
+  <graphics type='vnc' port='5900' autoport='yes' listen='0.0.0.0' keymap='en-us'/>
   </devices>
  </domain>
```

10. Restart the libvirtd daemon and any running guests for changes to take effect:

```
# virsh list
 Id Name                 State
----------------------------------
  1 rhel62               running

# virsh shutdown rhel62
Domain rhel62 is being shutdown
```

```
# /etc/init.d/libvirtd restart
Stopping libvirtd daemon:                                  [  OK  ]
Starting libvirtd daemon:                                  [  OK  ]
```

11. Allow VNC connections through your KVM host firewall by opening the corresponding port:
    a. Find the VNC port number of your KVM guest:

       ```
       # virsh vncdisplay rhel62
       :0
       ```

       The result (0) is the VNC console screen number for that particular guest (rhel62 in our example), if enabled.
    b. Add 5900 to this screen number (0 in this case) and you have the corresponding TCP port for your VNC connection. In this example, the equation is 5900 + 0 = 5900. This information is also available in the XML configuration file for the domain unless it is specified as *port='-1'*, which means a port is dynamically assigned by the libvirtd daemon.
    c. Open the TCP port for the VNC console by using the security level configuration tool:
       - Red Hat Enterprise Linux 6.2

         ```
         system-config-firewall-tui
         ```

       - SUSE Linux Enterprise Server 11 SP1

         ```
         yast2 firewall
         ```

    d. Click **Other Ports** and add the port number you obtained in Step b. In this example, that port is 5900. Make sure that TCP port 16514 is also opened for TLS connection.

       For more information about opening TCP port 16514 for the TLS connection, see "Step 6. Change the firewall configuration" on page 22.

Step 2. Client configuration:

On the client side, certificate and key configuration is largely dependent on the VNC client.

The following example uses the **virt-viewer** VNC client, which is built on top of libvirt. It automatically associate a virtual-machine guest with its corresponding VNC console screen number. The **virt-viewer** tool looks for the client certificates and keys in the **$HOME/.pki/libvirt-vnc/** directory, with the names of **clientcert.pem** for the certificate and **clientkey.pem** for the key. The CA certificate is expected to be found at **$HOME/.pki/CA/ca-cert.pem**. All of these certificates and keys can be reused from the libvirt client TLS configuration.

1. Create the **$HOME/.pki/libvirt-vnc/** directory, the **$HOME/.pki/CA/** directory, and link the certificate and key files to these directories.

   Note: The expected name of the CA certificate in the VNC environment (**ca-cert.pem**) is different from when it was created in the libvirt TLS configuration section (**cacert.pem**). Any discrepancy in file names and their locations to what are expected will cause authentication to fail.

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

2. Verify that the files are placed correctly:

   ```
   # ls -lR ~/.pki/*
   /home/kvmmgr/.pki/CA:
   total 0
   lrwxrwxrwx 1 kvmmgr kvmmgr 22 2010-04-12 22:14 ca-cert.pem -> /etc/pki/CA/cacert.pem

   /home/kvmmgr/.pki/libvirt-vnc:
   ```

```
total 0
lrwxrwxrwx 1 kvmmgr kvmmgr 31 2010-04-12 22:16 clientcert.pem -> /etc/pki/libvirt/clientcert.pem
lrwxrwxrwx 1 kvmmgr kvmmgr 38 2010-04-12 22:16 clientkey.pem -> /etc/pki/libvirt/private/clientkey.pem
```

3. With the information gathered in Step 2, use the **virt-viewer** tool to connect to the VNC console for the guest of choice:

   ```
   # virt-viewer -c qemu+tls://kvmhost.company.org/system guest01
   ```

   You should see a VNC session of your KVM guest open.

   The following table lists the expected locations of keys and certificates for the **virsh** command (TLS) and the **virt-viewer** command (VNC over TLS) on the client side and the server side.

Table 2. Client-side and host-side programs, certificates, and keys

Client-side program	Client-side certificates and keys	Host-side program	Host-side certificates and keys
virsh (TLS)	/etc/pki/CA/cacert.pem /etc/pki/libvirt/clientcert.pem /etc/pki/libvirt/private/clientkey.pem	libvirtd (TLS)	/etc/pki/CA/cacert.pem /etc/pki/libvirt/servercert.pem /etc/pki/libvirt/private/serverkey.pem
virt-viewer (VNC-over-TLS)	~/.pki/CA/ca-cert.pem ~/.pki/libvirt-vnc/clientcert.pem ~/.pki/libvirt-vnc/clientkey.pem	qemu (VNC-over-TLS)	/etc/pki/libvirt-vnc/ca-cert.pem /etc/pki/libvirt-vnc/server-cert.pem /etc/pki/libvirt-vnc/server-key.pem

Securing SPICE console with TLS

The Simple Protocol for Independent Computing Environments (SPICE) is an open source, adaptive, remote rendering protocol used by Red Hat Enterprise Linux to connect users to their virtual desktops.

Unlike other remote rendering protocols like RDP, the SPICE architecture allows it to choose the most efficient place to process graphics, providing an optimum user experience while reducing system load. For instance, if the client is powerful enough to process the graphics, SPICE sends it just the graphics commands, preserving server resources.

Step 1. Server (KVM host) configuration:

Configure the server to secure SPICE console with TLS.

Complete the following steps:

1. Copy the `cacert.pem` CA certificate file to `/etc/pki/libvirt-spice/ca-cert.pem`. If you are reusing the certificate from the libvirt TLS configuration, you can use a hard link instead:

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

2. Change the certificate file ownership to the “qemu” user:

   ```
   # chown qemu ca-cert.pem
   ```

3. Restrict access to the certificate file:

   ```
   # chmod 440 ca-cert.pem
   ```

4. Copy the `servercert.pem` client certificate file to `/etc/pki/libvirt-spice/server-cert.pem`. Copy the `serverkey.pem` key file to `/etc/pki/libvirt-spice/server-key.pem`. A hard link can also be used in case of reusing the certificates and keys from the libvirt TLS configuration:

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

   Note: The expected server and client key and certificate names in the SPICE environment are different from the simple TLS environment. Be sure that they are named correctly or the authorization fails.

5. Change the file ownership to the “qemu” user:

   ```
   # chown qemu server-cert.pem
   # chown qemu server-key.pem
   ```

6. Restrict access to these files:

```
# chmod 440 server-cert.pem
# chmod 440 server-key.pem
```

7. Verify that the files are where they belong:

```
# ls -lR /etc/pki/libvirt-spice/
/etc/pki/libvirt-spice/:
total 12
-r--r----- 1 qemu root 16 Apr 12 14:00 ca-cert.pem
-r--r----- 1 qemu root 25 Apr 13 00:01 server-cert.pem
-r--r----- 1 qemu root 32 Apr 13 00:02 server-key.pem
```

8. Uncomment (or add, if they do not exist) the following in the `/etc/libvirt/qemu.conf` file:

```
spice_tls = 1
spice_tls_x509_cert_dir = "/etc/pki/libvirt-spice"
```

9. Restart the libvirtd daemon and any running guests for changes to take effect:

```
# virsh list
 Id Name                 State
----------------------------------
  1 rhel62               running

# virsh shutdown rhel62
Domain rhel62 is being shutdown

# /etc/init.d/libvirtd restart
Stopping libvirtd daemon:                                  [  OK  ]
Starting libvirtd daemon:                                  [  OK  ]
```

10. Allow SPICE console connections through your KVM host firewall by opening the two ports used by SPICE. Port 5008 is the normal SPICE port. Port 5909 is the TLS SPICE port.
    a. Open the security level configuration tool:
       - Red Hat Enterprise Linux 6.2

         ```
         system-config-firewall-tui
         ```

       - SUSE Linux Enterprise Server 11 SP1

         ```
         yast2 firewall
         ```

    b. Click **Other Ports** and add ports 5008 and 5909.

Step 2. Guest domain configuration:

Edit the guest domain configuration to secure SPICE console with TLS.

Complete the following steps:

1. Shut down the guest domain if it is running:

```
# virsh list
 Id Name                 State
----------------------------------
  1 rhel62               running

# virsh shutdown rhel62
Domain rhel62 is being shutdown
```

2. Add the SPICE graphics device to the guest domain.

```
# virsh edit rhel62
```

   The following example shows the differences in the before and after versions of the guest domain XML configuration file. The VNC device was removed in the new version because a guest can have only one graphics device.

```
--- rhel62.xml.orig 2011-12-23 13:02:22.000000000 -0600
+++ rhel62.xml 2010-12-23 13:02:59.000000000 -0600
@@ -46,7 +46,7 @@
<target port='0'/>
```

```
</console>
<input type='mouse' bus='ps2'/>
- <graphics type='vnc' port='5900' autoport='yes' listen='127.0.0.1' keymap='en-us'/>
+ <graphics type='spice' port='5908' tlsPort='5909' autoport='no' listen='0.0.0.0'/>
</devices>
</domain>
```

3. Start the guest domain with the new configuration:

```
# virsh start rhel62
```

Step 3. Client configuration:

Configure the client to secure SPICE console with TLS.

Complete the following steps:

1. The **spicec** command needs the `$HOME/.spicec/spice_truststore.pem` file to recognize the host certificate. Replace (or create new if it does not exist) the `$HOME/.spicec/spice_truststore.pem` file on the client host with the `/etc/pki/libvirt-spice/ca-cert.pem` file on the remote host.
2. The **spicec** command needs the host subject information stored in the certification file. Run the following command to retrieve it:

```
# SPICE_SUBJECT=`openssl x509 -noout -text \
    -in $HOME/.spicec/spice_truststore.pem | grep Subject: | cut -f 10- -d " "
```

3. Run the **spicec** command, replacing "kvmhost.company.org" with your KVM host name:

```
spicec -h kvmhost.company.org -p 5908 -s 5909 --host-subject "$SPICE_SUBJECT"
```

A SPICE session of your KVM guest opens.

VM security

You can use the sVirt service to isolate virtual machines, control groups (cgroups) to prevent denial-of-service situations, disk-image encryption to protect data at rest, and auditing to obtain valuable forensic information.

The sVirt service

Learn about how the sVirt service isolates virtual machines, how to optionally create static sVirt labels, and how to verify dynamic and static sVirt labeling.

Overview of the sVirt service

The sVirt service is an SELinux framework included in libvirt that isolates virtual machines.

The sVirt service defines unique labels for the processes and disk images of each guest operating system. These unique labels isolate each virtual machine from other virtual machines. This isolation prevents malicious users of one guest operating system from accessing the host and attacking the processes and resources of other guest operating systems.

sVirt labels are SELinux labels and SELinux uses mandatory access control (MAC). Therefore, the MAC security policy includes the sVirt labels. MAC is a means of restricting access to objects based on the sensitivity (as represented by a label) of the information contained in the objects and the formal authorization (clearance) of subjects to access information of such sensitivity. The Linux kernel enforces the MAC security policy and thus also enforces the sVirt labels.

When you enable the SELinux policy, the sVirt service automatically runs and dynamically creates and manages the labels. sVirt dynamic labeling is recommended in most cases. However, you can disable dynamic labeling and create your own static labels. In this case, you are responsible for the uniqueness of the labels.

Currently, Red Hat is pursuing Common Criteria at Evaluation Assurance Level (EAL) 4+ for Red Hat Enterprise Linux 6. This effort includes certifying the KVM hypervisor on both Red Hat Enterprise Linux 5 and Red Hat Enterprise Linux 6. The isolation provided by the sVirt service meets the security functional requirements for this certification.

Related information:

Red Hat Enterprise Linux 6, KVM to Pursue Security Certification

sVirt in Red Hat Enterprise Linux 6

The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments

Creating static sVirt labels

You can manually assign sVirt labels to the QEMU processes and disk images of a guest operating system.

Before you start, verify that SELinux is running in the enforcing mode by using the **getenforce** command:

```
$ getenforce
Enforcing
```

If the **getenforce** command shows that SELinux is running in permissive mode, you can change the mode to enforcing mode with the **setenforce** command:

```
$ setenforce 1
```

To create static sVirt labels, complete the following steps:

1. Edit the `domain.xml` file of the libvirt facility by typing the following command:

   ```
   $ virsh edit domainID
   ```

   where *domainID* is the ID of the domain of the guest operating system whose processes and disk images you want to label.
2. Add the following code to the `domain.xml` file:

   ```
   <seclabel type='static' model='selinux'>
    <label>system_u:system_r:svirt_t:s0:category,category</label>
    <imagelabel>system_u:object_r:svirt_image_t:s0:category,category</imagelabel>
   </seclabel>
   ```

   where *category,category* are the categories that you want to assign to the processes and disk images of the guest operating system. For example: `c32,c256`
3. Label all of the virtual images that the guest operating system uses by typing the following command for each virtual image:

   ```
   # chcon -t svirt_image_t -l s0:category,category /var/lib/libvirt/images/filename
   ```

   where:

   - *category,category* are the categories that you defined in Step 2. For example: `c32,c256`
   - *filename* is the name of a virtual image that the guest operating system uses. For example: `foo.img`
4. Start the guest operating system by typing the following command:

   ```
   $ virsh start domainID
   ```

   where *domainID* is the ID of the domain of the guest operating system that you want to start.

Verifying sVirt labeling by examining the labels

You can verify dynamic and static sVirt labeling by examining the labels that sVirt assigns to QEMU processes and disk images.

Before you start, complete the following tasks:

1. Verify that SELinux is running in the enforcing mode by using the **getenforce** command as follows:

   ```
   $ getenforce
   Enforcing
   ```

2. Verify that the guest operating system, for which you want to examine the labels, is running. To start the guest operating system, type the following command:

   ```
   $ virsh start domainID
   ```

   where *domainID* is the ID of the domain of the guest operating system that you want to start.

To verify sVirt labeling, complete the following steps:

1. Examine the label of the QEMU process by typing the following command:

   ```
   $ ps -wwC qemu-kvm -o label,command
   ```

   The output might look like the following output:

   ```
   LABEL                                COMMAND
   system_u:system_r:svirt_t:s0:c12,c23   /usr/bin/qemu-kvm
                                        ...
                                        -drive file=/var/lib/libvirt/images/rh-5.5.qcow2,if=virtio,\
                                        index=0,boot=on -drive file=/var/lib/libvirt/images/ \
                                        rh-5.5.img,if=virtio,index=1
                                        ...
   system_u:system_r:svirt_t:s0:c132,c511 /usr/bin/qemu-kvm
                                        ...
                                        -drive file=/var/lib/libvirt/images/foo.qcow2,if=virtio, \
                                        index=0,boot=on
                                        ...
   ```

   The output shows the following information:

   - The type is `svirt_t`.
   - The category is unique for each QEMU process. Each row in the output represents a QEMU process. The first row in the output shows that the sVirt service assigned the `c12,c23` categories to a process. The second row in the output shows that the sVirt service assigned the `c132,c511` categories to another process.
   - The disk images that each process uses are also shown. The first process uses the `/var/lib/libvirt/images/rh-5.5.qcow2` and `/var/lib/libvirt/images/rh-5.5.img` files. The second process uses the `/var/lib/libvirt/images/foo.qcow2` file.

2. Examine the labels of the disk images by typing the following command:

   ```
   $ ls --scontext /var/lib/libvirt/images/{filename,filename}
   ```

   where *filename,filename* is a list of files whose labels you want to view. For example: `rh-5.5.*,foo.qcow2`

   The output might look like the following output:

   ```
   system_u:object_r:svirt_image_t:s0:c12,c23 /var/lib/libvirt/images/rh-5.5.img
   system_u:object_r:svirt_image_t:s0:c12,c23 /var/lib/libvirt/images/rh-5.5.qcow2
   system_u:object_r:svirt_image_t:s0:c132,c511 /var/lib/libvirt/images/foo.qcow2
   ```

   The output shows the following information:

   - The type is `svirt_image_t`.
   - The sVirt service assigned the `c12,c23` categories to the `/var/lib/libvirt/images/rh-5.5.img` and `/var/lib/libvirt/images/rh-5.5.qcow2` files.
   - The sVirt service assigned the `c132,c511` categories to the `/var/lib/libvirt/images/foo.qcow2` file.

3. Verify that the labels of the disk images from Step 2 match the labels of the corresponding QEMU processes from Step 1. For example:

- Step 1 shows a process labeled with the c12,c23 categories. That process uses the /var/lib/libvirt/images/rh-5.5.qcow2 and /var/lib/libvirt/images/rh-5.5.img files. Step 2 shows that the /var/lib/libvirt/images/rh-5.5.qcow2 and /var/lib/libvirt/images/rh-5.5.img files are labeled with the same category as the process that uses the files.
- Step 1 shows a process labeled with the c132,c511 categories. That process uses the /var/lib/libvirt/images/foo.qcow2 file. Step 2 shows that the /var/lib/libvirt/images/foo.qcow2 file is labeled with the same category as the process that uses the file.

Verifying sVirt labeling by viewing the domain.xml file

You can verify dynamic and static sVirt labeling by viewing the domain.xml file.

Before you start, complete the following tasks:

1. Verify that SELinux is running in the enforcing mode by using the **getenforce** command as follows:
   ```
   $ getenforce
   Enforcing
   ```
2. Verify that the guest operating system, for which you want to examine the labels, is running. To start the guest operating system, type the following command:
   ```
   $ virsh start domainID
   ```
   where *domainID* is the ID of the domain of the guest operating system that you want to start.

When you use dynamic labeling, the libvirtd daemon updates the domain.xml file to reflect dynamic labeling.

To verify sVirt labeling, view the domain.xml file by typing the following command:

```
$ virsh dumpxml domainID | grep label
```

where *domainID* is the ID of the domain for which you want to verify sVirt labeling. The output might look like the following output:

```
<seclabel type='dynamic' model='selinux'>
 <label>system_u:system_r:svirt_t:s0:c132,c511</label>
 <imagelabel>system_u:object_r:svirt_image_t:s0:c132,c511</imagelabel>
</seclabel>
```

The output shows the following information:

- The sVirt service assigned the c132,c511 categories to the process.
- The sVirt service assigned the same categories, c132,c511, to the disk images that the process uses.

Control groups (cgroups)

With cgroups, you can restrict a set of tasks to a set of resources, prevent denial-of-service situations in KVM environments, and monitor resource use.

Each guest operating system uses some amount of system resources, including processing power, memory, disk, and networking bandwidth. If a guest operating system is malicious, it might use most or all of the system resources. In this situation, the other guest operating systems might become unresponsive because they no longer have access to the resources that they need.

To ensure that resources remain continuously available, you can control the amount of resources that each guest operating system can use by configuring control groups (cgroups). When you configure cgroups, you associate a set of tasks with the resources that the set of tasks can use. In other words, you can use cgroups to restrict a set of tasks to a set of resources. In a KVM environment, each guest operating system is considered a set of tasks. Therefore, you associate each guest operating system with a set of resources that the guest operating system can use.

For example, you can define the following resource controls:

- The total amount of memory that the set of tasks can use.
- The total amount of processing power that can be allocated to the set of tasks.
- The resource allocation priority when there is a shortage of resources.
- The amount of network bandwidth that a set of tasks can use.

In KVM environments, cgroups provide overall resource restrictions and prevent denial-of-service situations:

- Memory example: A user writes a task that continuously acquires memory until it acquires most or all of the memory. This task interferes with the work of other tasks because other tasks no longer have access to the memory that they need. To prevent this situation, you can configure cgroups to limit the amount of memory that the task can use.
- Processing power example: A user creates many tasks that use a large amount of processing power. These tasks decrease the overall system performance and interfere with the tasks of other users. To prevent this situation, you can configure cgroups to restrict the amount of processing power that a user can use.
- Network bandwidth example: A user streams a large amount of data from one or more Internet sites, which uses a large amount of network bandwidth. This task interferes with the workload of a customer. The customer workload cannot meet the service level agreement (SLA) because the workload does not have enough network bandwidth. To prevent this situation, you can use cgroups to control network bandwidth in one of the following ways:
  - You can restrict the amount of network bandwidth that a set of tasks can use.
  - You can allocate proportional bandwidth to sets of tasks. In other words, you can assign more network bandwidth to more important workloads and assign less network bandwidth to less important workloads.
- Disk example: A user creates a group that issues much I/O to a disk which uses a large amount of I/O bandwidth. These tasks decrease the overall disk I/O performance, and interfere with the work of other users and groups. Other users and groups no longer have access to all the I/O bandwidth that they need. To prevent this situation, you can configure cgroups to restrict the amount of I/O bandwidth a group can use.

cgroups also provide resource usage statistics. From the host, you can monitor resource usage as follows:

- You can monitor the resources that a particular guest operating system uses.
- You can monitor the resources across the system.

Monitoring resources by using cgroups can help you determine whether tasks meet their targets and get the resources that they need. If the statistics show that a guest operating system fails its target, you can reallocate resources accordingly.

You can use libcgroup and libvirt to configure and manage cgroups, and to collect resource usage statistics.

The libcgroup facility

The libcgroup facility provides an easy interface to working with cgroups.

In some distributions, for example Red Hat Enterprise Linux 6.2, the libcgroup facility is installed by default and creates cgroup mount points for all kernel subsystems, and a task hierarchy inside these mount points. If your distribution does not use the libcgroup facility, check its documentation for information about how to set up cgroups.

The `/proc/cgroups` file shows all the available subsystems, but you can access only the subsystems that were mounted. You can see the mount points of the subsystems in the `/etc/cgconfig.conf` file.

```
# cat /etc/cgconfig.conf
mount {
    cpuset  = /cgroup/cpuset;
    cpu     = /cgroup/cpu;
    cpuacct = /cgroup/cpuacct;
    memory  = /cgroup/memory;
    devices = /cgroup/devices;
    freezer = /cgroup/freezer;
    net_cls = /cgroup/net_cls;
    blkio   = /cgroup/blkio;
}
```

In each line containing an equals sign, the token before the equals sign is the name of a kernel subsystem as described in the `/proc/cgroups` pseudo-file. The token after the equals sign is the mount point for a subsystem.

Defining cgroups

You can define cgroups and their resource constraints in the `/etc/cgroups.conf` file.

To define cgroups, complete the following steps:

1. Define cgroups and their resource constraints with stanzas in the `/etc/cgroups.conf` file:

   ```
   # cat /etc/cgconfig.conf
   ...
   group group1 {
       cpu {
           cpu.shares = "256";
       }
       memory {
           memory.limit_in_bytes = 1M;
       }
   }
   ```

   This example defines two cgroups named “group1”: one for subsystem cpu, and another for subsystem memory. It also restricts a resource for each:

   cpu.shares
   : An integer value representing relative share of CPU time available to the tasks in a cgroup. For instance, tasks in a cgroup that specifies a cpu.shares = 1 have half the amount of cpu time that tasks in a cgroup with cpu.shares = 2 have. In this example, it restricts the shares to 256, which is 25% of the time of processes not in the group1. It is 25% of the time because the default **`cpu.shares`** value is 1024. You can check the value in the pseudo-file /cgroup/cpu/cpu.shares.

   memory.limit_in_bytes
   : The maximum amount of user memory, including file cache. If no units are specified, the value is interpreted as bytes. Use the suffix K or k to indicate kilobytes, the suffix M or m to indicate megabytes, and the suffix G or g to indicate gigabytes. In this example, it restricts the maximum memory to 1 megabyte.

2. Restart the cgconfig service to activate the contents of the `/etc/cgroups.conf` file.

   ```
   # /etc/init.d/cgconfig restart
   ```

If you examine the `/cgroup/memory` and `/cgroup/cpu` pseudo-directories, you see a new entry called “group1”. This entry confirms that a cgroup named “group1” is defined for those subsystems.

In the `/cgroup/memory/group1` pseudo-directory, there is a pseudo-file called “memory.limit_in_bytes”. It contains the value 1048576, which is the value of 1 megabyte converted to bytes.

In the `/cgroup/memory/group1` pseudo-directory, there is a pseudo-file called “cpu.shares”. It contains the value 256.

For more information about other resources that can be controlled, see the "Chapter 3. Subsystems and Tunable Parameters" section at http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Resource_Management_Guide/ch-Subsystems_and_Tunable_Parameters.html.

Adding tasks to cgroups

Add a task to a cgroup to restrict the resources that the task can use. A task must be added to specific subsystems in a cgroup.

To add tasks to cgroups, complete the following steps:

1. Determine the process ID of the task using a utility like the **ps** command.
2. Echo the process ID of the task to the `tasks` pseudo-file for a subsystem of a cgroup. This step must be repeated for each subsystem for which the task must be restricted.

   The path to a specific `tasks` pseudo-file is `/cgroup/<subsystem>/<cgroup-name>/tasks`.

   a. Add the task to the “memory” subsystem of the “group1” cgroup:

      ```
      # echo <task-process-id> >/cgroup/memory/group1/tasks
      ```

   b. Add the task to the “cpu” subsystem of the “group1” cgroup:

      ```
      # echo <task-process-id> >/cgroup/cpu/group1/tasks
      ```

You can see the cgroups to which a task belongs by examining the `cgroups` file for the task process id. The format of the path to this file for a specific task is `/proc/<task-process-id>/cgroups`. This example shows the cgroups to which the task with a process id of 6671 belongs:

```
# cat /proc/6671/cgroups
42:blkio:/
41:net_cls:/
40:freezer:/
39:devices:/
38:memory:/group1
37:cpuacct:/
36:cpu:/group1
35:cpuset:/
```

Defining cgroup hierarchy

Define cgroup hierarchy to organize and inherit subsystem restrictions. The cgroup hierarchy establishes parent-child relationships between cgroups.

To define cgroup hierarchy, complete the following steps:

1. In the `/etc/cgroups.conf` file, define a child cgroup of a parent cgroup by separating the name of the parent cgroup from the child cgroup with the '/' character:

   ```
   group parent/child {
       ...
   }
   ```

   Note: If you do not explicitly define a parent cgroup, it is defined for you.

   Note: Top-level parent cgroups you define are contained by an implicit cgroup known as the root control group.
2. Repeat the previous step as necessary to achieve the hierarchy and depth you want:

   ```
   group group1/group2/group3/.../groupN {
       ...
   }
   ```

Working with libvirt cgroups

In some distributions, for example Red Hat Enterprise Linux 6.2, libvirt automatically creates a cgroup hierarchy of its own.

The hierarchy is structured as follows:

```
root control group (implicit)
  |
  +-  libvirt    (all virtual guests/containers run by libvirtd)
        |
        +- qemu  (all QEMU/KVM containers run by libvirtd)
            |
            +-  guest1    (QEMU guest called 'guest1')
            +-  guest2    (QEMU guest called 'guest2')
            +-  guest3    (QEMU guest called 'guest3')
            +-  ...       (QEMU guest called ...)
```

With this hierarchy, you can fine-tune the resource usage of each guest managed by libvirt. The **virsh** command provides an easy interface to change resource attributes.

To set libvirt guest cgroup resource, run a **virsh** command similar to one of these examples:

- Set the **cpu.shares** resource of the libvirt guest named "guest1" to 256:

  ```
  # virsh schedinfo --set cpu_shares=256 guest1
  ```

- Set the **memory.limit_in_bytes** resource of the libvirt guest named "guest1" to 1 megabyte:

  ```
  # virsh memtune --hard-limit 1024 guest1
  ```

  Note: Not all subsystem resources can be changed with the **virsh** command. Check libvirt specific documentation to see which resources can be changed.

KVM guest image encryption

Disk-image encryption is a technique aimed at protecting data at rest – that is, when the system is powered off and the attacker somehow got access to its disks or other external storage, in what is commonly known as an "Offline Attack."

A similar attack can be performed on virtual machines, with one important difference: it can be performed even without physical access to the system. If an attacker manages to compromise a virtualization host or hypervisor (locally or remotely), they can then proceed to attack its VM guests. Despite the fact that not much can be done for running guests, the system administrator can protect guests that are not running by encrypting their disk-images, and requiring an encryption passphrase or key to start them. Not having possession of it, the attacker cannot easily decipher the disk-image data to access its contents.

In some systems, including Red Hat Enterprise Linux 6.2, protecting guest VM images at rest through disk encryption can be done by using **dm-crypt** (a Kernel-level device-mapper driver) and **cryptsetup** (a user space utility) to encrypt a raw disk-device or an LVM Logical Volume for use as guest raw image back-end storage. An LVM back-end storage provides more flexibility in terms of management (for example, capability to resize and relocate) than a raw disks solution.

Creating an encrypted Logical Volume target in the host

You can create an encrypted Logical Volume target in the KVM host and use it as the backend storage device for KVM guests.

Before you start, ensure that you satisfy the following requirements:

- The **cryptsetup-luks** package is installed on the system.
- The system has at least 10 GB (or the VM guest image size you desire) of free space in an already-configured LVM Volume Group.
- You are the root user of the KVM host system.

Complete the following steps:

1. Create a new Logical Volume device to serve as the backend block device to be encrypted. Choose a size suitable to fit your desired guest image. The example below creates a 10 GB Logical Volume named `guest01-crypt` in the `VolGroup00` Volume Group:

   ```
   # lvcreate -L 10G -n guest01-crypt VolGroup00
     Logical volume "guest01-crypt" created
   ```

   Note: By default, Red Hat Enterprise Linux distributes all disk space across its partitions during installation. This might lead to insufficient space to create your backend device logical volume in VolGroup00, requiring you to shift space from other partitions or volume groups to VolGroup00.

   The following example steps illustrate freeing 10 gigabytes of space from logical volume lv_home in volume group vg_myhost:

   a. Reduce the size of the filesystem contained in the logical volume. The size value is the new size of the filesystem. In this example, the original size is 71 gigabytes. To free 10 gigabytes, resize the filesystem to 61 gigabytes.

      ```
      fsadm resize /dev/mapper/vg_myhost-lv_home 61G
      ```

   b. Reduce the size of the logical volume by 10 gigabytes.

      ```
      lvreduce -L -10G /dev/vg_myhost/lv_home
      ```

   For more information about making unused storage in other partitions or volume groups available in VolGroup00, see *Resizing Linux partitions (parts 1 and 2)* on the IBM®developerWorks® website at http://www.ibm.com/developerworks/linux/library/l-resizing-partitions-1/index.html?ca=dgr-lnxw97LXPartitionsdth-LX and http://www.ibm.com/developerworks/linux/library/l-resizing-partitions-2/index.html?ca=drs-.

2. Use the **cryptsetup luksFormat** command to set up the Logical Volume for encryption. This command will ask for confirmation and a passphrase used to unlock the device. The encryption protection will ultimately depend on the strength of the passphrase, so the system administrator must take extra care in choosing it and securely guarding it against unauthorized accesses. Losing or forgetting the passphrase will lead to data loss of the entire Logical Volume. There are many options in terms of safeguarding against passphrase loss or using larger, more secure keys to perform the encryption, but those are not within the scope of this blueprint. For more information about other safeguarding options, see Red Hat's documentation.

   ```
   # cryptsetup luksFormat /dev/VolGroup00/guest01-crypt

   WARNING!
   ========
   This will overwrite data on /dev/VolGroup00/guest01-crypt irrevocably.

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

3. Use the **cryptsetup luksDump** command to check that the device has been formatted for encryption successfully:

   ```
   # cryptsetup luksDump /dev/VolGroup00/guest01-crypt
   LUKS header information for /dev/VolGroup00/guest01-crypt

   Version:        1
   Cipher name:    aes
   Cipher mode:    cbc-essiv:sha256
   Hash spec:      sha1
   Payload offset: 1032
   MK bits:        128
   MK digest:      c9 79 da e3 a2 47 6d 06 02 8d 91 57 94 df c5 2c ee e3 af d8
   MK salt:        4a 8d 38 bc b1 4e 3e 18 3b f7 15 1e 9b 4b c6 87
                   54 0a fe 4e ff e7 fa ce c6 ce c8 87 a9 7c 45 c5
   MK iterations:  10
   UUID:           ad37e741-edef-4bb9-823f-b93562d49042
   ```

```
Key Slot 0: ENABLED
        Iterations:             408349
        Salt:                   ae 16 54 ec 3d fc ab fc 8b 4d 58 9b 4b eb eb f4
                                58 7a 93 be 07 51 f3 7d e1 6d 28 0c 8a 96 35 88
        Key material offset:    8
        AF stripes:             4000
Key Slot 1: DISABLED
Key Slot 2: DISABLED
Key Slot 3: DISABLED
Key Slot 4: DISABLED
Key Slot 5: DISABLED
Key Slot 6: DISABLED
Key Slot 7: DISABLED
```

4. Now the encrypted backend device needs to be opened using the passphrase to be usable. The example below uses the **cryptsetup luksOpen** command to open the *guest01-crypt* Logical Volume and create a *guest01-img* target device (already unencrypted). The passphrase configured previously is required:

   ```
   # cryptsetup luksOpen /dev/VolGroup00/guest01-crypt guest01-img
   Enter LUKS passphrase for /dev/VolGroup00/guest01-crypt:
   ```

5. The encrypted block device is now opened and the unencrypted counterpart is available for use. You can use the **dmsetup info** command to query more information about the device (*/dev/mapper/guest01-img* in this example):

   ```
   # dmsetup info guest01-img
   Name:              guest01-img
   State:             ACTIVE
   Read Ahead:        256
   Tables present:    LIVE
   Open count:        0
   Event number:      0
   Major, minor:      253, 3
   Number of targets: 1
   ```

   You can use the block device just like any other regular block device until you close it, from which point you would need to reopen the device with the **cryptsetup luksOpen** command.

For more information about setting up LVM, see *Logical Volume Management* on the IBM developerWorks website at http://www.ibm.com/developerworks/linux/library/l-lvm2/.

Migrating existing guests to encrypted storage

To migrate an existing guest (`kvm01`) to use the encrypted storage created earlier (`/dev/VolGroup00/guest01-crypt` with unencrypted counterpart `/dev/mapper/guest01-img`), you can perform a cold migration, where the migration is done when the guest is shut down.

Before you start, set up a Logical Volume Management (LVM) that contains at least one volume group with sufficient free space to hold a guest image. For instructions, see "Creating an encrypted Logical Volume target in the host" on page 35.

1. Make sure the guest you want to migrate is not running. If it is, shut down the guest by issuing the following command:

   ```
   # virsh shutdown kvm01
   ```

   Check that the guest was correctly shut down with **virsh list** command. It might take a few minutes for the guest to shut down properly.

   ```
   # virsh list
    Id Name                 State
   ----------------------------------
   ```

   Note: The guest is shut down when it does not show up as running in the above output.

2. Find the current location of the disk image for the guest to be migrated. Use the **virsh dumpxml** command to dump the guest's configuration by inspecting the <disk> tags under the <device> section:

```
# virsh dumpxml kvm01
.......
<devices>
  <emulator>/usr/libexec/qemu-kvm</emulator>
  <disk type='file' device='disk'>
    <driver name='qemu' cache='none'/>
    <source file='/var/lib/libvirt/images/kvm01'/>
    <target dev='vda' bus='virtio'/>
  </disk>
  <disk type='file' device='cdrom'>
    <driver name='qemu' cache='none'/>
    <source file='/root/RHEL5.5-Server-20100322.0-x86_64-DVD.iso'/>
    <target dev='hdc' bus='ide'/>
    <readonly/>
  </disk>
.......
```

   This example has the primary hard-disk image placed at **/var/lib/libvirt/images/kvm01**.

3. Use the **qemu-img info** command to check the disk's image size and format:

```
# qemu-img info /var/lib/libvirt/images/kvm01
image: /var/lib/libvirt/images/kvm01
file format: raw
virtual size: 10G (10737418240 bytes)
disk size: 10G
```

   Note that the standard installation methods create KVM guest images in the raw format, as stated by the `file format` line in the above output. Other formats will need additional converting which can be accomplished with the **qemu-img convert** command. Conversion between formats is not within the scope of this blueprint. Consult your product manual for additional information.

4. Perform a raw copy from the image file to the encrypted target device with the **dd** utility:

```
# dd if=/var/lib/libvirt/images/kvm01 of=/dev/mapper/guest01-img bs=4k
2621440+0 records in
2621440+0 records out
10737418240 bytes (11 GB) copied, 159.602 seconds, 67.3 MB/s
```

   Note: The **dd** utility should report successful copying of exactly the same number of bytes as the `virtual size` reported by the **qemu-img info** command, and this is usually sufficient to check if the copy was completed. Note that the use of hashing tools such as **md5sum** and **sha1sum** to verify integrity is usually not possible due to size differences between the original image file and the encryption target device. The size discrepancy comes mainly from LVM rounding (to the smallest number of physical extents), but also due to the additional size added to fit the encryption header.

5. Use the **virsh edit** command to change the disk configuration of the guest to point to its new storage location. Note the changes to the `disk type` from `file` to `block`, and the disk `source` to `dev` instead of `file` (the listing below shows the difference between the original configuration and the new one, in unified diff format):

```
--- kvm01.xml.orig 2010-05-14 16:26:35.000000000 -0300
+++ kvm01.xml 2010-05-14 16:27:21.000000000 -0300
@@ -19,9 +19,9 @@
<on_crash>restart</on_crash>
<devices>
  <emulator>/usr/libexec/qemu-kvm</emulator>
-  <disk type='file' device='disk'>
+  <disk type='block' device='disk'>
     <driver name='qemu' cache='none'/>
-    <source file='/var/lib/libvirt/images/kvm01'/>
+    <source dev='/dev/mapper/guest01-img'/>
     <target dev='vda' bus='virtio'/>
  </disk>
  <disk type='file' device='cdrom'>
```

6. Start the guest normally using the **virsh start** command:

```
# virsh start kvm01
Domain kvm01 started
```

Installing a new guest

To install a new KVM guest, instruct your libvirt management tool to use the encrypted target.

Before you start, set up a Logical Volume Management (LVM) that contains at least one volume group with sufficient free space to hold a guest image. For instructions, see "Creating an encrypted Logical Volume target in the host" on page 35.

This examples uses the **virt-install** command to install a guest into the `/dev/mapper/guest01-img` device that was previously opened with the **cryptsetup luksOpen** command. For more information about how to install KVM guests, see the *Quick start guide to KVM* and *The developer's approach to installing and managing KVM* blueprints at http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaai/liaaivirtualization.htm.

```
# virt-install --name guest01 --ram 512 --accelerate \
 --location /mnt/rhel5-u5/ \
 --extra-args "console=tty0 console=ttyS0,115200n8" \
 --os-type linux --os-variant rhel5 \
 --disk path=/dev/mapper/guest01-img,bus=virtio,device=disk \
 --disk path=/root/RHEL5.5-Server-20100322.0-x86_64-DVD.iso,bus=ide,device=cdrom \
 --network network:default --nographics

Starting install...
Retrieving file .treeinfo...     |  442 B     00:00
Retrieving file vmlinuz...       | 1.9 MB     00:00
Retrieving file initrd.img...    | 7.7 MB     00:00
Creating domain...               |    0 B     00:00
Connected to domain guest01
Escape character is ^]
Linux version 2.6.18-194.el5 (mockbuild@x86-005.build.bos.redhat.com) \
(gcc version 4.1.2 20080704 (Red Hat 4.1.2-48)) #1 SMP Tue Mar 16 21:52:39 EDT 2010
Command line:  console=tty0 console=ttyS0,115200n8
...
```

Storing encrypted guest images

In order to be protected against "Offline attacks," the system administrator must make sure to shut down any guests that are not in use and close their cryptographic storages. The manual step that is opening them again while providing a passphrase or key is exactly what protects those images from attacks.

To shut down the guest and close its encrypted image, complete the following steps:

1. Use the **virsh shutdown** command to stop the guest:

   ```
   # virsh shutdown kvm01
   Domain kvm01 is being shutdown
   ```

2. Before proceeding to close its encrypted image, make sure that the guest is no longer running by checking with the **virsh list** command. It might take a few minutes for the guest to shut down properly:

   ```
   # virsh list
    Id Name                 State
   ----------------------------------
   ```

   Note: The guest is shut down when it does not show up as running in the **virsh list** command output.

3. Close the encrypted device:

   ```
   # cryptsetup luksClose guest01-img
   ```

After the encrypted device is closed, it can be moved or copied to another location as with any other raw block device.

Auditing the KVM virtualization host and guests

In a production environment, even with security controls correctly in place, it is often necessary to track changes and interactions in the system. This tracking data can provide a trail to audit the system security and can be valuable forensic information in the advent of an information breach.

Recent Linux versions include the Linux audit service which is composed of a kernel component, a daemon that writes audit logs to disk, and userspace components (libraries, tools) that provide searching, parsing, and reporting facilities. In some systems, including Red Hat Enterprise Linux 6.2, the kernel component is built-in and the userspace components are deployed through the audit package.

This section presents a set of auditing rules to serve as an example, along with some brief explanations and useful tips. This does not replace the need to read and understand the original documentation from your Linux distribution.

Enabling host auditing

You can enable auditing on the KVM host.

Complete the following steps:

1. Configure the kernel to initialize auditing early in the boot process by editing the `/boot/grub/menu.lst` file and appending the"audit=1" kernel parameter to every kernel entry:

   ```
   password --md5 $1$eBtcO/$XjczkZLiy1vehilCUXENm/
   title Linux
   root (hd0,1)
   kernel /vmlinuz-el5.x86_64 ro root=/dev/sda1 rhgb audit=1
   initrd /initramfs-el5.x86_64.img
   ```

2. Reboot for the changes to take effect.
3. Verify that the audit package is installed and that the auditd service is set to automatically start on boot:

   ```
   # service auditd start
   auditd (pid 2843) is running...
   # chkconfig auditd on
   # chkconfig --list auditd
   auditd 0:off 1:off 2:on 3:on 4:on 5:on 6:off
   ```

Audit rules file

The definition for what is and what is not audited in a Linux system is typically a mix between SELinux Policy and the configured auditing rules.

For more information about SELinux, see the *First Steps with Security-Enhanced Linux (SELinux): Hardening the Apache Web Server* blueprint at http://publib.boulder.ibm.com/infocenter/lnxinfo/v3r0m0/topic/liaai/selinux/liaaiselinuxstart.htm.

The audit rules file at `/etc/audit/audit.rules` is a simple list of command-line arguments, one set per line, to be passed to the **auditctl** tool. Because the rules are evaluated sequentially from the oldest to newest (or equivalently from top to bottom in the rules file), their order of appearance is important.

The following example `audit.rules` file is aimed at:

- monitoring changes to the virtualization environment configuration
- auditing any suspicious or out of the ordinary activities, such as the reading of private keys, attempts at changing the logs, or attempts at changing the virtual machine images by entities outside the qemu_t domain.

Any type of auditing always results in an impact to performance. You must fine-tune the auditing rules for a good balance between traceability and performance requirements for the environment. Check the Performance Tips section of the **auditctl** man page for specific information about how to optimize your rules.

A copy of the rule file is located in "Sample audit rules file" on page 43. The following discussion explains the rules in this example rule file.

The following are the default rules:

```
# First rule - delete all.
-D

# Increase the buffers to survive stress events.
# Make this buffer bigger for busy systems.
-b 8192
```

The following lines are actually a set of "don't audit" rules covering operations that are considered normal or routine in the KVM host. Every qemu-kvm instance must have access to the KVM device node itself (`/dev/kvm`), to the KSM (`/dev/ksm`), and to pseudo-terminal devices (`/dev/ptmx` and `/dev/pts`). The qemu-kvm instance is represented by the **qemu_t** SELinux domain because in normal situations, only the qemu-kvm emulator and related utilities (such as **qemu-img**) are allowed to run in such domain.

```
#####
### "Don't audit" rules - explicit exclusions for more generic rules following them.
# Don't audit qemu read/writes to necessary devices.
-a exit,never -F path=/dev/kvm -F perm=rw -F subj_type=qemu_t
-a exit,never -F path=/dev/ksm -F perm=rw -F subj_type=qemu_t
-a exit,never -F path=/dev/ptmx -F perm=rw -F subj_type=qemu_t
-a exit,never -F dir=/dev/pts -F perm=rw -F subj_type=qemu_t
```

The following lines denote another set of exceptions for programs (represented by their respective SELinux domains) that interact with qemu or libvirt data in the same way:

```
# Don't audit dnsmasq writing to libvirt network runtime data.
-a exit,never -F dir=/var/run/libvirt/network -F perm=wa -F subj_type=dnsmasq_t

# Don't audit logrotate writing to logs.
-a exit,never -F dir=/var/log/libvirt/ -F perm=wa -F subj_type=logrotate_t

# Don't audit initrc_t domain writing to temporary storage data.
-a exit,never -F dir=/var/cache/libvirt/ -F perm=wa -F subj_type=initrc_t
```

(Optional) The following lines are a set of rules directed at auditing any domain other than qemu-kvm or libvirtd daemon attempts to read or change the TLS private keys, followed by a rule auditing any changes to the TLS certificates themselves.

Note: If you did not set up the TLS keys, do not include this set of rules in your audit rules.

```
#####
### Audit access attempts to TLS private keys.
-a exit,always -F path=/etc/pki/libvirt/private/serverkey.pem -F subj_type!=virtd_t -k virt_tls_privkey
-a exit,always -F path=/etc/pki/libvirt-vnc/server-key.pem -F subj_type!=qemu_t -k virt_tls_privkey

#####
### Audit attempts at changing libvirt and qemu certificates (both server and CA).
-a exit,always -F path=/etc/pki/CA/cacert.pem -F perm=wa -k virt_tls_cert
-a exit,always -F path=/etc/pki/libvirt/servercert.pem -F perm=wa -k virt_tls_cert
-a exit,always -F path=/etc/pki/libvirt-vnc/ca-cert.pem -F perm=wa -k virt_tls_cert
-a exit,always -F path=/etc/pki/libvirt-vnc/server-cert.pem -F perm=wa -k virt_tls_cert
```

The following rules audit changes to files related to **libvirt** configuration:

```
######
### Audit any changes to libvirt configuration.
-a exit,always -F dir=/etc/libvirt/ -F perm=wa -k virt_libvirt_cfg
-a exit,always -F path=/etc/sysconfig/libvirtd -F perm=wa -k virt_libvirt_cfg
-a exit,always -F path=/etc/sasl2/libvirt.conf -F perm=wa -k virt_libvirt_cfg
```

The rules file then defines a set of rules to audit any attempt at changing the virtualization environment, any attempt of a process in the **qemu_t** domain to change or interact with resources outside its domain, and any attempts at changing runtime data, logs, or executables related to the virtualization environment:

```
######
### Audit every attempt of qemu_t interaction with another domain, unless not
### explicitly excluded previously.
-a exit,always -F arch=b32 -S all -F perm=wax -F subj_type=qemu_t -F obj_type!=qemu_t -k virt_qemu_crossdomain
-a exit,always -F arch=b64 -S all -F perm=wax -F subj_type=qemu_t -F obj_type!=qemu_t -k virt_qemu_crossdomain

######
### Audit changes to virtual images from outside the qemu_t domain.
-a exit,always -F dir=/var/lib/libvirt/images/ -F perm=wa -F subj_type!=qemu_t -k virt_image_change
-a exit,always -F obj_type=virt_image_t -F perm=wa -F subj_type!=qemu_t -k virt_image_change

######
### Audit changes to the qemu/libvirt runtime data (exceptions above).
-a exit,always -F dir=/var/run/libvirt/ -F perm=wa -F subj_type!=virtd_t -k virt_runtime_change
-a exit,always -F dir=/var/lib/libvirt/ -F perm=wa -F subj_type!=virtd_t -k virt_runtime_change
-a exit,always -F dir=/var/cache/libvirt/ -F perm=wa -F subj_type!=qemu_t -k virt_runtime_change

######
### Audit changes to the qemu/libvirt logs (exceptions above).
-a exit,always -F dir=/var/log/libvirt/ -F perm=wa -F subj_type!=virtd_t -k virt_log_change

######
### Audit every libvirtd daemon execution.
-a exit,always -F path=/usr/sbin/libvirtd -F perm=x -k virt_libvirtd_exec

######
### Audit every libvirtd daemon executable change.
-a exit,always -F path=/usr/sbin/libvirtd -F perm=wa -k virt_libvirtd_change

######
### Audit every qemu execution.
-a exit,always -F path=/usr/libexec/qemu-kvm -F perm=x -k virt_qemu_exec

######
### Audit every qemu executable change.
-a exit,always -F path=/usr/libexec/qemu-kvm -F perm=wa -k virt_qemu_change
```

To use this example file as your audit rule file:

1. Copy the example file from "Sample audit rules file" on page 43.
2. Paste the content to the `/etc/audit/audit.rules` file.
3. Restart the auditd daemon:

```
# /etc/init.d/auditd restart
Stopping auditd:                                           [  OK  ]
Starting auditd:                                           [  OK  ]
```

You can specify an audit rule with a key using the **-k** flag. This key can be used as a filter by reporting tools. All keys in the example rules file are prefixed with `virt_`. This example shows how to use the **ausearch** command to display only events related to virtualization (given that the previous example `audit.rules` file is being used):

```
# ausearch -i -k "virt_"
type=PATH msg=audit(04/21/2010 04:02:05.977:165) : item=1 name=/var/log/libvirt/qemu/kvm1.log.1.gz \
inode=10223664 dev=fd:00 mode=file,600 ouid=root ogid=root rdev=00:00 obj=user_u:object_r:virt_log_t:s0
```

```
type=PATH msg=audit(04/21/2010 04:02:05.977:165) : item=0 name=/var/log/libvirt/qemu/ inode=10223653 \
dev=fd:00 mode=dir,700 ouid=root ogid=root rdev=00:00 obj=system_u:object_r:virt_log_t:s0
type=CWD msg=audit(04/21/2010 04:02:05.977:165) :  cwd=/
type=SYSCALL msg=audit(04/21/2010 04:02:05.977:165) : arch=x86_64 syscall=open success=yes exit=4 \
a0=7fff265986c0 a1=242 a2=8180 a3=312e676f6c2e316d items=2 ppid=4156 pid=4158 auid=root uid=root \
gid=root euid=root suid=root fsuid=root egid=root sgid=root fsgid=root tty=(none) ses=10 \
comm=logrotate exe=/usr/sbin/logrotate subj=user_u:system_r:unconfined_t:s0 key=virt_log_change
----
type=PATH msg=audit(04/21/2010 04:02:05.978:166) : item=0 name=(null) inode=10223664 dev=fd:00 \
mode=file,600 ouid=root ogid=root rdev=00:00 obj=user_u:object_r:virt_log_t:s0
type=SYSCALL msg=audit(04/21/2010 04:02:05.978:166) : arch=x86_64 syscall=fchmod success=yes exit=0 \
a0=4 a1=180 a2=8180 a3=312e676f6c2e316d items=1 ppid=4156 pid=4158 auid=root uid=root gid=root \
euid=root suid=root fsuid=root egid=root sgid=root fsgid=root tty=(none) ses=10 comm=logrotate \
exe=/usr/sbin/logrotate subj=user_u:system_r:unconfined_t:s0 key=virt_log_change
----
..
```

Sample audit rules file

You can copy the contents of this sample audit rules file for use in your environment.

The following auditd rule file is an example of the file discussed in "Audit rules file" on page 40.

```
# First rule - delete all
-D

# Increase the buffers to survive stress events.
# Make this bigger for busy systems
-b 8192

#####
### Don't audit rules - explicit exclusions for more generic rules after
# Don't audit Qemu read/writes to necessary devices
-a exit,never -F path=/dev/kvm -F perm=rw -F subj_type=qemu_t
-a exit,never -F path=/dev/ksm -F perm=rw -F subj_type=qemu_t
-a exit,never -F path=/dev/ptmx -F perm=rw -F subj_type=qemu_t
-a exit,never -F dir=/dev/pts -F perm=rw -F subj_type=qemu_t

# Don't audit dnsmasq writing to libvirt network runtime data
-a exit,never -F dir=/var/run/libvirt/network -F perm=wa -F subj_type=dnsmasq_t

# Don't audit logrotate writing to logs
-a exit,never -F dir=/var/log/libvirt/ -F perm=wa -F subj_type=logrotate_t

# Don't audit initrc_t domain writing to temporary storage data
-a exit,never -F dir=/var/cache/libvirt/ -F perm=wa -F subj_type=initrc_t

#####
### Audit access attempts to TLS private keys
-a exit,always -F path=/etc/pki/libvirt/private/serverkey.pem -F subj_type!=virtd_t -k virt_tls_privkey
-a exit,always -F path=/etc/pki/libvirt-vnc/server-key.pem -F subj_type!=qemu_t -k virt_tls_privkey

#####
### Audit attempts at changing libvirt and Qemu certificates (both server and CA)
-a exit,always -F path=/etc/pki/CA/cacert.pem -F perm=wa -k virt_tls_cert
-a exit,always -F path=/etc/pki/libvirt/servercert.pem -F perm=wa -k virt_tls_cert
-a exit,always -F path=/etc/pki/libvirt-vnc/ca-cert.pem -F perm=wa -k virt_tls_cert
-a exit,always -F path=/etc/pki/libvirt-vnc/server-cert.pem -F perm=wa -k virt_tls_cert

######
### Audit any changes to libvirt configuration
-a exit,always -F dir=/etc/libvirt/ -F perm=wa -k virt_libvirt_cfg
-a exit,always -F path=/etc/sysconfig/libvirtd -F perm=wa -k virt_libvirt_cfg
-a exit,always -F path=/etc/sasl2/libvirt.conf -F perm=wa -k virt_libvirt_cfg

######
### Audit every attempt of qemu_t interaction with another domain, unless not
### explicitly excluded above
-a exit,always -F arch=b32 -S all -F perm=wax -F subj_type=qemu_t -F obj_type!=qemu_t -k virt_qemu_crossdomain
-a exit,always -F arch=b64 -S all -F perm=wax -F subj_type=qemu_t -F obj_type!=qemu_t -k virt_qemu_crossdomain
```

```
######
### Audit changes to virtual images from outside qemu_t domain
-a exit,always -F dir=/var/lib/libvirt/images/ -F perm=wa -F subj_type!=qemu_t -k virt_image_change
-a exit,always -F obj_type=virt_image_t -F perm=wa -F subj_type!=qemu_t -k virt_image_change

######
### Audit changes to qemu/libvirt runtime data (exceptions above)
-a exit,always -F dir=/var/run/libvirt/ -F perm=wa -F subj_type!=virtd_t -k virt_runtime_change
-a exit,always -F dir=/var/lib/libvirt/ -F perm=wa -F subj_type!=virtd_t -k virt_runtime_change
-a exit,always -F dir=/var/cache/libvirt/ -F perm=wa -F subj_type!=qemu_t -k virt_runtime_change

######
### Audit changes to qemu/libvirt logs (exceptions above)
-a exit,always -F dir=/var/log/libvirt/ -F perm=wa -F subj_type!=virtd_t -k virt_log_change

######
### Audit every libvirtd execution
-a exit,always -F path=/usr/sbin/libvirtd -F perm=x -k virt_libvirtd_exec

######
### Audit every libvirtd executable change
-a exit,always -F path=/usr/sbin/libvirtd -F perm=wa -k virt_libvirtd_change

######
### Audit every Qemu execution
-a exit,always -F path=/usr/libexec/qemu-kvm -F perm=x -k virt_qemu_exec

######
### Audit every Qemu executable change
-a exit,always -F path=/usr/libexec/qemu-kvm -F perm=wa -k virt_qemu_change
```

libvirt auditing support

libvirt can record its own audit records.

In some distributions, for example Red Hat Enterprise Linux 6.2, libvirt auditing is enabled by default, depending on whether host auditing is enabled.

libvirt records three audit records:

VIRT_CONTROL
: Generated for guest life cycle events (shutdown and start).

VIRT_RESOURCE
: Generated for guest resources changes.

VIRT_MACHINE_ID
: Generated when the SELinux label is assigned to a guest. This is useful to an administrator who needs to track SELinux Access Vector Cache (AVC) denial records related to the same guest. AVC denial records are generated when a process tries to violate the existing MAC policies.

Enabling libvirt auditing

You can enable libvirt auditing.

Complete the following steps:

1. Edit the `/etc/libvirt/libvirtd.conf` file and change the following keys:

```
--- libvirtd.conf 2011-11-29 09:31:55.117512128 -0500
+++ libvirtd.conf 2012-01-10 18:39:23.427148188 -0500
 #   audit_level == 1  -> enable auditing, only if enabled on host (default)
 #   audit_level == 2  -> enable auditing, and exit if disabled on host
 #
-#audit_level = 0
+audit_level = 1
 #
 # If set to 1, then audit messages will also be sent
```

```
 # via libvirt logging infrastructure. Defaults to 0
 #
-audit_logging = 0
+audit_logging = 1
```

2. Restart the libvirtd service:

```
# /etc/init.d/libvirtd restart
```

Displaying libvirt audit records

You can display libvirt audit records with the **auvirt** command.

Note: To use auvirt support, you must install the audit 2.2.1+ and libvirt 0.9.10+ packages.

The **auvirt** command displays libvirt audit records, showing a list of virtual machine sessions. It also displays other events such as host shutdowns, AVC denials related to guests, and anomaly events associated with QEMU processes.

To display libvirt audit records, complete the following steps:

Run the **auvirt** command without options to display default information about libvirt and related audit records.

```
# auvirt
Fedora                  root        Tue Jan 24 16:28 - 16:28  (00:00)
Fedora                  root        Wed Jan 25 10:31 - 10:34  (00:02)
CentOS                  root        Wed Jan 25 10:34 - 13:50  (03:16)
Fedora                  root        Wed Jan 25 13:51 - 13:54  (00:03)
Fedora                  root        Wed Jan 25 13:54 - 13:55  (00:00)
Arch                    root        Wed Jan 25 13:55 - failed
Arch                    root        Wed Jan 25 13:56
```

The information includes:

- guest name
- user who started the guest
- time range during which the guest ran

Note in the example that one execution of the "Arch" guest failed and that another is still running.

Displaying libvirt audit records by time

You can display libvirt audit records by the time they were recorded.

Run the **auvirt** command with the **--start** and **--end** options. These options limit the output to a specific time range. For example:

```
# auvirt --start 01/25/2012 12:00:00 --end 01/25/2012
Fedora                  root        Wed Jan 25 13:51 - 13:54  (00:03)
Fedora                  root        Wed Jan 25 13:54 - 13:55  (00:00)
Arch                    root        Wed Jan 25 13:55 - failed
Arch                    root        Wed Jan 25 13:56
```

If the time portion of the date and time argument is omitted, it defaults to "00:00:00" for the **--start** option and "23:59:59" for the **--end** option.

Displaying libvirt audit records by guest

You can display libvirt audit records by guest they are associated with.

Perform either of the following steps:

- Run the **auvirt** command with the **--vm** option to limit the output to a specific guest name:

```
# auvirt --vm Fedora
Fedora                  root        Tue Jan 24 16:28 - 16:28  (00:00)
Fedora                  root        Wed Jan 25 10:31 - 10:34  (00:02)
Fedora                  root        Wed Jan 25 13:51 - 13:54  (00:03)
Fedora                  root        Wed Jan 25 13:54 - 13:55  (00:00)
```

- Run the **auvirt** command with the **--uuid** option to limit the output to a specific guest UUID. Include the **--show-uuid** option to include UUID information in the output:

```
# auvirt --show-uuid --uuid cabf9532-99f1-3756-930f-59e8f19d4144
Arch                    cabf9532-99f1-3756-930f-59e8f19d4144    root         Wed Jan 25 13:55 - failed
Arch                    cabf9532-99f1-3756-930f-59e8f19d4144    root         Wed Jan 25 13:56
```

Displaying additional libvirt audit record information

You can display additional libvirt audit record information.

Complete the following steps:

Run the **auvirt** command with the **--all-events** option to display all virtualization-related events:

```
# auvirt --vm CentOS --all-events
down           root   Wed Jan 25 08:34
res   CentOS   root   Wed Jan 25 10:34 - 13:50  (03:16)
   cgroup      allow    major   rw   pty

[...]

avc   CentOS   root   Wed Jan 25 10:34
   relabelto   denied   libvirtd     CentOS.img
   system_u:object_r:svirt_image_t:s0:c6,c883

[...]

res   CentOS   root   Wed Jan 25 10:34 - 13:50  (03:16)
   disk        start    /var/lib/libvirt/images/CentOS.img
res   CentOS   root   Wed Jan 25 10:34 - 13:50  (03:16)
   net         start    54:52:00:5b:1a:cd
res   CentOS   root   Wed Jan 25 10:34 - 13:50  (03:16)
   mem         start    1048576
res   CentOS   root   Wed Jan 25 10:34 - 13:50  (03:16)
   vcpu        start    1
start CentOS   root   Wed Jan 25 10:34 - 13:50  (03:16)

[...]

stop  CentOS   root   Wed Jan 25 13:50
```

The first field indicates the event type, which can have the following values: “start”, “stop”, “res”, “avc”, “anom”, and “down”. “down” indicates a host shutdown.
Resource assignments and AVC records provide additional information:

- The different types of resources that can be assigned to a guest, for example disks, virtual CPUs, memory, network devices. Restricting resources with cgroups is also logged as a resource event. Resource types include vcpu, mem, disk, net, and cgroup.
- Which action caused the resource assignment event. For example, a disk can be assigned to a guest when the guest is started or when it is running. In such cases, the action is “start” or “attach”, respectively. For cgroup events, the action information indicates if a resource is being allowed or denied. In that case, the value is “allow” or “deny”, respectively.
- The resource that is assigned to a guest. This is a path for disk resources, megabytes for memory, a MAC address for network devices, and the number of VCPUs for virtual CPUs. For cgroup events, this value includes three components: type, ACL, and the resource.

The AVC events displayed are denied accesses performed by the guest, and denied accesses to guest resources performed by another process on the host.
AVC events provide additional information:

- The denied operation performed by the process. Values include “read”, “write”, “open”, “relabelto”, and “relabelfrom”.
- An operation result, which usually has a value of “denied”.
- The program name of the process that tried to perform the denied operation.
- The target of the denied operation. For example, if a process inside the host tries to access a guest disk image file and its access is denied by SELinux, this field contains the disk image file name.
- Additional context information pertaining to the “relabelto” and “relabelfrom” operations. This indicates the related security context, which is the target context for “relabelto” operation, and the source context for “relabelfrom” operation.

Displaying raw libvirt audit record information

You can display the raw audit record content of libvirt audit records.

Complete the following steps:

1. Run the **auvirt** command with the **--proof** option to display audit event identifiers associated with each record:

   ```
   # auvirt --vm Fedora --start 01/23/2012 00:00 --end 01/23/2012 17:30 --proof
   Fedora                       root        Mon Jan 23 17:02 - 17:27  (00:24)
       Proof: 1327345338.087:41631, 1327346831.548:41702
   ```

   In this output, the audit event identifiers are the integers following the ':' characters in the line beginning with “Proof:”, namely 41631 and 41702. In this example, the time range selected with the **--start** and **--end** options displays a range of audit event identifiers from 41631 through 41702.
2. Run the **ausearch** command with the **--event** option to display the entire content of a specific audit record:

   ```
   # ausearch -a 41631
   ----
   time->Mon Jan 23 17:02:18 2012
   type=VIRT_CONTROL msg=audit(1327345338.087:41631): user pid=2433 uid=0
    auid=4294967295 ses=4294967295 subj=system_u:system_r:initrc_t:s0 msg='virt=kvm
    op=start reason=booted vm="Fedora" uuid=7bcae45c-c179-6cea-a185-c7abe50520c7:
    exe="/usr/sbin/libvirtd" hostname=? addr=? terminal=? res=success'
   ```

Maintaining virtual machines

To maintain the security of the virtual machines, apply the general principles and best practices recommended by the National Security Agency.

For information, see the Guide to the Secure Configuration of Red Hat Enterprise Linux 5. This guide provides detailed information and instructions about the following security practices:

- Encrypt transmitted data whenever possible
- Minimize software to minimize vulnerability
- Run different network services on separate systems
- Configure security tools to improve system robustness
- Grant the least privilege necessary for user accounts and software to perform tasks

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Dept. LRAS/Bldg. 903
11501 Burnet Road
Austin, TX 78758-3400
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® and ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Printed in USA