

# QEMU/KVM Hardening

---

## Check AppArmor setup

To ensure your Debian host and Debian guests are fully secured, let's pick up where we left off with the AppArmor verification. Run the clean filter command on your host:

```
# aa-status | grep -E "enforce|qemu"
```

What to check next based on your output:

- Scenario A: QEMU processes are under “processes are in enforce mode” Your current VMs are successfully sandboxed. You are ready to move on to other hardening steps like CPU Pinning or resource limits.
- Scenario B: QEMU processes are under “processes are unconfined” Your VMs are running unprotected. We need to fix the qemu.conf abstraction layers or check if virt-aa-helper is failing due to custom disk paths (e.g., if your VM disks are stored outside of /var/lib/libvirt/images).
- Scenario C: You see profile names but no active processes This means the security profiles exist, but no Debian guest VMs are currently powered on.

Normally on recent Debian versions AppArmor is automatically enabled. If needed you can also force it in /etc/libvirt/qemu.conf by removing the comment for the following line:

```
security_driver = "apparmor"
```

After this restart libvirt-daemon:

```
# systemctl restart libvirtd.
```

## CPU Security Mitigations and Architecture (CPU Passthrough)

If your Virt-Manager configuration uses a generic CPU model (like qemu64), the Debian guest kernel does not know the exact physical architecture of your host CPU. Here is why this is important for your single-VM Debian Trixie setup:

### The Risk: Side-Channel Vulnerabilities (Spectre / Meltdown)

If your Virt-Manager configuration uses a generic CPU model (like qemu64), the Debian guest kernel does not know the exact physical architecture of your host CPU.

\* The Problem: Without knowing the exact hardware, the Debian guest cannot apply the correct kernel-level mitigations for CPU vulnerabilities (such as Spectre, Meltdown, or Downfall). \* The Threat:

A malicious process running inside your Debian guest could potentially exploit these hardware flaws to break out of the virtual machine boundary and read the memory of your host system.

### How to check your current CPU model:

1. Open Virt-Manager and open your VM details (lightbulb icon).
2. Click on the CPUs section on the left.
3. Look at the Model field.

### How to harden it:

Change the model to host-passthrough.

\* What it does: It tells QEMU to mirror your physical host CPU exactly into the guest, including all safety flags and microcode extensions. \* Why it's secure: The Debian guest kernel will immediately detect your exact CPU type and automatically load the correct security patches and mitigations at boot. \* Bonus: It significantly improves performance because the guest can use your CPU's native instructions (like AES-NI for hardware-accelerated encryption).

## Network containing

To limit a KVM guest to only reach one specific gateway, you must block all other traffic using iptables or nftables on the Linux host. Because virt-manager does not have a built-in button for this, the cleanest way is to use a libvirt hook script. This script automatically applies network restrictions every time the VM starts.

### Step 1: Find your VM's network interface

Run this command on your host to find the virtual interface name while the VM is running and note the Interface name (e.g., vnet0 or vnet1):

```
# virsh domiflist <Your-VM-Name>
-----
Interface    Type      Source    Model    MAC
-----
vnet2        network  default   virtio    41:52:e0:7e:d3:aa
```

### Step 2: Create a Libvirt hook script

Libvirt can execute scripts when VMs start. We will use a qemu hook to isolate the traffic. Open a terminal on your host and create the hook directory if it does not exist.

```
# mkdir -p /etc/libvirt/hooks/qemu.d
```

Create and edit the hook file and paste the following script (replace vnet0 and 192.168.122.1 with your actual interface and allowed gateway):

```
# nano /etc/libvirt/hooks/qemu.d/vm-gateway-only-access
-----

#!/bin/bash

set -euo pipefail

VM_NAME="Debian13-ESP32-ARMHF"
VM_MAC="52:54:00:7e:d3:ee"

LIBVIRT_BRIDGE="virbr0"
ALLOWED_IPV4="192.168.178.1"

NFT="/usr/sbin/nft"

TABLE="vm_filter"
CHAIN="vm_${VM_NAME//[a-zA-Z0-9_]/_}"

install_rules()
{
    "$NFT" delete table inet "$TABLE" 2>/dev/null || true
    "$NFT" add table inet "$TABLE"
    "$NFT" add chain inet "$TABLE" "$CHAIN" \
        '{ type filter hook forward priority -300; policy accept; }'

    # Allow only VM -> 192.168.178.1
    "$NFT" add rule inet "$TABLE" "$CHAIN" \
        iifname "$LIBVIRT_BRIDGE" \
        ether saddr "$VM_MAC" \
        ip daddr "$ALLOWED_IPV4" \
        accept

    # Drop everything else originating from this VM.
    # This includes IPv4 and IPv6.
    "$NFT" add rule inet "$TABLE" "$CHAIN" \
        iifname "$LIBVIRT_BRIDGE" \
        ether saddr "$VM_MAC" \
        drop
}

remove_rules()
{
    "$NFT" delete table inet "$TABLE" 2>/dev/null || true
}

case "${2:-}" in
    start)
        [[ "$1" == "$VM_NAME" ]] && install_rules
        ;;
end
```

```
stopped)
    [[ "$1" == "$VM_NAME" ]] && remove_rules
    ;;
esac

exit 0
```

### Step 3: Make the script executable and restart libvirt

Make the script executable:

```
# chmod +x /etc/libvirt/hooks/qemu
```

Restart the libvirt service to load the new hook:

```
# systemctl restart libvirtd
```

Now, every time you start this specific VM, the host firewall will block all network traffic unless it is explicitly sent to or from your chosen gateway IP. If you are using nftables instead of iptables, or if you need help finding the exact subnet/gateway IP of your virt-manager network, let me know!

Check if the firewall settings are created and removed with:

```
# nft list ruleset
```

From:  
<https://wiki.oscardegroot.nl/> - HomeWiki

Permanent link:  
<https://wiki.oscardegroot.nl/doku.php?id=linux:apps:kvm:kvm-hardening&rev=1789311838>

Last update: **2026/09/13 15:03**

