USB Auto-Attach in Qubes » Saswat Padhi

Saswat Padhi ⋮ 11-14 minutes ⋮ 9/6/2021

I have been using Qubes as my primary OS[1] for a while now, and I am extremely happy with the level of control it provides me over my apps. Recently someone raised an interesting question on the Qubes OS forum:

> Hello,
>
> Is there a way to attach a particular USB device (by Vendor ID/Product ID) to a Qube automatically on connect? I have a sys-usb to manage USB devices.
>
> The USB device I am using frequently becomes disconnected (and reconnected) during use and it only needs to be connected to one particular Qube.
>
> Thanks,
> N
>
> https://forum.qubes-os.org/t/usb-device-auto-attach-to-qube/5977

▾ But what exactly is ?

Typically, is a minimal VM to which all USB controllers are assigned. So all USB devices are attached to when plugged in, and not to that has direct access to rest of the hardware. Attaching untrusted USB devices to is a potentially fatal security risk. Many ready-to-use implementations of common attacks already exist, e.g. the USB Rubber Ducky. Therefore, it's strongly recommended to use a dedicated USB VM, and only passthrough trusted devices to VMs that need to access them.

Below are some excellent resources on understanding the rationale behind using :

- Qubes OS documentation on device security
- Qubes OS documentation on USB qubes
- Discussion on Qubes OS Forum

Also note that running applications, such as your password manager etc. directly in defeats the whole point of using Qubes OS — *compartmentalization*. A malicious USB device, if connected, could potentially extract all of your application data. So, should only be used as a handler, and only *trusted* devices should be connected to application VMs (called *AppVMs* in Qubes).

I have wondered about this sort of automation before. I have a bunch of USB devices: external disks, crypto wallets, password managers, etc. which I attach to specific VMs immediately after plugging in. It would be nice to be able to attach trusted devices to certain VMs automatically.

On a "regular" OS, such as Debian or Fedora, triggering actions on connecting / disconnecting USBs devices is typically achieved using udev rules. We could match on the Vendor ID / Model ID etc. to detect when a particular USB device is added or removed, and may `RUN` a desired script on that event. However, this isn't as straightforward with on Qubes OS, since it typically would have no access to other running VMs and cannot attach devices to them directly.

POTENTIAL SECURITY RISK

Before trying out the proposed changes on your system, please remember that we are *sacrificing security*

for convenience. If implemented incorrectly, or without proper constraints, you might end up unintentionally attaching potentially harmful devices to your VMs **automatically**!

I proposed an initial idea on how this could be done, and this weekend I finally had some time to try it out 🙂 The high-level plan is outlined below:

1. Since cannot escape virtualization and directly attach devices to other VMs, we must somehow communicate via the *AdminVM* (i.e.,), which has access to all VMs.
2. Communication in Qubes typically happens using an elegant remote procedure call (RPC) framework, called Qrexec, so we must write some new Qrexec service that receives a device attachment request and fulfills it.
3. Our Qrexec service will be listening in , and a client must pass the target VM name and device name to this service from a udev trigger.
4. Finally, we must also setup an appropriate Qrexec policy for this service in .

If you are unsure of what RPC / Qrexec / udev are, then I would strongly suggest familiarizing yourself with those tools first before making any changes to your system.

ADVANCED QUBES / LINUX STUFF

DO NOT copy-paste any code or shell commands from the internet, unless you understand exactly what they do. You may compromise the security of your system, or damage it otherwise.

Changes in

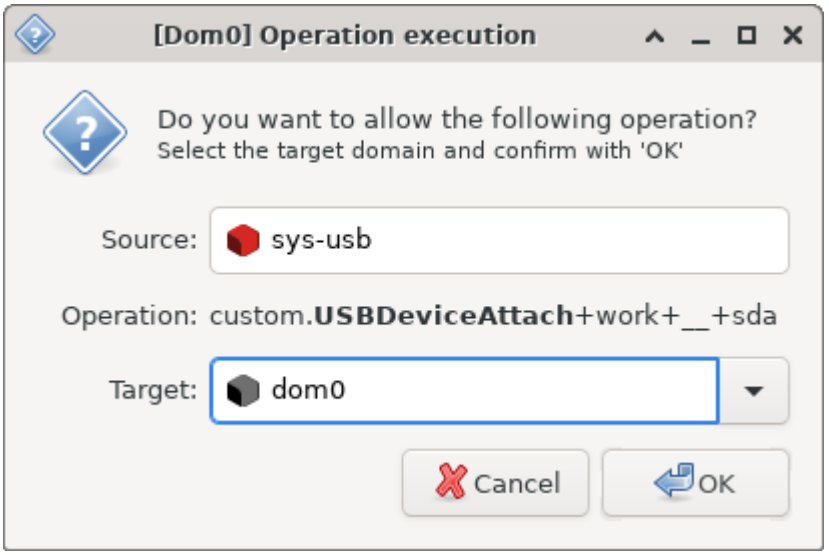
There are two main changes necessary in :
(a) a new Qrexec service to listen to device attachment requests from ,
(b) a Qrexec policy to restrict the source and destination VMs for calls to this service.

The Qrexec Service

First, let's start with the new Qrexec service that would handle requests from . I am only going to describe how to automatically attach block devices, but attaching full USB devices is very similar (see comment at the end of this section).

There are a few different ways of implementing this service. One important choice to make is regarding the mode of communication with . Only stdin/stdout is passed between the Qrexec server and client — in particular, there is no support for command-line arguments in Qrexec. However, instead of communicating over stdin/stdout, I chose to use a *service argument*, which is actually passed as a single *command-line argument* to the service. A service argument, unlike stdin/stdout communication, is visible on the prompt, so the communication is a bit more transparent. Moreover the service argument is also *sanitized* by Qrexec, before being supplied to the script.

However, since Qrexec only allows a single service argument currently, we must *pack* our target VM name and device name together. An example prompt from my should be displayed below. `custom.USBDeviceAttach` is the service that we are going to create, and `work+__+sda` is the service argument. In this case, I pack the target VM name (`work`) and block device name (`sda` within) together by concatenating both using `+__+`.



A prompt showing argument `work+__+sda` for `custom.USBDeviceAttach` Qrexec call.

The service script itself is pretty simple, as I show below.

```
#!/usr/bin/env bash

# Exit immediately if any of the following commands fail
set -e

# Expect exactly one argument -- the service argument
[ "$#" -eq 1 ]

# Unpack service argument and expect exactly 2 strings
ARG=($(sed 's;+__+; ;g' <<< $1))
[ "${#ARG[@]}" -eq 2 ]

# Make sure we have a remote VM name from Qrexec
[ -n "$QREXEC_REMOTE_DOMAIN" ]

# Start the target VM, if needed
qvm-start --skip-if-running "${ARG[0]}"

# Attach the target device from remote VM to target VM
qvm-block attach "${ARG[0]}" "${QREXEC_REMOTE_DOMAIN}:${ARG[1]}"
```

To make it available to Qrexec as a new service, this script must exist under `/etc/qubes-rpc` and must be marked as executable (`chmod +x`). I used `/etc/qubes-rpc/custom.USBDeviceAttach`, for instance.

As mentioned before, tweaking this script to work for full USB passthrough is easy — the `qvm-block` in line 20 could be changed to `qvm-usb`. However, full USB passthrough is less secure.

The Qrexec Policy

For each Qrexec service, we must also specify a Qrexec policy to whitelist calls from specific VMs. In this case, we want to allow calls, and deny all other calls:

```
sys-usb dom0 ask,default_target=dom0
```

```
@anyvm @anyvm deny
```

If you have multiple USB qubes, perhaps with different USB controllers assigned to each one, you might create copies of the first line and modify accordingly. The last line prevents all other (non-USB) qubes from triggering this Qrexec call.

This file should exist under `/etc/qubes-rpc/policy`, and must have the same name as the service file. I used `/etc/qubes-rpc/policy/custom.USBDeviceAttach`, for instance.

The `ask` in the first line could be changed to `allow` to skip the prompt, and allow all requests from , but beware that this is a significant security risk. A compromised may request to attach potentially harmful devices to arbitrary VMs, and these requests would be fulfilled without without any user approval!

Changes in

Finally, in we need to trigger our new Qrexec service when USB devices are connected to automatically attach them to desired VMs. But, before automating this process with udev, we should first manually test the RPC by running:

```
qrexec-client-vm dom0 custom.USBDeviceAttach+my-vm+__+sda
```

where should be a block device (e.g., `/dev/sda`) in , and should be some target VM. If everything has been setup correctly, should display a prompt, similar to one in my screenshot above. On approving this request in , the device should be correctly attached to the target VM, which can be checked by running `qvm-block` in .

If everything works correctly, then this Qrexec call could be automated via a udev rule in . As an example, I have the following rule in a new file `/etc/udev/rules.d/50-auto-attach.rules`:

```
ACTION=="add", SUBSYSTEM=="block", KERNEL=="sd[a-z]", \
RUN+="/bin/sh -c 'qrexec-client-vm dom0 custom.USBDeviceAttach+my-vm+__+
%k &'"
```

This rule will match new block devices with kernel names matching `sd[a-z]` regex, so partitions such as `sda1` etc. wouldn't match this rule. Upon a successful match, the `RUN` command is executed, and is requested to attach the device to . Beware that this rule is not very secure and must be constrained with additional `ATTRS` or `ENV` values of the particular device to match on. An introduction to Udev is an excellent quick tutorial on this topic.

Unblocking udev

Observe that the `RUN` command has an & at the end, and thus runs in a detached shell. This is not a performance optimization, but is *necessary* to resolve a dependency cycle:

1. udev stays blocked on `RUN` scripts — a USB device is connected and available in only after this script terminates
2. However, the Qrexec call triggered by the run script expects the device to already be available in , and attempts to attach it to !

To resolve this dependency cycle, we run the Qrexec call in a detached shell and unblock udev. By the time processes the RPC, receives user approval (via the prompt), and attempts to attach the USB device to another VM, udev should have finished connecting the device (usually instantaneous).

This (using &) is not the *correct* way to detach scripts from udev. There are multiple threads online regarding this, e.g., see: this StackOverflow thread, this AskUbuntu thread, and this blogpost. However, this approach suffices for our case, because Qrexec does not require the client to stay alive after making the RPC. So although udev would kill the detached script after connecting the USB devices, would have received the call and would carry out the requested operation. might show a warning popup if the client is killed before finishes attaching the device and returns a success code to client:

A warning shown if the client is killed before returns a success code.

However, this warning is completely harmless. Any of the solutions proposed in the above threads could be implemented to properly detach the script and get rid of the warning.

Persistent udev Rules

Typically is an application VM (called *AppVM* in Qubes). Since the root filesystem for AppVMs is discarded on shutdown and reset to the corresponding *TemplateVM* root filesystem, the new rules saved under `/etc/udev/rules.d/` would not persist across reboots. Therefore, the `50-auto-attach.rules` file should be saved in the AppVM's persistent storage (within `/rw`), and must be loaded into udev during boot. For instance, I have saved it at `/rw/config/50-auto-attach.rules` and have the following in my `/rw/config/rc.local` init script:

```
cp /rw/config/*.rules /etc/udev/rules.d/

udevadm control --reload-rules && udevadm trigger
```

Furthermore, for persistent DispVM configuration, the changes above must be made in the DispVM template, and not directly.