Secure Desktops with Qubes: Extra Protection

Find out how to make Qubes even more secure with a few advanced features.

KYLE RANKIN

Kyle Rankin is a Sr. Systems Administrator in the San Francisco Bay Area and the author of a number of books, including *The Official Ubuntu Server Book*, *Knoppix Hacks* and *Ubuntu Hacks*. He is currently the president of the North Bay Linux Users' Group.

PREVIOUS
Dave Taylor's Work the Shell

NEXT
Shawn Powers' The Open-Source Classroom

THIS ARTICLE IS THE FOURTH IN MY SERIES ABOUT THE QUBES OPERATING SYSTEM, a security-focused Linux distribution that compartmentalizes your common desktop tasks into individual VMs. In the previous articles, I gave a general introduction to Qubes, walked through the installation process and discussed how I personally organize my own work into

different appVMs. If you haven't read these earlier articles, I suggest you do so before diving in here. In this article, I focus on some of the more advanced security features in Qubes, including split-GPG, the usbVM and how I control where URLs open.

Split GPG

One of the concerns I've always had with my Linux desktop was with the fact that I had a copy of my GPG key on the desktop so I could use it for my email. Of course, my key was password-protected, but I still worried that if my computer ever were compromised, an attacker still could grab it and try to brute-force it.

You can use a number of different techniques to protect GPG keys, including methods that store the master key offline in an air-gapped system while you use subordinate keys. Qubes offers a pretty novel approach to GPG key security with its split GPG system that acts kind of like a poor man's Hardware Security Module. With split GPG, you store your GPG key in a highly trusted appVM, such as the vault VM Qubes installs by default. The vault has no network card at all and is designed to store sensitive files like GPG keys. When an application wants to access the GPG key to encrypt or decrypt a file, instead of calling GPG directly, it calls a wrapper script that is included in the Qubes default templates. This script asks for permission to access the vault GPG key for a limited amount of time (which means you get an unspoofable prompt on your desktop from the vault), and if you grant it permission, it will send the GPG input (data to encrypt or decrypt, for instance) to the vault. The vault then will perform the GPG operation and send the output back to the appVM. In this way, the key always stays within the vault, and the appVM never sees it.

Split GPG is relatively simple to set up with the main challenge being that because it relies on a wrapper script, any application that calls out to GPG will need to be changed to point to a script like qubes-gpg-client or qubes-gpg-client-wrapper (the former works on the command line, which preserves environment variables, and the latter works better in programs that don't preserve them well, like email clients). Qubes has great documentation on split GPG on its website that includes some diagrams, detailed explanations of how it works and its limitations, and a how-to

guide with sample configs for Thunderbird and Mutt.

The first step is to make sure that the qubes-gpg-split package is installed in your appVMs (you should be able to use your regular package manager to install it if it isn't) and that qubes-gpg-split-dom0 is installed in dom0 (if not run `sudo qubes-dom0-update qubes-gpg-split-dom0` in a dom0 terminal).

Once the software is installed, put your GPG keyring into the appVM you want to use as your vault. I recommend the default vault unless you have more advanced and specific reasons you want to use another appVM. Execute some GPG command-line commands from a terminal in the vault (such as `gpg -K`) to confirm that the GPG key is installed correctly.

Now to use split GPG, just set the `QUBES_GPG_DOMAIN` environment variable to the name of the appVM that has your GPG key, and then you should be able to run qubes-gpg-client from that appVM with the same kind of arguments you normally would pass GPG. For applications like mail clients that may not be able to load that environment variable, you must use qubes-gpg-client-wrapper instead. This script is configured to read the contents of the /rw/config/gpg-split-domain file to determine which appVM to use, so be sure it contains the name of your vault:

```
$ sudo bash -c 'echo vault > /rw/config/gpg-split-domain'
```

That's pretty much it for basic GPG usage. The one major use case it doesn't cover is importing keys from an appVM back to the vault. You want to import keys in a trusted way, so Qubes provides a different script for this purpose that will prompt you from the vault in an unspoofable window before it imports the key. To import a key, just use this:

```
$ export QUBES_GPG_DOMAIN=vault
$ qubes-gpg-import-key somekey.asc
```

That should be enough to get you started with split GPG, but if you need particular split GPG configuration examples for applications like Thunderbird, Mutt, and Git, I recommend checking out the Qubes split GPG documentation page at https://www.qubes-os.org/doc/split-gpg.

USB VM

One of the major risks of compromise against a personal computer is the USB port. You can find a large number of stories on the internet about organizations (including governments) who were compromised because someone plugged in an untrusted USB key. There are even some fun hardware projects out there like the USB Rubber Ducky that provide what looks like an innocent USB thumbdrive but can act like a USB input device when you plug it in, and with its scripting language, you can program it to type whatever compromising keystrokes you want against your victim (including waiting some time later before unleashing your payload).

Given that just about anyone can create a malicious USB device now, you definitely want to be careful about what USB devices you plug in. Even Qubes installs may suffer the same risk, because by default, the dom0 VM is assigned the USB PCI controllers, so if you mistakenly plug in an infected USB key, it could potentially compromise your whole machine. Thankfully, Qubes provides a countermeasure for this with the option of creating a special USB VM that is assigned all of your USB PCI devices. With the USB VM in place, if an attacker plugs a malicious USB device in to your computer while you are away (or you plug it in yourself), the damage is contained to the USB VM.

Of course, if all of your USB devices are now assigned strictly to one VM, how can you use them on your other appVMs? For input devices like mice and keyboards, Qubes provides an input proxy service that will proxy input devices to the rest of the appVMs provided the user accept a prompt when the devices are plugged in. When you plug in a USB storage device, it shows up only in the USB VM for starters, and you then can assign it to other appVMs in the Qubes VM Manager by right-clicking on the appVM and selecting the device from the attach/detach block devices menu (be sure to detach it before you unplug it, otherwise Xen has been known to get confused about the state of the block device).

If you do want to enable the USB VM, the sys-usb USB VM shows up as an option during the install on the screen where you select which default appVMs to load. Otherwise, if you want to try it out post-install, you can run the following commands from the dom0 VM (Qubes 3.1 or newer):

```
$ qubesctl top.enable qvm.sys-usb
$ qubesctl state.highstate
```

These commands will run through an automated Salt script the Qubes team has put together that will configure the sys-usb VM appropriately. Of course, if you want to do this all by hand, you also could just create your own sysVM (I recommend not giving it a network card if you can help it), and in the Qubes VM Manager, go into that VM's settings and identify and assign your PCI USB controllers to it.

Now, there's a reason that sys-usb is disabled by default in the installer. Although desktop computers still offer PS/2 ports and many laptops use PS/2 as the interface for their main keyboard and mouse, some laptops (such as current MacBooks for instance) use a USB interface for the main keyboard. If that's the case, you can end up with a situation where you are locked out of your computer, because your USB keyboard will be assigned to your USB VM at boot, and you won't be able to log in. Another downside is that although there are services to share input devices and storage devices with other appVMs, any other USB devices (such as webcams or network cards) cannot be shared and can be used only from applications within the USB VM. Finally, the USB VM is unstable on some kinds of hardware, depending on how well it supports Qubes.

By default, only mice are allowed through the Qubes input proxy (and then only if you accept a prompt). Keyboards are not allowed through by default, because of the extra risk a malicious keyboard input device can pose to a system, including the fact that the USB VM can then read anything you type on that keyboard in other appVMs (such as passwords) or could enter its own keystrokes. If you are willing to accept this risk, you still can provide a level of protection by ensuring that you are prompted before an arbitrary USB keyboard is shared with the rest of the environment. In a dom0 terminal, add the following line to /etc/qubes-rpc/policy/qubes.InputKeyboard:

```
sys-usb dom0 ask
```

In this case, I specified sys-usb, but if you use a different appVM as your USB VM, use its name here instead.

The one big challenge you may find if you use a USB VM is in video conferences. Since basically every webcam shows up as a USB device, and there currently is no way to share that USB device with other VMs, you

are stuck using your webcam with apps in the USB VM only. A further challenge is that, by default, the sys-usb VM has no network access, and because it's a system VM type, you can't just point it to your sys-net VM to grant it network access. Instead, if you want to video conference with a USB VM, you unfortunately must replace the existing sys-usb VM with a new one that is either an appVM or proxyVM so it can get network access. Then you will have to install and run your video conferencing software from within that VM.

The major downside to this approach is that the USB VM is traditionally considered an untrusted VM much like sys-net, yet video conferencing is a somewhat trusted application, because you have to provide login credentials. Whether the security risk is worth the benefit is something you will have to decide for yourself based on your threat model. Fortunately in the future, there is hope that once a new USB virtualization feature hits Xen (and Qubes), you may be able to treat webcams like input or storage devices within Qubes.

URL Handlers

One final enhancement you can make with your Qubes desktop that not only helps with security but also ease of use is the Qubes CLI tools to open URLs in different appVMs. One common use case for this is to configure your email client to open attachments in a disposable VM (something Qubes covers for mail clients like Mutt on its main documentation page). In addition to that, I like to set the default URL handler for appVMs that aren't meant to run a web browser to be my general-purpose untrusted appVM. The untrusted appVM has no personal files or anything really of value to me on it, and I don't enter any login credentials into that appVM. Therefore, I can destroy it and re-create it at any time.

The default URL handler is a setting that you have to make in each appVM. The way I set it is to go to the shortcuts editor for that appVM (by clicking the "add more shortcuts..." option in that appVM's desktop menu) and adding the Preferred Applications program to that appVM's shortcuts. This is the same program that you use on a regular desktop to choose Firefox or Chrome as your default web browser. Once I've added the shortcut, I then launch the program, and in the section where I can

set the web browser (Figure 1), I choose Other so I can type in my own custom command, and in that field, I type:

```
qvm-open-in-vm untrusted "%s"
```

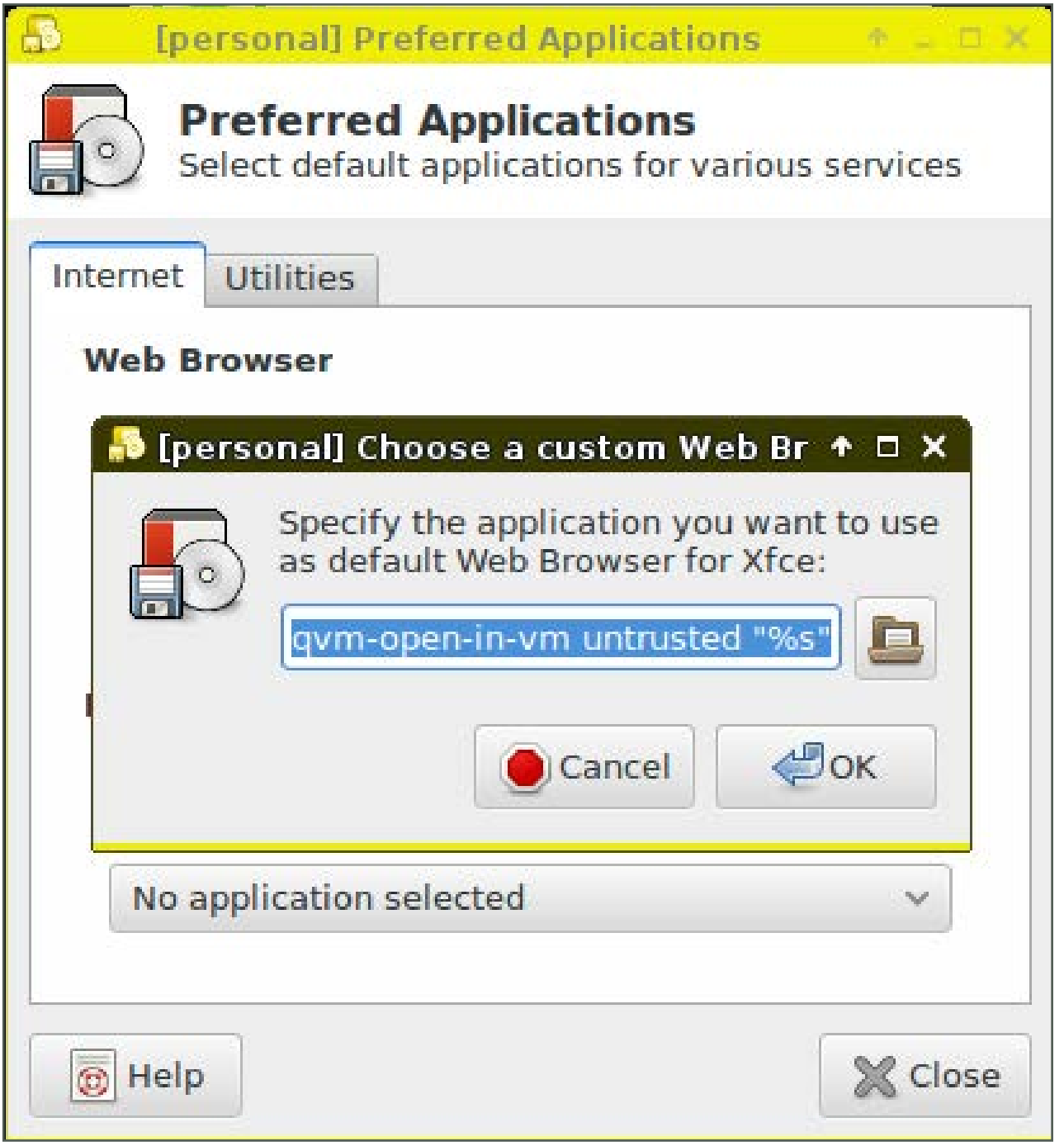

Figure 1. Changing the Default URL Handler

Once I save that, the next time I click on a URL within that appVM (for instance, in a chat session), I will get a prompt from the untrusted appVM to allow this appVM to open a URL there. You either can allow that every time or select "Yes to All" to allow this appVM to open URLs in untrusted without prompting permanently. I use a similar method to open URLs from my KeePassX password manager in my personal-web appVM instead of untrusted by going into the KeePassX settings and configuring a custom URL handler there instead. If you would rather open URLs in a disposable VM, just change this command to be:

```
qvm-open-in-dvm "%s"
```

Although Qubes definitely is very secure by default, these additional settings will help you lock it down even further, and each one gives a prime example of how the principle of compartmentalization can help you isolate and protect your computer. This is the last article I intend to write in this Qubes series for now; however, I likely will come back to more specific Qubes how-tos in the future.■

Send comments or feedback via http://www.linuxjournal.com/contact or to ljeditor@linuxjournal.com.

RETURN TO CONTENTS