Qubes in tmpfs 🤫

xuy ⋮ 91-115 minutes ⋮ 9/7/2023

April 28, 2022, 10:13am 1

You can use your QubesOS `stateless` just like TailsOS, with **persistent storage** for VMs. That is pretty simple! It takes 6Gb of extra `RAM` (for store root filesystem files).

The steeps:

1. Install QubesOS, boot to it and make base configuration: screen resolution, keyboard layout, etc.
2. Edit kernel parameters variables at grub settings file `/etc/default/grub`:

```
GRUB_CMDLINE_LINUX="... rd.break"
GRUB_CMDLINE_XEN_DEFAULT="... dom0_mem=max:10240M ..."
```

The `rdbreak` option - drop to a shell at the end (see `man dracut`).

3. Generate grub2 config and reboot PC:

```
sudo grub2-mkconfig | sudo tee /boot/efi/EFI/grub.cfg
reboot
```

4. "`Press Enter for maintinance`", then copy files from SSD to `RAM`:

```
umount /sysroot
mkdir /mnt
mount /dev/mapper/qubes_dom0-root /mnt
mount -t tmpfs -o size=100% none /sysroot
cp -a /mnt/* /sysroot
```

- Press Ctrl-D to continue QubesOS boot-up.

  Hooya! Your `QubesOS working in RAM`.

You can create a `dracut` module to automate the steep four, if that makes sense.

Then the volume

1. Mount a special `hidden` partition to /opt
2. Create qubes VMs files at varlibqubes pool

```
qvm-create -P varlibqubes --class TemplateVM --label black debian-10-pool
qvm-create -P varlibqubes --template debian-10-pool --label blue darknet-
```

```
i2p
```

3. Change the path via symlinks to access the VMs:

```
sudo rm -Rf /var/lib/qubes/vm-templates/ ; ln -s /opt/vm-templates/ /var/
lib/qubes/
sudo rm -Rf /var/lib/qubes/appvms/ ; ln -s /opt/appvms /var/lib/qubes/
```

- In the /opt directory should be a VM files created earlier, with identical to current VMs names.

You should like to configure the system

1. Add bash aliases

```
echo '
alias qvm-clone="qvm-clone -P varlibqubes"
alias qvm-create="qvm-create -P varlibqubes"
' >> $HOME/.bashrc
```

2. Configure AppVMs

```
lspci
qvm-pci attach --persistent --verbose vmname dom0:06_00.0

qvm-prefs --set vmname ip 10.137.0.81
qvm-prefs --set vmname netvm none
qvm-prefs --set vmname provides_network true
qvm-prefs --set vmname memory 800
qvm-prefs --set vmname maxmem 8000
```

Good luck!

NOTE: Install QubesOS updates from normal persistent mode (not from `RAM` mode).

References

1. Linux - Load your root partition to `RAM` and boot it - Tutorials - reboot.pro: _Linux - Load your root partition to RAM and boot it - Tutorials - reboot.pro
   _Linux - Load your root partition to RAM and boot it - Tutorials - reboot.pro
2. `Dracut` Wiki: _Home · dracutdevs/dracut Wiki · GitHub
3. Deniable encryption · Issue #2402 · QubesOS/qubes-issues · GitHub: _Deniable encryption · Issue #2402 · QubesOS/qubes-issues · GitHub
4. AMD Memory Encryption — The Linux Kernel documentation: _14. AMD Memory Encryption — The Linux Kernel documentation

xuy April 28, 2022, 10:43am 2

One little detail, for sync the app shorcuts with a template:
`qvm-sync-appmenus debian-10-pool`

alzer89 April 28, 2022, 8:13pm 3

This could be written as a `systemd` service that could read the kernelopts for something like `rd.qubes.run_in_ram`.

I will post a sample `systemd` service file for automating this by the end of next week.

This could prove very handy when paired with a veracrypt drive.

Fantastic job!

deeplow April 29, 2022, 8:34am 4

This is awesome! Thank you for writing this!

Could this be used to revive the Qubes live USB project?

51lieal April 29, 2022, 1:47pm 5

I think it could be included in initramfs, would be great for testing, thanks.

alzer89 May 3, 2022, 12:16am 6

Why yes, yes it could

An update on getting this to work. Those commands that wrote seem to be for a specific hardware configuration and software personal preference.

I'm trying to get it to parse an existing Qubes OS configuration, so that it'll recreate the entire configuration in RAM, as opposed to just a "default config".

This might take longer than a week, but I'll post up what I have by the end of the week anyway

deeplow May 4, 2022, 10:03am 7

Awesome!

deeplow May 4, 2022, 10:03am 8

(renamed the title for it to be ascii. Otherwise people won't find it by searching)

enmus May 4, 2022, 12:00pm 9

This is so interesting, thanks.

Since the OP compared it to TailOS, what about the footprins on the SSD? I'm not sure I can perceive it.

51lieal May 7, 2022, 12:23pm 10

I think for the live usb project is not simply as above, the op guide is just for "already" installed qubes, if you want to open an enhancement issue I will join the discussion.

Wow, "Qubes Stateless" is insanely great! Thanks to @xuy for this simple but powerful solution!

I just tested Qubes Stateless out, using a fresh Qubes 4.1. Dom0 does run stateless from RAM. It works!

This provides a way to do up-to-date and useful stateless amnesic Qubes + Whonix now! Better than Tails, like @ adw's cheeky "**TaiQuWhonDo**" in qubes-issues #2024.

Until a true read-only Qubes Live is built by someone, I plan to now use this Qubes Stateless method in real world scenarios going forward. Thank you @xuy!

Here's some further observations, instructions, thoughts:

GRUB Instructions Fix and Boot Process Description:

Step #3 did not work for me, which was:

```
sudo grub2-mkconfig | sudo tee /boot/efi/EFI/grub.cfg
```

When I rebooted, the GRUB boot config had not changed and Qubes OS just did a typical normal boot.

Some researching led me to find that @xuy's instruction was maybe only for UEFI booted systems, where my system boots via Legacy BIOS.

So I fixed this step by changing step #3 into these two commands:

```
sudo grub2-mkconfig -o /boot/grub2/grub.cfg
sudo grub2-mkconfig -o /boot/efi/EFI/grub.cfg
```

I believe this changes the GRUB boot config to simultaneously work for both types of systems (both UEFI and Legacy BIOS).

This fix caused my next reboot to break out of the typical normal Qubes OS boot process, and prompts you to "Press Enter for maintenance", which then drops you into a Dracut command line.

Note that this boot process break happens AFTER entering the Qubes OS disk decryption passphrase. So everything boots normally until after you enter your Qubes OS disk decryption passphrase (if you have full disk encryption enabled).

After decrypting the disk, and you "Press Enter for maintenance", the system drops into a Dracut shell and you proceed by entering the exact commands @xuy provided in Step #4:

```
umount /sysroot
mkdir /mnt
mount /dev/mapper/qubes_dom0-root /mnt
mount -t tmpfs -o size=100% none /sysroot
cp -a /mnt/* /sysroot
```

Note that the last "cp" (copy) step can take a little while (1-4 minutes) to complete, as it is copying all of Dom0 into RAM (I believe).

Pressing Ctrl + D exits you out of the Dracut command line and continues booting into Qubes Stateless Dom0 from RAM.

Once booted up and logged in, I noticed some interesting things…

Lower RAM Optimization:

First, I noticed that the Dom0 RAM partition only takes up ~3.5GB for me initially, which grew to ~5.5GB with some use. Maybe it would grow further, but I didn't seem to need 10GB of RAM allocated to Dom0, so I rebooted into normal Qubes OS mode, and repeated Step #2 & Step #3, and lowered the Dom0 RAM partition in the GRUB boot config to 6GB (dom0_mem=max:6000M). I rebooted into Qubes Stateless mode and it seemed to still work fine with only 6GB allocated to Dom0.

This means that one can explore and choose a higher or lower Dom0 RAM partition size for better RAM efficiency on lower RAM systems. If you only have 4GB of total system RAM, then you could still probably bootup Qubes Stateless. However, the additional RAM you'll then need depends on the number and type of additional qubes and apps you want to run once booted up. Note that not all system RAM

should be allocated to Dom0. RAM used by other qubes is not to be confused with the Dom0 RAM partition size that you set in the GRUB boot config with Step #2.

VM Pool on Disk is still Accessible & Writable:

The follow-on volume instructions provided @xuy show you one way to configure file-backed storage for your VMs, in the “varlibqubes” pool, which is located in the Dom0 directory path /var/lib/qubes.

However, in testing, I didn’t have any secondary storage setup with such pre-existing VM storage loaded. So I instead played with some other options.

You can simply create new qubes and they will exist and work temporarily in RAM, but then you will loose them by default upon shutdown. That may be okay if you don’t want to save anything from your work session.

When creating new qubes in RAM, you can choose from a couple storage pools, which work differently.

If you create a new qube in the “varlibqubes” pool, and you haven’t implemented @xuy’s secondary storage symlinks, then your new qube’s storage will be file-backed in RAM at /var/lib/qubes and will be completely lost upon shutdown by default.

If you create a new qube in the “vm-pool” pool, then your new qube’s storage will be written to your boot drive’s qubes storage pool, and the data will be saved to your boot drive, but the qube in Qube Manager will disappear upon shutdown, due to the metadata (in qubes.xml I believe) being located in the Dom0 RAM partition and being wiped. However, if you try to recreate a qube with the same name using the same storage pool in the future, Qube Manager throws an error but then seems to overwrite the old qube data if you try again, so be careful and don’t assume your data will survive this method unless you really know what you are doing and take advanced precautions.

Using Qubes Stateless and saving your data while using it is best done with adding a secondary storage location as @xuy describes as file-backed, or Qubes documentation describes as pool-backed, or via online cloud storage if that works with your threat model. But, for advanced users, I just wanted to point out that RAM storage and boot drive storage of your qubes data technically do work and these are available for expert data management use cases, or alternatively if one just does not care to save any local data while using Qubes Stateless.

Qubes Stateless Does NOT Seem to Work with a Read-Only Boot Drive:

I tried booting Qubes Stateless on a read-only USB connected drive and some errors did not allow it to work. The boot process of Qubes OS in general seems to demand write access in the initial stages of startup.

First, the LUKS FDE (full drive encryption) seems to demand write access to pass, which can be bypassed by simply not using drive encryption (not ideal).

Second, in @xuy’s Step #4, the following command seems to throw an error when using a read-only boot drive, but seems to work when write access is enabled to the boot drive:

```
mount /dev/mapper/qubes_dom0-root /mnt
THROWS ERROR:
mount: /mnt: can’t read superblock on /dev/mapper/qubes_dom0-root.
```

When booting the drive with write access, I was able to pull out the USB drive once Dom0 booted up to the user login screen, seemingly without encountering any unexpected errors after that.

One time, during some boot testing I was doing, I accidentially borked my Dom0 (i forget how, but it seemed unexpected), and had to reinstall from scratch. So a true read-only Qubes Live boot drive would be much more ideal for protection and security, compared to needing to give write access to the boot drive.

Dual Boot Qubes and Qubes Stateless:

It was implied by @xuy's post, but just wanted to mention this insight for those who may have misunderstood this concept, that:

Qubes Stateless kind of creates an optional "dual boot" mode for Qubes. One, the normal fully stateful mode. The other, the new stateless dom0 mode.

The normal stateful mode becomes the "base" for configuring each live boot of Qubes Stateless. You can reboot into Qubes normal mode, change and reconfigure your Qubes environment, then reboot into Qubes Stateless and have your newly configured Qubes environment running as Qubes Stateless.

This "dual mode" or "dual boot" for Qubes is pretty neat, although somewhat less ideal for security, compared to a true read-only Qubes Live implementation.

Automated Qubes Stateless Booting Considered Desirable:

Automating the inconvenient manual boot Step #4 would be great!

These commands:

```
umount /sysroot
mkdir /mnt
mount /dev/mapper/qubes_dom0-root /mnt
mount -t tmpfs -o size=100% none /sysroot
cp -a /mnt/* /sysroot
```

@ xuy mentions the idea of implementing this automation as a Dracut Module.

@ alzer89 mentions the idea of implementing this automation as a SystemD Service.

However best, it would be awesome to have the code made available for automating this Qubes Stateless boot process, as it gets bothersome when repeating this manually multiple times per day.

Qubes Stateless to Qubes Live:

It would be great to see a truly read-only Qubes Live implementation come out of this innovative Qubes Stateless method somehow!

This Qubes Stateless implementation seems very very close already to a read-only Qubes Live, suitable for everyday use at least from advanced users initially, except for the read-only drive errors during boot and the needed automation of the tmpfs steps.

I'd love to be able to boot from a read-only Qubes Live using read-only media.

@51lieal and All Other Community Members - YES! - as we now seem quite close, let's start a technical discussion about extending this excellent initial work of Qubes Stateless into a truly working read-only Qubes Live! Would this thread, a new forum thread, or a GitHub issue be best?

Can a Qubes moderator please review and approve my previous post that was auto censored by akismet? Thank you!

xuy May 9, 2022, 2:34pm 13

The “Qubes in tmpfs” will be like a green wig feature for LiveUSB. At least till, that becames a mainstram, hah!

Continue, the automation:
This boot scripts will ask you “Boot to RAM? (n)” question at graphical boot splash, just after the “Disk password” screen. You can just press Enter if no.

The boot to RAM automation with a dracut module

See the man dracut.modules

1. Create foder for the boot module

```
cd /usr/lib/dracut/modules.d/
sudo mkdir 01ramboot
```

2. module-setup.sh: the module main script

```
echo '#!/usr/bin/bash

check() {
    return 0
}

depends() {
    return 0
}

install() {
        inst_simple "$moddir/tmpfs.sh" "/usr/bin/tmpfs"
        inst_hook cleanup 00 "$moddir/pass.sh"
}
' | sudo tee 01ramboot/module-setup.sh
```

3. pass.sh: ask question script

```
echo '#!/usr/bin/bash

command -v ask_for_password >/dev/null || . /lib/dracut-crypt-lib.sh

PROMPT="Boot to RAM? (n)"
CMD="/usr/bin/tmpfs"
TRY="3"

ask_for_password \
    --cmd "$CMD" \
    --prompt "$PROMPT" \
    --tries "$TRY" \
    --ply-cmd "$CMD" \
```

```
    --ply-prompt "$PROMPT" \
    --ply-tries "$TRY" \
    --tty-cmd "$CMD" \
    --tty-prompt "$PROMPT" \
    --tty-tries "$TRY" \
    --tty-echo-off
' | sudo tee 01ramboot/pass.sh
```

See the `man plymouth`.

4. tmpfs.sh: mount `tmpfs` script

```
echo '#!/usr/bin/bash

read line

case "${line:-Nn}" in
        [Yy]* )
                mkdir /mnt
                umount /sysroot
                mount /dev/mapper/qubes_dom0-root /mnt
                mount -t tmpfs -o size=100% none /sysroot
                cp -a /mnt/* /sysroot
                exit 0
                ;;
        [Nn]* )
                exit 0
                ;;
        * )
                exit 1
                ;;
esac
' | sudo tee 01ramboot/tmpfs.sh
```

5. Make scritps executable:

`sudo chmod 755 01ramboot/pass.sh 01ramboot/tmpfs.sh`

6. Enable the module

```
echo 'add_dracutmodules+=" ramboot "' | sudo tee /etc/dracut.conf.d/
ramboot.conf
```

See the `man dracut.conf`.

7. Regenerate the latest `/boot/initramfs...` image with `ramboot`

```
sudo dracut --verbose --force

sudo reboot
```

NOTE: This method can be modify with dracut `Hook: cmdline` if you need to have a special kernel boot string in the grub boot menu and not to use the graphical boot splash screen.

0. To remove the scripts

```
sudo rm -Rf /usr/lib/dracut/modules.d/01ramboot
sudo rm -f /etc/dracut.conf.d/ramboot.conf
sudo dracut --verbose --force
```

51lieal May 9, 2022, 3:12pm 14

Skipping the 2 and 3 step, then rebuild new initramfs with this module, then add new grub menu would be good too. So we can have 1 persistent + 1 tmpfs, the persistent aka normal boot would be only use for upgrading dom0, or configuring something that need to be persistent.

I've not tested how it would be good for others, since it need a lot of ram, I'll try to upgrade my laptop to 64gb this week and see if my daily use is fine.

I still searching a good way to boot it as live os, at least we need a kickstart config and livecd-tools.

Have you tried it with less ram or not at all? If so, how much ram and how did it go?

Also, I'm assuming this to be compatible with a detached encrypted boot. Do you have any info?

51lieal May 9, 2022, 4:19pm 16

Yes its compatible, The detached boot is happen in the earlier stage, but my approach is not like in the script above, "it would do same" but not identical.

Actually even with 8gb ram it would run but not much you can do, and for just default install I think 16-32gb would be fine. **Need to test first.**

tripleh May 10, 2022, 7:06am 17

Btw tmpfs can be swapped to disk.

ramfs cannot.

xuy May 10, 2022, 9:18am 18

With ramfs there is no dynamic size limit. The ramfs size is severely cut off from RAM at the ramfs starts. So ramfs is not optimal memory costs. With `tmpfs` an empty filesystem size doesn't take RAM space.

And that is true, tmpfs can be swapped to disk. Therefore **the swap desirable to be disabled**. If you leave the swap enabled, some `tmpfs` (root fs) data may be saved on swap. This data may contain a names of your AppVMs and other `confidential information`.

The simplest solution:

```
sudo swapoff -a
```

No swap, no problem.

_
Or if swap is necessary for a some reason, you need a **separate physical SSD storage for swap**.

- We can encrypt the standard QubesOS swap [LVM Logical] volume `/dev/qubes_dom0/swap`, but that will be a *double encryption*, because the `/dev/qubes_dom0/swap` is already encrypted with the underlying [LVM Physical] `qubes_dom0` volume. That's why we don't do that.

Instead, we'll encrypt the swap SSD with **detached header, so no one knows it's a** `swap SSD` (moreover, it is an encrypted drive at all, just an unformatted disk with `random data`). The swap header file can be removed immediately, as well as the swap drive data is temporary and are not valuable.

1. Prepate a swap drive, let's say `sdX`:

```
DRIVE=/dev/sdX
```

2. Fill the `sdX` with uniform layer of random data:

```
dd if=/dev/urandom of=$DRIVE bs=4096 status=progress
```

3. Encrypt drive with detached header:

Generage keyfile (instead of a password):

```
cd /dev/shm
sudo dd bs=512 count=4 if=/dev/urandom of=swapkey.luks iflag=fullblock
sudo chmod 600 swapkey.luks
```

Format and Open the drive:

```
yes | sudo  cryptsetup luksFormat $DRIVE --key-file swapkey.luks --header
swap-header.luks
sudo cryptsetup luksOpen --header swap-header.luks --key-file
swapkey.luks $DRIVE swap
```

4. Create and mount a new swap:

```
sudo mkswap /dev/mapper/swap
sudo swapon /dev/mapper/swap
```

The `swap-header.luks` is the drive header file.

More detailed, look here:
https://wiki.archlinux.org/title/Dm-crypt/Device_encryption

enmus May 10, 2022, 5:04pm 19

You don't if you can spare a couple more gigs

51lieal May 15, 2022, 11:30pm 20

Have tested root to tmpfs yesterday, this is my notes:

- After rd.break, we need to mount everything manually which I don't like so I did change the base dracut script instead of using rd.break which would boot same as the normal does.
- The difference in performance is not too significant (I install template and run it from varlibqubes), but still good since there's no additional log from now on.

After researching for a few days I got many new thing that I don't know, for the qubes os live project, I've seen joanna post about the disadvantage of using qubes live, and one of the reason is

> 3. We had to solve the problem of Qubes too easily triggering Out Of Memory condition in Dom0 when running as Live OS.
>
> This last problem has been a result of Qubes using the copy-on-write backing for the VM's root filesystems, which is used to implement our cool Template-based scheme [2]. Normally these are backed by regular files on disk – even though these files are discardable upon VM reboots, they must be preserved during the VM's life span, and they can easily grow to a few tens of MBs per VM, sometimes even more. Also, each of the VM's private image, which essentially holds just the user home directory, typically starts with a few tens of MBs for an "empty VM". Now, while these represent rather insignificant numbers on a disk-basked system, in case of a LiveUSB scenarios all these files must be stored in RAM, which is always a scare resource on any OS, and especially Qubes.

I will continue my report in next week about the progress.

enmus November 12, 2022, 12:56am 21

Did you find time to continue this?

51lieal November 12, 2022, 2:16pm 22

haven't try again, maybe in the next time.

xuy December 20, 2022, 8:07am 23

Also, because Qubes manager will not show a in-RAM created VMs after PC reboot.
Therefore, change the default pool parameters to save new VMs **as files** (instead as LVM volumes) will be more user-friendly. This is an example parameters for **opt00** file-pool:

```
qvm-pool add opt00 file -o dir_path=/opt
qubes-prefs --set default_pool opt00
qubes-prefs --set default_pool_private opt00
qubes-prefs --set default_pool_root opt00
qubes-prefs --set default_pool_volatile opt00
```

If you need to remove the wrongly created VMs as LVM Volumes you can use `lvdisplay`, `lvremove` commands. It's probably a way to clone (copy) a VM instead of removal. Anyway, it's best to avoid a compromising names of your virtual machines.

I have tried this and after step 3 and reboot the system boots just as it did before.

On step 1 do I only need to add rd.break at the end of the line and leave the rest of the line as it is?

xuy March 26, 2023, 3:16am 25

Try `cat /proc/cmdline` to find out booted kernel command line parameters.

Try `sudo find /boot -name "*.cfg"` to search the grub config location.

stant, you should edit the line as shown in the example in first post

I have been told 40GB ram is not enough for running qubes in ram.

xuy March 28, 2023, 2:44pm 27

The way the "*Qubes OS in tmpfs*" implies that AppVMs are simply stored on an encrypted disk (SSD). The main trick is the SSD is encrypted with a detached header. Without the detached header file, your disk will be the same as empty disk. The file is stored somewhere else from the SSD.

We can make the task easier and don't boot Qubes in RAM to make traceless/stateless system:

You can just mount the SSD`s filesystem containing your secret AppVMs files above the default file system containing public AppVMs files. So you will get two versions of the same AppVMs: secret and public.

When you've done your secret daily work, you should stop all you secret AppVMs and unmount the disk (SSD filesystem). Then the public (non-secret) AppVMs becomes available on the lower (layer) file system directory. So you're not leaving any trace of your work outside the encrypted SSD. You just need to hide your SSD`s detached header file.

If human right violators will force you to decrypt the system, you will boot to your Qubes OS, show you AppVMs and say that (secret;) SSD doesn't contain any data. You can store your detached header file (16MB size) in a cloud file hosting service or somewhere else.

In general terms you will need to:

- Encrypt you SSD with a detached header [`cryptsetup`]
- Create a pool to store an AppVMs files (choose a storage directory) [`qmv-pool`]
- Mount the encrypted SSD to the AppVMs files storage directory [`mount`]
- Create an AppVMs in this pool [`qvm-create -P`]

All needed terminal commands are available on this thread.
This method doesn't change the OS system requirements. You just need to use **non-compromising names** for your AppVMs. And disable the `swap`, I guess.

I must warn you, there may be a slight difficulty in changing the parameters **of both** (secret and public) machines from Qubes Manager, private storage size, for example.

xuy May 20, 2023, 3:00pm 28

For RAM saving, you can use a compressed RAM-based block device - `zram` kernel module instead of `tmpfs`. There is approximately 45% compression ratio for root directory (`/`) :

Demonstration:

```
user@dom0 ~ % zramctl

NAME       ALGORITHM DISKSIZE DATA COMPR TOTAL STREAMS MOUNTPOINT
/dev/zram0 lzo-rle        10G   7G  3.1G  3.2G       8 /

user@dom0 ~ % zramctl -h
...
DATA  uncompressed size of stored data
```

```
COMPR  compressed size of stored data
...
```

Setup actions:

1. Add + lines, remove - line in our (mentioned earlier) file `/usr/lib/dracut/modules.d/01ramboot/tmpfs.sh` :

```
-               mount -t tmpfs -o size=100% none /sysroot
+               modprobe zram
+               echo 10G > /sys/block/zram0/disksize
+               /mnt/usr/sbin/mkfs.ext2 /dev/zram0
+               mount /dev/zram0 /sysroot
```

2. Add line to our file `/etc/dracut.conf.d/ramboot.conf` :

```
add_drivers+=" zram "
```

3. Regenerate the /boot/initramfs… and reboot.

```
user@dom0 ~ % sudo dracut --verbose --force
```

Hello ! I've tried to use the script of @xuy in the last version of Qubes OS 4.2.0-rc4 and nothing appear. When i start my system i don't have the option "Boot to ram" perhaps it's because i didn't setup a disk password in the installation of qubes ?

Do i need to reinstall Qubes OS 4.2.0-rc4 with disk password encryption for the script running correctly ?

News… Qubes Stateless (RAM-based Qubes in tmpfs/zram, like Tails OS) is working in Qubes 4.2.

Easy and full Step-by-Step instructions provided here in this post to implement a basic Qubes Stateless mode in 4.1 or 4.2. Super credit to @xuy!

Qubes Stateless: Overview...

Qubes Stateless runs your entire Qubes Dom0 environment as Disposable / Amnesic / Ephemeral / Anti-Forensic in RAM, where, unless you purposefully save things in specific alternative ways, your Qubes environment may reset your changes upon restart. This is similar to how Tails OS works, but with the more powerful security & capabilities of Qubes OS.

At each boot, you have the option to run your Dom0 session in Stateless mode or Persistent (normal) mode. Qubes Stateless runs Dom0 from RAM, and you can optionally create and run other user qubes in RAM using the "varlibqubes" storage pool. Your persistent boot storage drive and other attached persistent storage drives will also be available and writable to you during a Stateless session, including any pre-existing user qubes stored on these drives.

Qubes Stateless is an "advanced hybrid state system" where you have both Stateless and Persistent spaces to work with in the same OS environment, and you can optionally control where individual qube volumes operate from (Stateless storage or Persistent storage). This gives us great power and flexibility for controlling the level of statelessness of our qubes.

For those wondering how much hardware system RAM is required to run Qubes Stateless, I'd say 8GB minimum to work and 16GB for basic comfort. Here are my general feelings on amounts of RAM:

- 8GB RAM = Minimal Running
- 16GB RAM = Basic Comfort - A basic Qubes Stateless system can run comfortably on 16GB RAM if number and intensity of AppQubes remains towards lower end.
- 32GB RAM = Extra Comfort - Only needed if your number or intensity of AppQubes demands more.
- 64GB RAM = Luxury Comfort - Only needed if your number or intensity of AppQubes demands more.

Below are the exact step-by-step commands & instructions for a basic RAM-based Qubes Stateless, primarily derived from @xuy's posts…

These steps implement the following aspects & components:

- Basic RAM-based Qubes from xuy's post #1.
- Dracut Automation Module from xuy's post #13.
- Disable Dom0 Swap Permanently in `/etc/fstab`.
- Disable Dom0 Swap Temporarily from xuy's post #18.
- ZRAM Compression from xuy's post #28.
- Unified grub.cfg location for both UEFI and legacy boot (#7985) → R4.2 Release Notes.
- Minor command & code style adjustments for greater newbie ease.

Qubes Stateless: Step-by-Step…

WARNING: Although these Qubes Stateless instructions are made easy enough for almost any Qubes user to follow, Qubes Stateless does involve the modification of your Dom0 system files. Therefore, as a precaution, it is highly recommended that you first perform a full backup of your qubes and data before proceeding, in case your system accidentally gets corrupted or won't boot. No warranty is provided.

In Dom0 of Qubes OS 4.1 or 4.2, open a command line Dom0 Terminal, and perform the following commands & instructions:

+ Step 0. Make a Save Qube:

Optional (Recommended):

```
qvm-create -l blue --prop=netvm= save
```

Further explanation:

To give yourself a convenient option for permanently saving data files while operating in RAM-based Stateless mode, I suggest you create a persistent vault AppQube named "save", so that you can easily copy/move files that you want to save, to the "save" qube, before your computer shuts down, restarts, crashes, etc, and you lose important files while working in the RAM-based Stateless mode.

If you create a "save" qube while in the normal persistent boot mode, then this "save" qube will be available and permanently retain any files sent to it, on your Qubes storage drive, even before, during, and after you are in the RAM-based Stateless mode.

WARNING: If you don't understand what you're doing with an advanced RAM-based stateless configuration of Qubes OS, then it can be very easy to lose data. Ensure that you understand and get familiar with (by testing) the different modes and data saving capabilities of this RAM-based system, and take extra precaution to setup and test reliable methods for saving your important data beyond a stateless session.

+ Step 1. Disable Dom0 Swap:

```
sudo nano /etc/fstab
```

Using nano, type a **#** character in front of the line containing "`/dev/mapper/qubes_dom0-swap`", to comment it out and disable it, like this:

```
#/dev/mapper/qubes_dom0-swap ...
```

Press `Ctrl + O` to save `fstab` file.
Press `Ctrl + X` to exit nano editor.

That will permanently disable your Dom0 Swap after restart, which is likely best for operating Qubes Dom0 in Stateless mode.

If you still want to frequently use Qubes in Persistent mode and have Dom0 Swap enabled, then you can alternatively skip this permanent disable step, and rather temporarily disable Dom0 Swap when you startup Qubes Stateless by remembering (each and every time) to manually run the following Dom0 command…

```
sudo swapoff -a
```

+ Step 2. Edit GRUB to Increase Dom0's Max RAM:

`sudo nano /etc/default/grub`

Within the `GRUB_CMDLINE_XEN_DEFAULT` line, make a partial modification of this line, from `dom0_mem=max:4096M` to now be `dom0_mem=max:10240M` → Just change the text "`4096M`" to "`10240M`" (without quotes).

Press `Ctrl + O` to save grub file.
Press `Ctrl + X` to exit nano editor.

> Note: If you have something like 32GB of RAM or greater in your system, and you want to run several fully stateless user qubes within Dom0 RAM space, then you may want to consider increasing this "10240M" Dom0 RAM maximum value, in order to allow for more Dom0 RAM space where you can store more user qubes as fully stateless. For example, with 32GB of system RAM, you may choose to allocate "20480M" (20GB) or even greater if you wish.

+ Step 3. Regenerate with New GRUB Configuration:

If you are using Qubes 4.2, you should run this single command:

```
sudo grub2-mkconfig -o /boot/grub2/grub.cfg
```

If you are using Qubes 4.1, you should instead run these two commands:

```
sudo grub2-mkconfig -o /boot/grub2/grub.cfg
sudo grub2-mkconfig -o /boot/efi/EFI/grub.cfg
```

+ Step 4. Make New Directory for Dracut Automation Module:

```
sudo mkdir /usr/lib/dracut/modules.d/01ramboot
```

+ Step 5. Make New Dracut Script File module-setup.sh:

```
sudo touch /usr/lib/dracut/modules.d/01ramboot/module-setup.sh
sudo nano /usr/lib/dracut/modules.d/01ramboot/module-setup.sh
```

Edit the newly created `module-setup.sh` file and type-in exactly the following 11 lines of code:

```
#!/usr/bin/bash
check() {
return 0
}
depends() {
return 0
}
install() {
inst_simple "$moddir/tmpfs.sh" "/usr/bin/tmpfs"
inst_hook cleanup 00 "$moddir/pass.sh"
}
```

Press `Ctrl + O` to save `module-setup.sh` file.
Press `Ctrl + X` to exit nano editor.

Perform the following verification steps for your `module-setup.sh` file:

a. Verify that your `module-setup.sh` file has the exact correct contents with a file hash checksum…

```
sudo sha256sum /usr/lib/dracut/modules.d/01ramboot/module-setup.sh
```

…should show output of:

```
cb3e802e9604dc9b681c844d6e8d72a02f2850909ede9feb7587e7f3c2f11b8a    /
usr/lib/dracut/modules.d/01ramboot/module-setup.sh
```

> If this output shows as incorrect, run the previous nano edit commands to correct the file contents and then re-verify this file again. Repeat until correct. Note that only an exact duplication, character by character, of the same file contents from these instructions will produce a successful file hash match, where even extra blank spaces or extra blank lines will cause a changed file hash mismatch. Being very precise counts!

b. Verify that your `module-setup.sh` file is owned by root…

```
sudo ls -l /usr/lib/dracut/modules.d/01ramboot/module-setup.sh
```

…should show output of: “`root root`” for `/usr/lib/dracut/modules.d/01ramboot/module-setup.sh`.

> If this output shows as incorrect, run the following command and then re-verify this file again…
>
> ```
> sudo chown root:root /usr/lib/dracut/modules.d/01ramboot/
> module-setup.sh
> ```

+ Step 6. Make New Dracut Script File pass.sh:

```
sudo touch /usr/lib/dracut/modules.d/01ramboot/pass.sh
sudo chmod 755 /usr/lib/dracut/modules.d/01ramboot/pass.sh
sudo nano /usr/lib/dracut/modules.d/01ramboot/pass.sh
```

Edit the newly created `pass.sh` file and type-in exactly the following 6 lines of code:

```
#!/usr/bin/bash
command -v ask_for_password >/dev/null || . /lib/dracut-crypt-lib.sh
PROMPT="Boot to RAM? (y/n)"
CMD="/usr/bin/tmpfs"
TRY="3"
ask_for_password --cmd "$CMD" --prompt "$PROMPT" --tries "$TRY" --ply-cmd
"$CMD" --ply-prompt "$PROMPT" --ply-tries "$TRY" --tty-cmd "$CMD" --tty-
prompt "$PROMPT" --tty-tries "$TRY" --tty-echo-off
```

Press `Ctrl + O` to save `pass.sh` file.
Press `Ctrl + X` to exit nano editor.

Perform the following verification steps for your `pass.sh` file:

a. Verify that your `pass.sh` file has the exact correct contents with a file hash checksum…

```
sudo sha256sum /usr/lib/dracut/modules.d/01ramboot/pass.sh
```

…should show output of:

```
a2750fa31c216badf58d71abbc5b92097e8be21da23bbae5779d9830e2fdd144    /
usr/lib/dracut/modules.d/01ramboot/pass.sh
```

> If this output shows as incorrect, run the previous nano edit commands to correct the file contents and then re-verify this file again. Repeat until correct. Note that only an exact duplication, character by character, of the same file contents from these instructions will produce a successful file hash match, where even extra blank spaces or extra blank lines will cause a changed file hash mismatch. Being very precise counts!

b. Verify that your `pass.sh` file is owned by root and is executable…

```
sudo ls -l /usr/lib/dracut/modules.d/01ramboot/pass.sh
```

…should show output of: “`-rwxr-xr-x`” and “`root root`” for `/usr/lib/dracut/modules.d/01ramboot/pass.sh`.

> If this output shows as incorrect, run the following commands and then re-verify this file again…
>
> ```
> sudo chown root:root /usr/lib/dracut/modules.d/01ramboot/
> pass.sh
> sudo chmod 755 /usr/lib/dracut/modules.d/01ramboot/pass.sh
> ```

+ Step 7. Make New Dracut Script File tmpfs.sh:

```
sudo touch /usr/lib/dracut/modules.d/01ramboot/tmpfs.sh
sudo chmod 755 /usr/lib/dracut/modules.d/01ramboot/tmpfs.sh
sudo nano /usr/lib/dracut/modules.d/01ramboot/tmpfs.sh
```

Edit the newly created `tmpfs.sh` file and type-in exactly the following 21 lines of code:

```
#!/usr/bin/bash
read line
case "${line:-Nn}" in
```

```
  [Yy]* )
  mkdir /mnt
  umount /sysroot
  mount /dev/mapper/qubes_dom0-root /mnt
  modprobe zram
  echo 10G > /sys/block/zram0/disksize
  /mnt/usr/sbin/mkfs.ext2 /dev/zram0
  mount /dev/zram0 /sysroot
  cp -a /mnt/* /sysroot
  exit 0
  ;;
  [Nn]* )
  exit 0
  ;;
  * )
  exit 1
  ;;
  esac
```

Press `Ctrl + O` to save `tmpfs.sh` file.
Press `Ctrl + X` to exit nano editor.

Perform the following verification steps for your `tmpfs.sh` file:

a. Verify that your `tmpfs.sh` file has the exact correct contents with a file hash checksum…

```
sudo sha256sum /usr/lib/dracut/modules.d/01ramboot/tmpfs.sh
```

…should show output of:

```
d9e85c06c3478cc0cf65a4e017af1a4f9f9dd4ad87c71375e8d4604399f5217d    /
usr/lib/dracut/modules.d/01ramboot/tmpfs.sh
```

> If this output shows as incorrect, run the previous nano edit commands to correct the file contents and then re-verify this file again. Repeat until correct. Note that only an exact duplication, character by character, of the same file contents from these instructions will produce a successful file hash match, where even extra blank spaces or extra blank lines will cause a changed file hash mismatch. Being very precise counts!

b. Verify that your `tmpfs.sh` file is owned by root and is executable…

```
sudo ls -l /usr/lib/dracut/modules.d/01ramboot/tmpfs.sh
```

…should show output of: “`-rwxr-xr-x`” and “`root  root`” for `/usr/lib/dracut/modules.d/01ramboot/tmpfs.sh`.

> If this output shows as incorrect, run the following commands and then re-verify this file again…
>
> ```
> sudo chown root:root /usr/lib/dracut/modules.d/01ramboot/
> tmpfs.sh
> sudo chmod 755 /usr/lib/dracut/modules.d/01ramboot/tmpfs.sh
> ```

+ Step 8. Make New Dracut Config File ramboot.conf:

```
sudo touch /etc/dracut.conf.d/ramboot.conf
sudo nano /etc/dracut.conf.d/ramboot.conf
```

Edit the newly created `ramboot.conf` file and type-in exactly the following 2 lines of code:

```
add_drivers+=" zram "
add_dracutmodules+=" ramboot "
```

Press `Ctrl + O` to save `ramboot.conf` file.
Press `Ctrl + X` to exit nano editor.

Perform the following verification steps for your `ramboot.conf` file:

a. Verify that your `ramboot.conf` file has the exact correct contents with a file hash checksum…

```
sudo sha256sum /etc/dracut.conf.d/ramboot.conf
```

…should show output of:

```
60d69ee8f27f68a5ff66399f63a10900c0ea9854ea2ff7a77c68b2a422df4bef    /
etc/dracut.conf.d/ramboot.conf
```

> If this output shows as incorrect, run the previous nano edit commands to correct the file contents and then re-verify this file again. Repeat until correct. Note that only an exact duplication, character by character, of the same file contents from these instructions will produce a successful file hash match, where even extra blank spaces or extra blank lines will cause a changed file hash mismatch. Being very precise counts!

b. Verify that your `ramboot.conf` file is owned by root…

```
sudo ls -l /etc/dracut.conf.d/ramboot.conf
```

…should show output of: "`root root`" for `/etc/dracut.conf.d/ramboot.conf`.

> If this output shows as incorrect, run the following command and then re-verify this file again…
>
> ```
> sudo chown root:root /etc/dracut.conf.d/ramboot.conf
> ```

+ Step 9. Regenerate with New Dracut Automation Module:

```
sudo dracut --verbose --force
```

> This command may take a couple minutes to complete. When done, the last line will read something like: "`dracut: *** Creating initramfs image file '/boot/initramfs-6.1.56-1.qubes.fc37.x86_64.img' done ***`".

+ Step 10. Shutdown, Restart, Test Qubes Stateless:

Shutdown & Restart your computer. You are ready to test Qubes Stateless!

After normally decrypting your Qubes boot drive, you should be prompted with:

```
Boot to RAM? (y/n):
```

- You can enter "**y**" (without quotes) to boot & use your Qubes Dom0 to Stateless mode in RAM. Your pre-existing persistent TemplateQubes, AppQubes, Data Files, etc, will also be accessible to you while operating in Stateless mode. Note that it can take a few minutes to load Qubes Dom0 into RAM once you enter "**y**".

- You can enter "**n**" (without quotes) to boot & use your Qubes Dom0 normally to Persistent mode from your storage drive. Here you can set and save any permanent state changes you'd like for the next time you boot to Stateless mode. For example, use this normal Persistent mode for: installing Qubes OS updates, installing/configuring TemplateQubes, creating new persistent AppQubes, ServiceQubes, etc, where your next boot into Stateless mode can then take advantage of such persistent changes, improvements, updates, etc.

Notes on Using Qubes Stateless:

Booting into Persistent mode is where you can make any permanent changes/settings and updates to your Dom0/TemplateQubes, etc, which will then be available in your next Stateless mode session.

You'll most likely want to keep your Qubes OS Xen, Dom0, and TemplateQubes up to date by semi-regularly restarting into Persistent mode and installing updates there.

Booting into Stateless mode is where you can operate a stateless Dom0 environment in RAM (which resets its state, upon shutdown, to your last saved Persistent state) and where you also have simultaneous Persistent read/write access to your boot storage drive and secondary storage drives.

When operating in Stateless mode, you can still access and save data to your pre-existing user qubes that were created in Persistent mode or on a secondary storage drive, as well as create and use new user qubes and set their storage volumes (root, private, volatile) to operate fully stateless in Dom0 RAM space using the "`varlibqubes`" storage pool.

You can create new user qubes while in Stateless mode, however, they may or may not persist and be accessible after shutdown (Dom0 metadata is wiped). In the "Create new qube" GUI tool, you can use the "Advanced" tab to specify a "Storage pool". The "`vm-pool`" pool is located on your boot storage drive by default and the "`varlibqubes`" pool is located in your Dom0 filesystem at `/var/lib/qubes` as traditional file-backed volumes (runs in Dom0 RAM space during Stateless mode). However, if you choose "`varlibqubes`", depending on some various conditions and settings, all data volumes of the qube may not be stored in RAM and require extra special configuration for running fully stateless user qubes.

As a precaution to losing important data, assume all of your new changes and new data may be lost by default after your stateless session ends, unless you send the data to a pre-existing persistently created "save" qube (see Step #0) or otherwise really know what you are doing to save your data effectively.

WARNING: If you don't understand what you're doing with an advanced RAM-based stateless configuration of Qubes OS, then it can be very easy to lose data. Ensure that you understand and get familiar with (by testing) the different modes and data saving capabilities of this RAM-based system, and take extra precaution to setup and test reliable methods for saving your important data beyond a Stateless session.

WARNING: For those relying upon Qubes Stateless for anti-forensic use of Qubes OS (which is possible), there are some major caveats to learn about. By default, Qubes Stateless only runs the Dom0 environment as stateless in RAM (take care to disable Dom0 Swap). Other user qubes (such as AppQubes, ServiceQubes, TemplateQubes, etc) might retain some state in their storage volumes if not specifically setup to operate fully stateless in Dom0 RAM space. Other user qubes have multiple storage volumes (root, private, volatile) which can each operate either as Persistent or Stateless depending on various conditions and settings.

Qubes Stateless is an "advanced hybrid state system" where you have both Stateless and Persistent spaces to work with in the same OS environment, and you can optionally control where individual qube volumes operate from (Stateless storage or Persistent storage). The

various storage drives, storage pools, and storage volumes involved can be confusing but very relevant to the degree of statelessness you need or desire for each individual qube.

Some useful commands for managing your statelessness:

- qubes-prefs
- qvm-pool
- qvm-volume
- qvm-prefs
- lsblk
- ls -lsa /var/lib/qubes

Qubes Stateless: Conclusion...

A massive THANK YOU to the ultimate legend @xuy who brought Qubes in tmpfs/zram to life for us!

Everyone feel free to enjoy Qubes Stateless and post any feedback, improvements, tweaks, critiques, issues, ideas, etc.

Enjoy being stateless.

Edit Updates:

#1. 2023-11-11

- Corrected malformed curly/angled-quotes to now be plain straight-quote characters ("), by removing improper HTML code tags that did not work well with Qubes Forum software (Discourse), which was preventing the stateless boot software from running properly if users had copied the malformed quote characters verbatim.
- Added Qubes 4.1 GRUB commands.
- Added user backup warning/recommendation.
- Minor edits.

#2. 2023-11-12

- Added Disable Dom0 Swap step.
- Added file verification sub-steps.
- Minor edits.

#3. 2023-11-12

- Added links to cited resources.
- Minor edits.

#4. 2023-11-12

- Fixed error in tmpfs.sh that caused stateless boot prompt to be skipped.

#5. 2023-11-12

- Corrected number of lines listed for tmpfs.sh file.

#6. 2023-12-08

- Added permanent disable of Dom0 Swap.

#7. 2023-12-08

- Improved several explanations throughout post.

- Minor edits.

Thanks for the update ! But when i try to run “sudo dracut --verbose --force” i have the following error “dracut Cannot install a hook (”/usr/lib/dracut/modules.d/01ramboot/pass.sh") that does not exist.

pass.sh is installed inside the 01ramboot folder. I tried to delete every files and folder and restart again but i have the same error 3 times…

Hi @qubesuser1234

Next time you get this error, before deleting or altering the files, can you then run and post the output of the following two commands, so we can potentially help you fix…

```
sudo ls -lsa /usr/lib/dracut/modules.d/01ramboot
```

```
sudo sha1sum /usr/lib/dracut/modules.d/01ramboot/*
```

Thanks for your reply ! So with your first command `sudo ls -lsa /usr/lib/dracut/modules.d/01ramboot` I got this :

total 20
4 drwxr-xr-x 2 root root 4096 Nov 7 23:02 .
4 drwxr-xr-x 64 root root 4096 Nov 7 23:01 …
4 -rwxr-xr-x 1 yukz yukz 172 Nov 7 22:59 module-setup.sh
4 -rwxr-xr-x 1 yukz yukz 386 Nov 7 22:59 pass.sh
4 -rwxr-xr-x 1 yukz yukz 292 Nov 7 22:59 tmpfs.sh

and with your last command i got this : a7eb5499b7d45c33fa49a268947c4fc9093e5e55 /usr/lib/dracut/modules.d/01ramboot/module-setup.sh
bef4f0747b4a9b43886e48bd7466018e53f1204b /usr/lib/dracut/modules.d/01ramboot/pass.sh
900621278b582f42dedf6354b3530ebef8d8dbb7 /usr/lib/dracut/modules.d/01ramboot/tmpfs.sh

I forget to say i’ve reinstalled Qubes 4.1.2 i’m not using anymore 4.2-rc4. Do you think the issue is this ?

@qubesuser1234

The only thing I believe that’s changed here between Qubes R4.1 and R4.2 is in Step #3…

If still using Qubes 4.1.X, you’ll also want to change Step #3 to this:

```
sudo grub2-mkconfig -o /boot/grub2/grub.cfg
sudo grub2-mkconfig -o /boot/efi/EFI/grub.cfg
```

However, I do not think that issue caused your specific error that you described when running Step #9 “`sudo dracut --verbose --force`”.

First thing I notice is that your script files in the 01ramboot directory are owned by a user account named “yukz” instead of “root”. This is probably caused by not typing the commands exactly and not placing “sudo” in the commands for creating and editing these script files.

You could correct this by either deleting and re-doing the steps exactly with “sudo”, or just run the following command to change ownership to root for the existing script files you’ve already created:

```
sudo chown root:root /usr/lib/dracut/modules.d/01ramboot/*
```

Re-running `sudo ls -lsa /usr/lib/dracut/modules.d/01ramboot` should then show all the script files as owned by “root root” instead of “yukz yukz”.

Another issue I notice is that your sha1sum file hashes are different than mine for all 3 .sh script files. This means that the exact contents put into the script files is different than the instructions precisely

demonstrate. Even less or more blank spaces or extra blank lines can change the hashes, so it may or may not have an impact in causing the error, depending upon what exact differences you put into the script files, even if just minor typo mistakes.

You could double check the exact character by character contents of the script files, since your posted hashes mean that something is different in each of the 3 .sh script files from my exact posted instructions.

Then you can try re-running Step #9 again to see if the error is resolved.

```
sudo dracut --verbose --force
```

If errors still persist, then re-check those script files until there is absolutely no mistakes or differences from the posted instructions.

I will probably post file hashes soon to help further aide users in confirming that they typed in the exactly correct file contents that matches the instructions character for character.

- Note to self: We need an easy way to copy these texts or files into Dom0, without precise script file typing needed by users. Anyone know of a good way to do this?

Interested to hear if you get Qubes Stateless working and how you enjoy it!

Depending on what your circumstances are, packaging them as an RPM package may have merits.

Quoting myself (source):

> […] copying files from less-trusted to more-trusted qubes should be avoided. Because *dom0* is the most trusted qube and the most critical qube, copying files to *dom0* from a work qube on a regular basis doesn't seem reasonable to me (your circumstances may be different!)
>
> There are, however, mechanisms by which *dom0* can be updated. In particular, the dom0 secure updates mechanism provides better security than copying files between qubes. In order to take advantage of the *secure updates* mechanism, we need to package our [files] as RPM packages, and use the security features of the RPM workflow to allow *dom0* to verify them. This guide explains how to do that.

My personal approach is to make it easier for folks to package themselves the files I provide, so they only have to trust themselves, but you could as well distribute RPM packages if that's your preference.

That writing and code originated from the following topic on this forum (centered on copying *Salt* files, but applies to any kind of file):

Thanks for help ! i've listen what you said and it's working running "sudo dracut --verbose --force" give me no error now. I wrote the script myself instead of copy and paste !

But… there is a little issue the script don't work my system is booting without telling me "Boot to ram ?"

I don't know if it's normal or not but when i edit grub to increase dom0_mem=max:10240M for me i have to do "sudo grub2-mkconfig -o /boot/efi/EFI/qubes/grub.cfg" instead of `sudo grub2-mkconfig -o /boot/grub2/grub.cfg sudo grub2-mkconfig -o /boot/efi/EFI/grub.cfg`

Perhaps i have to delete the folder "/boot/efi/EFI/qubes/grub.cfg" ? I'm wondering if this a good idea…

Thanks @gonzalo-bulnes - Yes, packaging as RPM would likely be the ultimate best, if I get time to learn in the future. Thanks for the resources on that!

Using this command seems to be the right quick and dirty method (source):

```
qvm-run -p QUBE_NAME 'cat NAME_OF_FILE ' > NAME_IN_DOM0
```

@qubesuser1234 - I also have a 4.1.X installation, and the grub.cfg is located specifically at “/boot/efi/EFI/grub.cfg”. You could try running the command for all three locations. But I don’t think this step should fundamentally prevent the stateless system from booting either, rather, should just boot with lower maximum Dom0 RAM available.

```
sudo grub2-mkconfig -o /boot/grub2/grub.cfg
sudo grub2-mkconfig -o /boot/efi/EFI/qubes/grub.cfg
sudo grub2-mkconfig -o /boot/efi/EFI/grub.cfg
```

It looks like the Qubes Forum software maybe changed the quote characters in my posted script files (`module-setup.sh`, `pass.sh`, `tmpfs.sh`) from straight-quotes (") to curly/angled-quotes (”), because I used a different code tag than I probably should have. These malformed quote characters may have been some or all of the source of confusion and non-working file corruption.

All quotes typed into script & config files should strictly be plain straight quote characters (").

I will plan to soon repost with corrected formatting of quote characters.

In the meantime, here are the believed correct file hashes I have:

sha1sum

```
105190ac916968e49f85911fd3ca2e29909ae327  /usr/lib/dracut/
modules.d/01ramboot/module-setup.sh
8ea26fb215217c67c6806b40d760b57f24025047  /usr/lib/dracut/
modules.d/01ramboot/pass.sh
f9155e309af3a0a776b3a5b0f8ec2951f1178ff1  /usr/lib/dracut/
modules.d/01ramboot/tmpfs.sh
1ed767101974b65db6fb30346e96030a78d89567  /etc/dracut.conf.d/ramboot.conf
```

sha256sum

```
cb3e802e9604dc9b681c844d6e8d72a02f2850909ede9feb7587e7f3c2f11b8a  /usr/
lib/dracut/modules.d/01ramboot/module-setup.sh
a2750fa31c216badf58d71abbc5b92097e8be21da23bbae5779d9830e2fdd144  /usr/
lib/dracut/modules.d/01ramboot/pass.sh
37cafbead5124b57a98bb9427e078cecf7d3c04003267d9fc16e34188aaffe93  /usr/
lib/dracut/modules.d/01ramboot/tmpfs.sh
60d69ee8f27f68a5ff66399f63a10900c0ea9854ea2ff7a77c68b2a422df4bef  /etc/
dracut.conf.d/ramboot.conf
```

sha512sum

```
8a1551a0d9fdb6fe543ef302a89f085de6c3e4dfb1648795fa73f6096277e5d9e2e108fcb
4756549f4ab1544c6950a86c76f05701156b229325e592447972bae  /usr/lib/dracut/
modules.d/01ramboot/module-setup.sh
3a890a4055bd8cad1b790ad2092c3f481ed512d7082d491951f2b474f519124c086cc38b8
a3055914938cab223e496a1c101cc79d415cb5cf57010d6e6ef2fb6  /usr/lib/dracut/
modules.d/01ramboot/pass.sh
```

```
401f4f3be222b07c4d36705410e5525c62ec6c025f86eae1eca5dc7538739eff0c845a9aa
978dbb2b2985ee9c885911fbbbc721d3f7128314dac8f0e53eef0bb  /usr/lib/dracut/
modules.d/01ramboot/tmpfs.sh
86d4ff45c86c4cee0ac5e92f8ed52183fa2e8445154f963bfe3cd91ced390a488a926e362
7c55b9c6fd598e8105088638c0b90a838a403c4acaefe6519758a05  /etc/
dracut.conf.d/ramboot.conf
```

This seems indeed to be the case. When I edit your post, I see some HTML tags (e.g. `<code>`, `<br>`). Did you input these by hand?

If you did, that was a good guess, but clearly the Discourse editor is not quite happy with them, the code is formatted as code, but the usual replacement characters for non-code were applied. (For example, the curly quotes.)

Try using *fenced code blocks* instead. You can open and close them with three backticks (```) on their own line before and after the block. As a bonus, the newlines will be reapected, so no need for `<br>` or any other HTML.

Got it. Yup, you're right. For some reason, Discourse editing/formatting buttons aren't visually showing up in my Tor Browser, so it was partial guessing.

Thanks for the helpful tips @gonzalo-bulnes!

Apparently I don't seem to have an edit capability on past posts , so I may have to re-post new step-by-step instructions without fixing the previously posted code that contain error producing curly-quotes in it.

That's surprising. Any thoughts @deeplow ?

deeplow November 10, 2023, 7:20am 41

As peoppe use the forum they gain more capabilities. @qstateless was pretty much in Trust Level 2 (missing only giving one like). I've upgraded you know.

So you should now be able to edit posts up to 30 days I hope that helps

@qstateless I think i have to delete the folder and files inside efi/qubes and then try to run "sudo grub2-mkconfig -o /boot/grub2/grub.cfg" and "sudo grub2-mkconfig -o /boot/efi/EFI/grub.cfg"

But i'm wondering if it will break my system ? Can someone tell me what will happen if i try to do this ?

Thanks @deeplow and @gonzalo-bulnes! It looks like I can edit that past post now. That helps!

I will plan on updating the step-by-step instructions post with proper straight-quote (") characters, as well as some file hash verification steps.

@qubesuser1234 - DO NOT delete your EFI or EFI/qubes folders. There are other Xen files in there, and your system might not boot if you did that. No to deleting folders here.

The matter just comes down to what path your EFI `grub.cfg` file should be located at. As mentiond…

My EFI `grub.cfg` in Qubes 4.1.X is specifically located at:

`/boot/efi/EFI/grub.cfg`

Buy you said that your EFI `grub.cfg` in your Qubes 4.1.X is specifically located at:

`/boot/efi/EFI/qubes/grub.cfg`

I wonder if you also have another grub.cfg at `/boot/efi/EFI/grub.cfg` ?

To check for these grub.cfg files, run the following commands and see if any `grub.cfg` files are present in one or both of these locations:

```
sudo ls /boot/efi/EFI
sudo ls /boot/efi/EFI/qubes
```

I believe the `grub2-mkconfig` command generates the `grub.cfg` file and places it at whatever path one specifies in the `-o` parameter.

I would not think that having an extra `grub.cfg` file within both of the `/boot/efi/EFI` and `/boot/efi/EFI/qubes` directories would interfere with the boot, as the system would probably just only use one of them.

After updating your `/etc/default/grub` file in Step #2, you could probably generate a `grub.cfg` in all three locations, like this:

```
sudo grub2-mkconfig -o /boot/grub2/grub.cfg
sudo grub2-mkconfig -o /boot/efi/EFI/qubes/grub.cfg
sudo grub2-mkconfig -o /boot/efi/EFI/grub.cfg
```

In my step-by-step instructions, the only thing that's changed in GRUB is the Max Dom0 memory, which should not prevent stateless booting itself.

Since Dom0 system files are being altered, it is highly advisable that one first performs a full backup all of their qubes and data, in case the worst case scenario happens and your system won't boot.

Running sudo ls command give me this :

> yukz@dom0 ~> sudo ls /boot/efi/EFI
> BOOT fedora grub.cfg qubes
> yukz@dom0 ~> sudo ls /boot/efi/EFI/qubes
> fonts grub.cfg grubx64.efi initramfs-6.1.12-1.qubes.fc32.x86_64.img xen-4.14.6.efi xen.efi

Yes but why my grub.cfg is inside a qubes folder by default ? I do not understand why it's not like you but like i said i think this is the problem your script can't work properly on my system because of this

FYI: I just updated the step-by-step instructions to correct the malformed quote characters. Will plan to still add some additional verification steps in another update.

So you do have 2 separate `grub.cfg` files, one in both of those different directories.

Not sure why. Are you sure it was in there by default? Could you have possibly accidentally mistyped a command or file path and caused an extra one to be put into the qubes sub-directory?

xuy's original instructions from last year, during the Qubes 4.1.X era, used the same path that my own system's EFI `grub.cfg` file is in (`/boot/efi/EFI/grub.cfg`).

A clean install of 4.1.X could tell you what the default location of the `grub.cfg` files are for certain, if you are determined to find that out.

In Qubes 4.2, the multiple `grub.cfg` files situation for EFI and legacy boot was unified into one single `grub.cfg` file location, so it is a bit simpler in Qubes 4.2.

I don't think this would be the case, as explained in my previous replies.

Have you re-checked the hashes of your script files since I posted the hashes?

Run:

```
sudo sha1sum /usr/lib/dracut/modules.d/01ramboot/*
sudo sha1sum /etc/dracut.conf.d/ramboot.conf
```

Should match these hashes:

> yukz@dom0 ~> sudo sha1sum /usr/lib/dracut/modules.d/01ramboot/*
> 105190ac916968e49f85911fd3ca2e29909ae327 /usr/lib/dracut/modules.d/01ramboot/module-setup.sh
> 8ea26fb215217c67c6806b40d760b57f24025047 /usr/lib/dracut/modules.d/01ramboot/pass.sh
> f9155e309af3a0a776b3a5b0f8ec2951f1178ff1 /usr/lib/dracut/modules.d/01ramboot/tmpfs.sh
> yukz@dom0 ~> sudo sha1sum /etc/dracut.conf.d/ramboot.conf
> 1ed767101974b65db6fb30346e96030a78d89567 /etc/dracut.conf.d/ramboot.conf

As you can see we have the same hashes but it's doesn't work i will try to reinstall Qubes 4.1.2 and see if it's working.

Just for be sure i will reinstall the system and give you some feedback i will see if it was there by default

I just updated the Qubes Stateless step-by-step instructions. Enjoy!

This update includes the addition of a Disable Dom0 Swap step and file verification sub-steps.

Edit Updates thus far:

#1. 2023-11-11

- Corrected malformed curly/angled-quotes to now be plain straight-quote characters ("), by removing improper HTML code tags that did not work well with Qubes Forum software (Discourse), which was preventing the stateless boot software from running properly if users had copied the malformed quote characters verbatim.
- Added Qubes 4.1 GRUB commands.
- Added user backup warning/recommendation.
- Minor Edits

#2. 2023-11-12

- Added Disable Dom0 Swap step
- Added file verification sub-steps
- Minor Edits

Bob3 November 12, 2023, 9:47am 48

Thanks a lot @qstateless and @xuy . You came up with a big breakthrough for Qubes IMHO. I am a big noob and I am concerned that I might mess up or misunderstand something, so as a good noob I went to a language model and asked to make a script based on the instructions of @qstateless .

```
#!/bin/bash

# Check if the script is running as root
if [ "$(id -u)" != "0" ]; then
   echo "This script must be run as root" 1>&2
   exit 1
fi
```

```
# Step 0: Create a Save Qube (optional)
echo "Creating a Save Qube..."
qvm-create -l blue --prop=netvm= save

# Step 1: Disable Dom0 Swap
echo "Disabling Dom0 Swap..."
sudo swapoff -a

# Step 2: Edit GRUB to Increase Dom0's Max RAM
echo "Editing GRUB Configuration..."
sed -i 's/dom0_mem=max:4096M/dom0_mem=max:10240M/' /etc/default/grub

# Step 3: Regenerate with New GRUB Configuration
echo "Updating GRUB Configuration..."
if [ -d /boot/efi/EFI/grub.cfg ]; then
    sudo grub2-mkconfig -o /boot/grub2/grub.cfg
    sudo grub2-mkconfig -o /boot/efi/EFI/grub.cfg
else
    sudo grub2-mkconfig -o /boot/grub2/grub.cfg
fi

# Step 4: Make New Directory for Dracut Automation Module
echo "Creating Dracut Automation Module directory..."
sudo mkdir -p /usr/lib/dracut/modules.d/01ramboot

# Step 5: Create Dracut Script File module-setup.sh
echo "Creating module-setup.sh..."
cat <<EOF > /usr/lib/dracut/modules.d/01ramboot/module-setup.sh
#!/usr/bin/bash
check() {
    return 0
}
depends() {
    return 0
}
install() {
    inst_simple "\$moddir/tmpfs.sh" "/usr/bin/tmpfs"
    inst_hook cleanup 00 "\$moddir/pass.sh"
}
EOF
sudo chmod 755 /usr/lib/dracut/modules.d/01ramboot/module-setup.sh

# Step 6: Create Dracut Script File pass.sh
echo "Creating pass.sh..."
cat <<EOF > /usr/lib/dracut/modules.d/01ramboot/pass.sh
#!/usr/bin/bash
command -v ask_for_password >/dev/null || . /lib/dracut-crypt-lib.sh
PROMPT="Boot to RAM? (y/n)"
CMD="/usr/bin/tmpfs"
TRY="3"
ask_for_password --cmd "\$CMD" --prompt "\$PROMPT" --tries "\$TRY" --ply-
cmd "\$CMD" --ply-prompt "\$PROMPT" --ply-tries "\$TRY" --tty-cmd "\$CMD"
--tty-prompt "\$PROMPT" --tty-tries "\$TRY" --tty-echo-off
EOF
sudo chmod 755 /usr/lib/dracut/modules.d/01ramboot/pass.sh
```

```
# Step 7: Create Dracut Script File tmpfs.sh
echo "Creating tmpfs.sh..."
cat <<EOF > /usr/lib/dracut/modules.d/01ramboot/tmpfs.sh
#!/usr/bin/bash
read line
case "\${line:-Nn}" in
[Yy]* )
    mkdir /mnt
    umount /sysroot
    mount /dev/mapper/qubes_dom0-root /mnt
    modprobe zram
    echo 10G > /sys/block/zram0/disksize
    /mnt/usr/sbin/mkfs.ext2 /dev/zram0
    mount /dev/zram0 /sysroot
    cp -a /mnt/* /sysroot
    exit 0
    ;;
[Nn]* )
    exit 0
    ;;
esac
EOF
sudo chmod 755 /usr/lib/dracut/modules.d/01ramboot/tmpfs.sh

# Step 8: Make New Dracut Config File ramboot.conf
echo "Creating ramboot.conf..."
cat <<EOF > /etc/dracut.conf.d/ramboot.conf
add_drivers+=" zram "
add_dracutmodules+=" ramboot "
EOF

# Step 9: Regenerate with New Dracut Automation Module
echo "Regenerating Dracut..."
sudo dracut --verbose --force

echo "Setup completed. Please manually shut down and restart your system
to test Qubes Stateless."
```

Is this any good? Would it work just moving it to dom0 and executing it as root?

Bob3 November 12, 2023, 10:52am 49

I have tried to follow all of the instructions as well, I have checked that all of the hashes match and they do. But it is not working for me .

The system boots again but without asking if I want to boot in RAM or not.

Major Update: I just updated the Qubes Stateless step-by-step instructions.

CODE ERROR DETECTED AND CORRECTED!

I found the slight code error that is the culprit preventing your Qubes Stateless from running on boot.

There indeed was a code error preventing a successful run within the `tmpfs.sh` file.

The case logic in my posted `tmpfs.sh` file ended like this:

```
[Nn]* )
exit 0
;;
esac
```

However, it should have ended like this:

```
[Nn]* )
exit 0
;;
* )
exit 1
;;
esac
```

It was missing this slight portion of code near the end:

```
* )
exit 1
;;
```

This was causing the stateless “Boot to RAM? (y/n)” prompt not to show and be automatically skipped.

Somehow, between my development computer and my test computer, a code regression occurred, where my test computer received the good working code but my development computer was showing the bad non-working code making me misbelieve it was good.

Error is fixed now. Should be fully working. My apologies for the frustration.

@qubesuser1234 and @Bob3 and everyone else… Please feel free to retry the Qubes Stateless step-by-step instructions again and confirm that Qubes Stateless is now successfully working for you.

Also, here are the new believed correct file hashes I have (which I matched in the updated instructions):

sha1sum

```
105190ac916968e49f85911fd3ca2e29909ae327  /usr/lib/dracut/
modules.d/01ramboot/module-setup.sh
8ea26fb215217c67c6806b40d760b57f24025047  /usr/lib/dracut/
modules.d/01ramboot/pass.sh
9c8309dc808204d4eccee0da55d592a6cc320834  /usr/lib/dracut/
modules.d/01ramboot/tmpfs.sh
1ed767101974b65db6fb30346e96030a78d89567  /etc/dracut.conf.d/ramboot.conf
```

sha256sum

```
cb3e802e9604dc9b681c844d6e8d72a02f2850909ede9feb7587e7f3c2f11b8a  /usr/
lib/dracut/modules.d/01ramboot/module-setup.sh
a2750fa31c216badf58d71abbc5b92097e8be21da23bbae5779d9830e2fdd144  /usr/
lib/dracut/modules.d/01ramboot/pass.sh
```

```
d9e85c06c3478cc0cf65a4e017af1a4f9f9dd4ad87c71375e8d4604399f5217d  /usr/
lib/dracut/modules.d/01ramboot/tmpfs.sh
60d69ee8f27f68a5ff66399f63a10900c0ea9854ea2ff7a77c68b2a422df4bef  /etc/
dracut.conf.d/ramboot.conf
```

sha512sum

```
8a1551a0d9fdb6fe543ef302a89f085de6c3e4dfb1648795fa73f6096277e5d9e2e108fcb
4756549f4ab1544c6950a86c76f05701156b229325e592447972bae  /usr/lib/dracut/
modules.d/01ramboot/module-setup.sh
3a890a4055bd8cad1b790ad2092c3f481ed512d7082d491951f2b474f519124c086cc38b8
a3055914938cab223e496a1c101cc79d415cb5cf57010d6e6ef2fb6  /usr/lib/dracut/
modules.d/01ramboot/pass.sh
e95af846c48d7cf6e0fd659af5296aa532cbd1fb1d337d05c2a69251429e17b47471d4476
ad31fe93f96b17703bf38e1eb20306a3875aef973592103c40f20e8  /usr/lib/dracut/
modules.d/01ramboot/tmpfs.sh
86d4ff45c86c4cee0ac5e92f8ed52183fa2e8445154f963bfe3cd91ced390a488a926e362
7c55b9c6fd598e8105088638c0b90a838a403c4acaefe6519758a05  /etc/
dracut.conf.d/ramboot.conf
```

Yeah, big thanks to @xuy for researching and delivering the innovation to us. I've been using it since last year and it is a game changer. I hope my step-by-step instructions can now make it accessible for more people in the Qubes community.

Couldn't be sure without testing. A lot of it looks ok at a glance, but not sure.

I may be thinking of developing an automated Qubes Stateless installer script in the upcoming future to make this system even more easy.

Yes! My fault. Error found and corrected. See my prior post for explanation.

Qubes Stateless step-by-step instructions have been updated and should be fully working now.

Enjoy being stateless!

Bob3 November 12, 2023, 1:12pm 52

I am really eager to complete this and build my system around it so I already made the changes you suggested.

I get asked if I want to boot to RAM after inserting the decryption pass, but I tried to open a new tab in firefox on the personal domain and after restarting and booting selecting **n** I find the tab where it was . So I either did not understand how this works or something went wrong and the system is not running in RAM.

@Bob3 - Good to hear that the "Boot to RAM? (y/n)" prompt is showing up for you now.

Your stateless system is likely working now and there's probably just a slight misunderstanding on how the system works…

If your `personal` qube was created in the Normal Persistent mode, then it will save anything written to it on your storage drive, even while using that `personal` qube during Stateless mode.

With the Qubes Stateless implementation, it is specifically the Dom0 environment that runs in RAM.

Your storage drive will still be accessible and writable, along with any pre-existing TemplateQubes, AppQubes, data files, etc that were created in the Normal Persistent mode.

Any AppQubes you create purely while in Stateless mode using the "varlibqubes" storage pool will only exist in RAM and be fully gone and wiped after restart, along with their data fully gone and wiped, unless you otherwise persistently save such data somewhere else (like a persistent "save" qube) while in a stateless session.

If you don't want any changes saved within an AppQube, then you likely need to consider creating such qubes while in Stateless mode using the "varlibqubes" storage pool, which places them within Dom0 while running in RAM, which is fully wiped upon restart. Note the RAM-based storage space for "varlibqubes" is only limited to your free Dom0 memory.

The current implementation of Qubes Stateless is a "hybrid" environment, where Dom0 is running in RAM but what pre-exists on your storage drive is still accessible and writable (including any persistently created qubes). This is different than a full blown Qubes Live system, where everything is non-persistent in RAM by default for Qubes Live. There are some tradeoffs to each type of system, Qubes Stateless vs Qubes Live, but with some strategic configuration, you can likely make Qubes Stateless act how you want it to.

More info on these topics has been previously explored somewhat throughout this thread.

Feel free to ask further questions and provide further thoughts.

Bob3 November 12, 2023, 2:21pm 54

Thanks a lot for replying so quickly.

As far as I understood then I can create a "stateless qube" using "varlibqubes" storage pool while using the non stateless mode, set it up as I wish and them use the qube for daily work, everything that was done in that qube will be deleted at reboot every time I boot into "stateless mode".

Did I understand correctly?

@Bob3 - Cheers… I believe you are correct.

I haven't personally made use of that approach, but I think it should work.

My understanding…

- 1. Create work qube using "varlibqubes" storage pool in Normal Persistent mode. This work qube will persistently exist as a file-backed qube in Dom0 within /var/lib/qubes.
- 2. Boot into Stateless mode. The work qube and its persistent files/configurations will load just as they were in your last Persistent mode session, but in RAM during your Stateless mode session.
- 3. Shutdown Stateless mode. Any and all changes in the work qube will be fully wiped and it will entireley revert back to its previous state from before your Stateless mode session began.

The "varlibqubes" storage pool is a file-backed (stored as traditional files) in Dom0 within the /var/lib/qubes directory. Since Dom0 loads into RAM during Stateless mode, all your file-backed qubes that use "varlibqubes" should not be able to persist any state from their operations in Statless mode.

A small caveat: If that work AppQube is based on a persistent TemplateQube and that persistent TemplateQube is updated or changed while in Stateless mode, then those TemplateQube changes will persist for all qubes that use this template, including the "stateless" work qube. However, template updates/changes usually are only for application packages, where your "Home" folder's data files and app configurations should not be affected in the work qube itself. You can go deeper and customize just about anything to work non-typical in Qubes OS, but typical use of TemplateQubes should not likely interfere with your "stateless" work qube plans.

Different combinations of when, how, where you create & use various types of qubes & data can have

different state outcomes. So, yeah, Qubes Stateless is an “advanced hybrid state system” like that.

Testing and confirming your expectations is always the key!

Hope that is clear and helps.

To those interested in fully stateless (anti-forensic) use of Qubes Stateless, this is an important update.

@Bob3 et al…

I did some further digging into and testing of Qubes Stateless operations.

It seems there are some lingering issues & caveats with achieving fully stateless (anti-forensic) use of Qubes Stateless.

1. Dom0 Still Swaps to Storage Drive

More info here on Wikipedia for those not familiar with what swap is.

It appears that Dom0 Swap is still enabled with each system restart, and that @xuy’s recommended command `sudo swapoff -a` in post #18 for turning off Dom0 Swap is only temporary for any given Persistent mode session or Stateless mode session, but doesn’t last beyond restart (or maybe even a logoff?).

We need to come up with a persistent way to disable Dom0 Swap. I haven’t researched this yet.

In the meantime, you should run the temporary command in the Dom0 Terminal at the beginning of every new Qubes Stateless session…

```
sudo swapoff -a
```

2. AppQubes Typically Created in ‘varlibqubes’ Still Partially Using Storage Drive

While testing in Qubes 4.2-RC4, I discovered that creating an AppQube that is based on a TemplateQube and setting the storage pool to ‘`varlibqubes`’ stores everything in Dom0’s `/var/lib/qubes` directory, EXCEPT for one thing I’ve found…

Running in Dom0 Terminal:

```
lsblk

lsblk | grep YOURQUBENAME
```

…reveals that your AppQube is still storing its ‘root’ volumes on your storage drive.

Here, you can clearly see volumes named ‘`qubes_dom0-vm--YOURQUBENAME--root--snap`’.

These ‘root’ volume can write sensitive data during your Qubes Stateless session to your persistent storage drive, which could potentially be recovered from the storage drive.

For a fully stateless workaround with AppQubes, see the approaches in “*4. Workarounds for Fully Stateless AppQubes & DisposableQubes*”.

3. DisposableQubes in Stateless Mode are Typically Not Fully Stateless

You should be aware that typical default configurations of DisposableQubes while in Stateless mode are not fully stateless and store their data on your storage drive.

For a fully stateless workaround with DisposableQubes, see specific approah “c” in “*4. Workarounds for Fully Stateless AppQubes & DisposableQubes*”.

4. Workarounds for Fully Stateless AppQubes & DisposableQubes

Thankfully, there seems to be a fully stateless workaround for the issues I described in “*2. AppQubes Typically Created in ‘varlibqubes’ Still Partially Using Storage Drive*” and “*3. DisposableQubes in Stateless Mode are Typically Not Fully Stateless*”, although quite costly in some RAM space…

You can take a few different approaches to resolve this fully stateless issue now:

- a. Instead of using AppQubes based on TemplateQubes, you could alternatively create StandaloneQubes in the ‘`varlibqubes`’ storage pool, which appear to store ALL data in traditional image files within Dom0’s ‘`/var/lib/qubes`’ directory. This is very costly in Dom0 RAM, as it copies your entire TemplateQube’s OS into Dom0 RAM space for each StandaloneQube you make, whether it is actively running or not, which is usually multiple extra GBs per qube, in addition to the RAM it takes to store any user files and the RAM it takes to run and operate the qube’s OS & apps.
- b. You could create a new TemplateQube from the previous TemplateQube you want to use, but store that new TemplateQube in the ‘`varlibqubes`’ storage pool. Then create a new AppQubes based on this new TemplateQube, and store this new AppQube in the ‘`varlibqubes`’ storage pool too. Now, when you use this new AppQube, it appears to store ALL data in traditional image files within Dom0’s ‘`/var/lib/qubes`’ directory. This is as costly as the other method for the first AppQube, but you do not have to copy & store the entire TemplateQube OS root filesystem for every AppQube you want to make with it, so this saves a lot of RAM space for using more than one qube.
- c. Like “b”, you could create a new TemplateQube from the previous TemplateQube you want to use, but store that new TemplateQube in the ‘`varlibqubes`’ storage pool. Then create a Disposable Template by creating new AppQubes based on this new TemplateQube, and store this new AppQube in the ‘`varlibqubes`’ storage pool too. After creation, in the settings of this AppQube, under the “Advanced” tab, you can check to turn on “Disposable template” and after applying also select “Default disposable template” to either be “(none)” or that very same AppQube itself. Now, you can use both this TemplateQube and Disposable Template AppQube to create new AppQubes and DisposableQubes fully within the ‘`varlibqubes`’ storage pool. This is likely to generally be the most desirable approach for most people.

Here is an example implementation of approach “c”:

Let’s say you want to base some of your fully stateless qubes on the ‘`debian-12-xfce`’ persistent template.

- 1. In Persistent mode: Create & Configure a new TemplateQube named ‘`debian-12-xfce-stateless`’ based on ‘`debian-12-xfce`’ and choose to store it in storage pool ‘`varlibqubes`’ (Advanced tab).
- 2. In Persistent mode: Create & Configure a new AppQube named ‘`debian-12-xfce-stateless-dvm`’ based on ‘`debian-12-xfce-stateless`’ and choose to store it in storage pool ‘`varlibqubes`’ (Advanced tab).
- 3. In Persistent mode: After creation, for the AppQube ‘`debian-12-xfce-stateless-dvm`’, change the ‘Advanced’ tab setting ‘Disposable template’ to be checked as turned on (click Apply), then the ‘Default disposable template’ to either be ‘(none)’ or ‘`debian-12-xfce-stateless-dvm`’ itself.
- 4. In Persistent mode: Create & Configure any new AppQubes based on ‘`debian-12-xfce-stateless`’ that you want to exist across multiple stateless boot sessions and choose to store them in storage pool ‘`varlibqubes`’ (Advanced tab).
- 5. In Stateless mode: You are free to now use any AppQubes based on ‘`debian-12-xfce-stateless`’ and DisposableQubes based on ‘`debian-12-xfce-stateless-dvm`’, which appear to remain fully stateless by storing ALL data in traditional image files within Dom0’s ‘`/var/lib/qubes`’ directory (that directory gets wiped and reset back to match the state of

your last persistent session once your stateless session is powered down).

One may need to think about re-creating more or all of their system's various types of qubes to be fully stateless like this, if needing such levels of statelessness. It should be possible to make every single qube on one's system be fully stateless in '`varlibqubes`' storage pool.

With higher fully stateless RAM space demands, you may need a computer with higher amounts of hardware RAM and an increase to the ' `dom0_mem=max:10240M`' setting in my Qubes Stateless step-by-step instructions to be set meaningfully higher than '`10240M`'.

5. The Pull Out Method

One approach I experimented with last year was to boot from a USB drive, and once Qubes Stateless was at the login screen, I just pulled out the USB drive, and the system seemed to continue working fine.

I haven't done further testing on this approach yet, but it could be a powerful hardware enforced method for ensuring you are fully stateless.

Qubes Stateless is an advanced hybrid state system, so controlling your level of statelessness can be a complex thing to manage, if desiring to be fully stateless.

Feel free to ask questions and provide further thoughts.

Hey thanks you !! Your script is working correctly without trouble

You're welcome @qubesuser1234. Glad it's working out for you and others. Awesome that you're running a stateless Qubes OS Dom0 now! Super thanks to @xuy!

A couple updates to come soon. Stay tuned.

Hi @deeplow and @gonzalo-bulnes - It looks like we are beyond 30 days now, but I was hoping to update my step-by-step instructions post to make some important changes for anyone relying upon the post.

Is there anyway my account can be adjusted so I can edit that post beyond the 30 day limit?

Rather than continually "spamming" this thread with new big, long, repetitively new step-by-step instruction posts whenever there may be an instructions update, it seems more clean & correct to be able to edit the single instructions post and then make a brief notification/explanation post with a link to the "canonical" instructions post for those who may be further interested.

Thanks!

I made that post a wiki, does that help @qstateless ?

Would it be worth splitting the topic so that it becomes a guide on its own (still a wiki, but as the first post of its own topic)?

Thanks @gonzalo-bulnes! Edit ability for the post solves, so a "wiki" works I guess.

As far as making the post its own independent topic, I'm not sure about that yet. This version of instructions are primarily meant to be a step-by-step implementation of @xuy's how-to posts within this thread. I may decide to fundamentally upgrade and re-implement the whole Qubes Stateless system in the future, so that would certainly deserve its own topic.

For now, let's just keep this post in-place, and I will let you know if that should change.

Thank you!

Today, 2023-12-08, I have updated the Step-by-Step Instructions for Qubes Stateless.

This update includes an instruction to permanently disable Dom0 Swap.

This update also includes many re-written explanations to better explain what Qubes Stateless is and how it works. For those making use of Qubes Stateless, it may be worth re-reading.

Feel free to ask any questions or make any comments. Enjoy!

nealr December 9, 2023, 9:15am 63

I read this post yesterday and read it closer today.

I can see where this would be useful for a live USB setup, but I think it would be better as a live DVD, a sort of "take your hard drive free laptop to DEFCON' kinda thing, where you can leave it sitting somewhere and nobody can do anything that would survive a restart.

I am not sure what value there is to having dom0 running from ram on a system that does have a disk. If something in a VM can reach out and compromise dom0, it's already compromised Xen, and it can fiddle with VMs on disk. I guess dom0 mods for persistence, something that kinks one or more VMs each time the system started, would be a potential problem?

But the "first, let's assume dom0 is compromised" seems like an enormous stretch. A quick Google shows there have been four 'game over' Xen breakouts during the life of this project. Has there every been a proof of concept demonstrated?

Isn't there a trusted boot process that would render persistence impossible?

Hi @nealr

Good thoughts…

From a hardcore security perspective, you are generally right that a traditional boot drive is still accessible and writable, even with Dom0 running from RAM. So a breakdown of Qubes OS security via a Xen VM breakout exploit could still compromise one's system across restarts, if a persistent drive is used.

Detectable at least, with a TPM. I've read about things like UEFI Secure Boot, Heads, Anti Evil Maid, TrenchBoot (currently being integrated for Qubes AEM by 3mdeb), etc.

Yes. Using write-protected USB or read-only DVD boot media would certainly increase one's ability to eliminate the possibility of persistent exploits that could otherwise survive a system restart.

I see a few general reasons for making a Qubes Dom0 boot session be disposable in RAM…

- System Administration
- Anti-Borking
- Anti-Forensics
- Anti-Fingerprinting
- System Security

System Administration:

- Having to account for and manage less state in less places can generally help ease sysadmin burdens. I personally take advantage of this benefit across a small fleet of multiple computers I administer. I'm planning to go further and use this to eliminate the use of local drives via RAM-based Qubes OS over PXE Network Boot.

Anti-Borking:

- If one wants to run scripts, tests, etc that might bork or mess up one's Dom0, then running from RAM allows for a quick reset to clean system state. I personally have a Qubes-based development

test computer setup specifically for this.

Anti-Forensics:

- On a normally-functioning and non-compromised system, one can achieve a greater level of anti-forensic / disposable computing for the entire Qubes OS system by having no Dom0 logs/state saved.

Anti-Fingerprinting:

- If someone needed their Qubes OS environment to have the same fingerprint between multiple restarts, then a RAM-based Dom0 would help with achieving that.

System Security:

- With a persistent boot drive, while running Dom0 from RAM, a Dom0 exploit would have to take the extra step of reaching out to compromise the drive or drive contents, beyond compromising the running Dom0 environment within RAM. So there would probably be some unaware exploits that do not assume Dom0 is even in RAM, where one's system could be further protected against such exploits from persistent compromise beyond a restart. Not absolute security, but one more step further.

Of course, one could potentially use write-protected USB or read-only DVD boot media too. I experimented with RAM-based Dom0 and write-protected USB last year using Qubes 4.1, which I believe I wrote about some in this thread.

Layering various system attributes can potentially increase the fitness of a RAM-based Dom0 for various use cases and use conditions. A RAM-based Dom0 is another key attribute of choice that is now accessible to all who may find a use for it. Big thanks to @xuy for the technique.

Thoughts?

nealr December 10, 2023, 2:15am 65

Yes, that clarified things for me.

My LinkedIn profile says R&D oriented CTO, and that's pretty much why I'm in here. I talk to people who handle potentially "hot" data, or who operate in dangerous places. My big picture goal is living with Qubes daily on an HP Z420, then acquiring a stout laptop, likely a Dell Precision Xeon machine, and I expect pretty soon I'll start getting messages that say "So … do I need to get this Qubes stuff, too?"

I'm more of an integrator than a developer, and I'd be advising people in operations.

I've encountered an Evil Maid in the wild and it is an ongoing issue, one that I suspect will ramp up as things in the U.S. increasingly destabilize. Ensuring the system I come home to is running the same OS I shut down when I left is important to me, and to others.

Write protected USB? Does such a thing actually exist? I recall having write protected SD adapters for microSD, this was just a simple plastic slide that put one pin out of service. I need to do some research here.

Having dom0 in ram looks like it's good for development, testing, etc, but the added capital cost for groups of irregulars who struggle to get just **A** laptop that'll run Qubes … my systems have the capability to do this, but as a rule I do not think that will be the case.

I moved at the start of the month and now I've got an environment where I can push off, do a half turn while rolling, and be in front of my second Z420, instead of having to run up and down stairs. I have a couple SATA SSDs, a couple small PCIe NVMe, and an external USB to SATA drive carrier. I'm working through the production scenarios - a person in the field with a single laptop, a single external drive, how

do they install/run/backup? How do they handle seizure or theft of a machine?

So … I can see a dozen things I need to do prior to ever considering mucking around in dom0 internals and I'm likely the only one in my environment who would use this tmpfs solution. That could maybe change if a true read only USB device were available, but I suspect there are other keyring U2F devices that would accomplish more.

@nealr

Glad it helped!

Yes. Having a securely measured and verifiable boot process, remote backup, and battery-redundant remote video/sensor physical monitoring of one's spaces & devices would likely be important to crafting an Evil Maid solution. The Haven app was made to help with evil maid attacks, I believe.

Yes. Write protected USB devices do exist. I've used USB thumbdrives with a WP switch, and SATA to USB adapters with a WP switch, and USB proxies that had a WP feature. Not sure how their circuitry is implemented. Maybe usually a simple one pin disable slide, like you mentioned.

Without checking, I've also never heard of true RO USB (read-only), as in being like ROM (read-only memory). I'm just familiar with WP USB (write-protected). Since one probably has to be able to write/rewrite data onto the USB storage device to be generally useful, any read-only/write-protection features would probably need to be switchable or added in-line as a secondary device. A write-once/non-rewriteable optical disk (DVD, etc) via a USB connected reader is the only thing I could think of for true RO USB.

Nice. I know that feeling well. I have had to do the repetitive up and down stairs sysadmin bit many times.

If a network could be utilized without major issues (and that's potentially a big IF in hostile scenarios), then, for local data purposes at least, I'd think it would be ideal to simply not store such data locally, and just access/load such things remotely instead.

Best of luck with your pursuits! Qubes is really a fun and powerful OS for us security-minded geeks. With how easy other monolithic OSes can be penetrated, like a hot knife through butter, I think something like Qubes is a necessity to those wanting a fighting chance at having their digital stuff remain their own.

I've implemented Qubes into some non-technical people's lives too, however, it can take consistent IT assistance. For the extra effort, either be paid very well, love the people, or love the mission!

Cheers!

ddevz August 15, 2024, 9:40pm 68

I've been looking at flashable firmware on PCI devices. I wanted to mention that if you wanted a truely "persistence impossible" situation, you would need to address flashable firmware in your PCI devices, as i do not believe that gets addressed in "secure boot/measured boot/trusted boot".

Here is a link of my attempt at addressing the situation: How to avoid flashable firmware compromises via pass-through PCI devices?