When security matters: working with Qubes OS at the Guardian

Philip McMahon ⋮ 15-19 minutes ⋮ 4/4/2024

This post was updated on 9th April 2024 following feedback from the Securedrop Workstation/Qubes community

If you've ever scrolled through the Guardian homepage, you may have come across the 'Contact the Guardian securely' banner. This links to a page explaining how to share sensitive information with the Guardian in a number of different ways. Of these, the one that offers the best security is SecureDrop.

SecureDrop is a system developed by the Freedom of the Press Foundation to enable sources to contact media organisations anonymously. It is a system widely respected and deployed in a large number of newsrooms.

The current configuration of SecureDrop involves three separate machines – one connected to the SecureDrop server running Tails OS, another 'air gapped' entirely offline machine for decrypting the messages on, and (typically) a final online machine for emailing PGP encrypted messages to the relevant journalists. Transferring messages between the machines typically involves a lot of carefully managed USB sticks, time and patience.

The Freedom of the Press Foundation is working on a more streamlined solution for journalists working with SecureDrop called SecureDrop Workstation. SecureDrop Workstation is based on Qubes OS. It removes the need for a separate 'air gapped' machine (see here for details of the tradeoffs), replacing it with an offline virtual machine or 'Qube'. Messages can be downloaded and read on the same machine – on an interface that looks much like a standard chat client, with all the decryption handled in the background.

This blog post is focused on some of our learnings whilst working with Qubes OS. Our objective here was to set up multiple SecureDrop Workstation machines, with the SecureDrop Workstation app installed along with some other useful tools.

Configuring a Qubes workstation was a new challenge for the team as we abandoned years of experience writing Infrastructure as Code for the cloud and started learning how to write Salt configuration. Salt (also know as SaltStack) is a management engine available by default in Qubes.

Most of the Salt code in this post would have been impossible to write without inspiration from the SecureDrop Workstation repo (thank you, Freedom of the Press Foundation!). It's also worth noting that we are not Qubes experts, this is just what worked for us at the Guardian.

A rough idea of what we were trying to achieve with Qubes is as follows:

- An offline VM based on Debian 11;
- Some packages installed from the default repositories and a custom repository;
- The VM should be 'amnesic' – after a restart it should be reset to a consistent state (so any generated files/downloaded data should be deleted);
- A custom package installed from a private repository hosted in Amazon S3;
- A Nautilus extension;

- A configuration file containing secrets that can’t be hard-coded.

To begin, let’s pick just the first of those items: an offline VM based on debian 11.

Salt configuration in Qubes starts with a .top file:

```
# guardian.top

base:
 dom0:
   - guardian-vms
```

What this top file says is “on dom0, apply the state guardian-vms” (dom0 is the root VM on Qubes OS)

The ‘state’ in question refers to a ‘state file’ – with the extension .sls. Here’s our first state file.

```
# guardian-vms.sls

create-guardian-template:
 qvm.vm:
   - name: guardian-template
   - clone:
     - source: debian-11
     - label: black
   - prefs:
     - netvm: ""


create-app:
 qvm.vm:
   - name: app
   - present:
     - template: guardian-template
     - label: green
   - prefs:
     - template: guardian-template
     - netvm: ""
```

In the state file above we do two things:

- Create the ‘guardian-template’ TemplateVM based on Debian 11, which is offline;
- Create a ‘guardian’ AppVM based on the template VM. It too is offline.

You can read more about Template and App VMs in the Qubes documentation. At a high level, the template VM (guardian-template) is where we install any relevant software. While this template doesn’t have direct access to the internet, we can still install packages on it via an updates proxy. The AppVM (app) is based on the TemplateVM. Every time ‘app’ is restarted, the file system (except the home folder) resets to the state of ‘guardian-template’. This is one of the many security features of Qubes OS – resetting the system to a known stable state helps ensure any malicious code installed during a session will be removed.

Installing Software

OK, fantastic, we've got an offline VM called 'app'. This VM could be useful for some basic tasks, but it doesn't yet have any useful software installed on it. Let's install some tools for viewing, editing and sanitising files. One option for installing software would be to simply open a terminal in the template VM and install some software with APT. To do that in the Salt configuration, we'll need a new state file

```
# install-packages.sls

install-packages:
 pkg.installed:
   - pkgs:
     - libreoffice
     - gedit
     - vlc
```

We also need to update our .top file to tell Qubes to apply the install-packages state to the guardian-template VM:

```
# guardian.top

base:
 dom0:
   - guardian-vms

 guardian-template:
   - install-packages
```

Installing software unavailable in default repositories

That's all very well if the software you want to use is available in the default repositories. What if you want to install something else? This is possible in Qubes, but is 'not recommended for trusted templates'. You can read the full details on installing software from other sources on the Qubes website – here we'll look at adding a package from another source to our template in as simple a way as possible.

For our use case we wanted to install dangerzone, a tool maintained by the Freedom of the Press Foundation which allows you to 'take potentially dangerous PDFs, office documents, or images and convert them to safe PDFs'.

The easiest way to install software from non-default repositories is to modify our template VM to give it network access (this is not best practice – see the next section for an improvement). We do this by setting the netvm property to sys-firewall. At the same time let's set the label to 'red' to indicate that this template is less trusted (now that it has a network connection):

```
# guardian-vms.sls

create-guardian-template:
 qvm.vm:
   - name: guardian-template
   - clone:
     - source: debian-11
     - label: red
   - prefs:
     - netvm: sys-firewall
```

Next, we need to update our install software state file to set up the repository from which dangerzone will be installed:

```
# install-packages.sls

# freedomofpress repo is required to install dangerzone
add freedomofpress repo:
 pkgrepo.managed:
    - name: "deb https://packages.freedom.press/apt-tools-prod bullseye
main"
   - keyserver: keys.openpgp.org
   - keyid: DE28AB241FA48260FAC9B8BAA7C9B38522604281
   - humanname: freedomofpress


install-guardian-dependencies:
 pkg.installed:
   - pkgs:
     - libreoffice
     - gedit
     - vlc
     - dangerzone # this package will now be available
```

At this point, we have the ability to open office documents and clean files within our AppVM. The VM itself is not connected to the internet, which reduces the chances of it being compromised.

Avoiding enabling networking on the template VM

There is a way of installing from custom repositories without giving network access to the template VM – which is preferable from a security perspective. For this to work, instead of fetching the key for the repository from a keyserver, we can load it from disk.

```
# install-packages.sls

# freedomofpress repo is required to install dangerzone
add freedomofpress repo:
 pkgrepo.managed:
    - name: "deb https://packages.freedom.press/apt-tools-prod bullseye
main"
   - key_url: salt://dangerzone/fpf-apt-tools-archive-keyring.gpg
   - humanname: freedomofpress
```

```
install-guardian-dependencies:
 pkg.installed:
   - pkgs:
     - dangerzone # this package will now be available
```

For the above to work, you will need to have downloaded the key and moved it to /srv/salt/dangerzone/fpf-apt-tools-archive-keyring.gpg on dom0. There are instructions on downloading the key here.

Alternatively, you can add an extra step after installing the software which removes the netvm from the template. See here for documentation dealing with dependencies/ordering between states.

Disposable VMs

At the Guardian, we needed to go further. We use Qubes VMs to process sensitive documents, and we'd like as much certainty as possible that those documents can only be read by specific people. With the current AppVM behaviour, while the packages and operating system of the VM will be reset on every restart, the contents of the home folder persists. For a completely 'amnesic' VM that wipes all documents on a restart, we need to use a disposable VM, or 'DispVM'.

Disposable VMs, rather than being based on TemplateVMs, use a special kind of AppVM as their base template. We need to make some changes to our state files:

Updated guardian-vms.sls:

```
# guardian-vms.sls

create-guardian-template:
 qvm.vm:
   - name: guardian-template
   - clone:
     - source: debian-11
     - label: black


create-guardian-template-disp:
 qvm.vm:
   - name: guardian-template-disp
   - present:
     - template: guardian-template
     - label: black
  - class: AppVM
   - prefs:
     - template: guardian-template
     - template_for_dispvms: True
     - netvm: ""


create-app:
 qvm.vm:
   - name: app
   - present:
     - template: guardian-template-disp
     - label: green
     - class: DispVM
   - prefs:
     - template: guardian-template-disp
     - netvm: ""
```

Note that we now have three VMs or Qubes defined: the template, the AppVM we use as a template for our DispVM, and the 'app' DispVM itself. 'app' is a 'named disposable' Qube. We've now got an amnesic VM with some useful software installed on it.

Security Notes on the above (thank you Francisco Rocha):

- In the above setup, by default there is still a network path via the disposable of the disposable Qube. To prevent this , you can either change the default disposable Qube so that it does not have network access, or prevent the disposable Qube from using other disposables. In the UI I did this via Qube Settings>Advanced>Default disposable template>(none). See here for how to do it in Salt
- Disposable Qubes are technically not fully amnesic – see the documentation for details.

Customising the VMs

Next, we wanted to add a custom right click menu option to the file explorer that would run a script on the file (to save users having to open up a terminal window). We did this using a nautilus extension. Nautilus extensions are installed inside the home folder, which for our 'app' VM will be based on guardian-template-disp – our disposable template. For this, we'll need a new state file, an updated top file and the extension itself:

```
# guardian.top

base:
 dom0:
   - guardian-vms


 guardian-template-disp:
   - install-nautilus-extension


 guardian-template:
   - install-packages
```

```
#install-nautilus-extension.sls
nautilus-extension:
 file.managed:
    -  name:  /home/user/.local/share/nautilus-python/extensions/nautilus-
extension.py
   - source: salt://guardian/nautilus-extension.py
   - makedirs: true
   - user: user
```

```
    - group: user
    - mode: 555
```

```
# nautilus-extension.py

from urllib.parse import urlparse, unquote
from gi.repository import Nautilus, GObject


class                      RunSpecialScriptMenuProvider(GObject.GObject,
Nautilus.MenuProvider):


 def run_special_script(self, menu, files):
   for file in files:
     file_path = unquote(urlparse(file.get_uri()).path)
     print("Hi! I am processing " + file_path)


 def get_file_items(self, window, files):


   item = Nautilus.MenuItem(name='Run special script',
                            label='Run special script'
                            )
   item.connect('activate', self.run_special_script, files)
   return item,
```

The file.managed action in install-nautilus-extension.sls will look for a file on dom0 at /srv/salt/guardian/nautilus-extension.py and copy it to the location 'name' on the VM which is run on (in our case guardian-template-disp), creating any required intermediate directories and applying the file permissions indicated.

Let's take a look at dealing with configuration that you don't want to hard code into your state files or other installation files (perhaps because they're checked in to a public repo). We can make use of the jinja2 template support in Salt to swap in values from a config file. In the example below, let's assume we want to set up an AWS credentials file on one of our VMs, with an access key and secret key.

Here's the template credentials file:

```
# credentials.j2

[default]
aws_access_key_id = {{ access_key_id }}
aws_secret_access_key = {{ secret_access_key }}
```

We need our config file itself:

config.json:

```
{
 "access_key_id": "ACCESSKEY",
 "secret_access_key": "reallyverysecretkey"
}
```

Then we need a state file to wire it all together:

```
# setup-aws-credentials.sls

{% import_json "guardian/config.json" as config %}


install-config:
 file.managed:
   - name: /home/user/.aws/credentials
   - source: "salt://guardian/credentials.j2"
   - template: jinja
   - context:
       access_key_id: {{ config.access_key_id }}
       secret_access_key: {{ config.secret_access_key }}
   - user: user
   - group: user
   - makedirs: True
```

We'd also need to update our top file to apply the new state file to a Qube. There wouldn't be much point having AWS credentials for an offline Qube, so in the topfile below I've invented a new online Qube – which would need defining in guardian-vms.sls.

```
# guardian.top

base:
 dom0:
   - guardian-vms


 guardian-template-disp:
   - install-nautilus-extension


 guardian-template:
   - install-packages


 aws-management-app:
   - setup-aws-credentials
```

Finally, let's take a look at actually applying state files to a Qubes system.

For this project, we used an RPM package to bundle up our state files and install them into dom0. Making this package was made a lot easier thanks to a tool called FPM. After installation, the folder structure on dom0 looks like this:

```
/srv/salt/
   guardian.top
   install-nautilus-extension.sls
   guardian-vms.sls
   install-packages.sls
   setup-aws-credentials.sls
   guardian/
       nautilus-extension.py
       credentials.j2
       config.json
```

Once the files are installed, we need to tell Qubes to make use of them.

Firstly, let's enable our .top file:

```
sudo qubesctl top.enable guardian
```

To apply a state to dom0, you can use the following command:

```
qubesctl --show-output state.apply guardian-vms
```

state.apply takes a comma separated list of state files to apply (without the .sls extension). If you just want to apply all states associated with an enabled topfile, then you can run:

```
qubesctl --show-output state.apply
```

To apply all enabled states to a specific VM (eg. if we have added a new package to install-packages.sls), you can skip dom0 updates and target that VM:

```
qubesctl    --show-output    --skip-dom0    --targets    guardian-template
state.apply
```

Or if you want to just apply a specific state to that VM:

```
qubesctl --show-output --skip-dom0 --targets guardian-template
state.apply install-packages
```

Finally, if you want to apply Salt configuration to dom0 and all of your Qubes you can use the below command. Beware this will take a really long time …

```
sudo qubesctl --show-output --all state.apply
```

Getting to grips with Qubes OS and Salt was a sharp learning curve for us. Hopefully this blogpost will be of use to anyone trying to do the same. If you have any questions or feedback for us, please contact digital.investigations@theguardian.com.

Update 11/04: Following publishing this post, we received some excellent feedback from Ben Grande detailing some improvements/issues with the code which is recommended reading if you are attempting to do something similar. You can read it here