The story of adding TPM support to oVirt

Milan Zamazal
mzamazal@redhat.com

FOSDEM
February 5, 2022

What is this talk about: The device

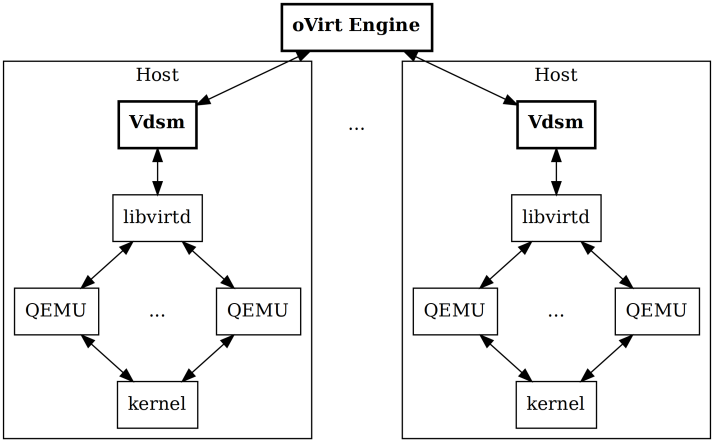
TPM (Trusted Platform Module): A piece of hardware providing some security-related features and a small amount of secure memory.
In virtualization, it can be just another emulated device in a VM (virtual machine):

```
qemu-kvm ... -device tpm-crb,... ...
```

Useful for Windows 11 / 2022, BitLocker, LUKS, ...

What is this talk about: The software

oVirt: An open-source distributed virtualization solution.
Based on libvirt and QEMU:



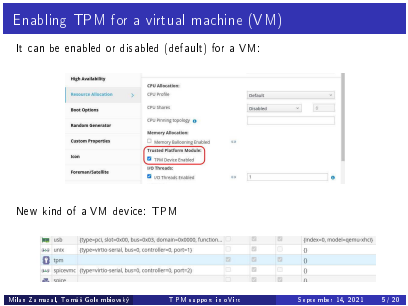
What is this talk about: The problem

- Not only users suffer from software ~~problems~~ complexity, developers struggle with it too.
- We learn, again and again, that seemingly easy things may not be that easy.
- Insight into virtualization feature development and the related problems (do you sometimes wonder why things work the way they work?).

What is this talk not about

- TPM itself.
- User experience — already presented at oVirt 2021 conference:

TPM support in oVirt



Adding TPM: Solution

oVirt uses libvirt and libvirt supports TPM:

```
<devices>
  ...
  <tpm model="tpm-crb">
    <backend type="emulator" version="2.0"/>
  </tpm>
  ...
</devices>
```

Is it really so easy? *Spoiler warning...*

Adding TPM: Solution

oVirt uses libvirt and libvirt supports TPM:

```
<devices>
  ...
  <tpm model="tpm-crb">
    <backend type="emulator" version="2.0"/>
  </tpm>
  ...
</devices>
```

Is it really so easy? **No.**

We cannot handle mathematics

Crises of mathematics:

- $\sqrt{2} =?$ $\rightarrow$ irrational numbers
- **Infinitely small** in calculus? $\rightarrow$ some 150 years later: $\lim_{x \to 0}$ etc.
- Foundational crisis: How to use **natural numbers** consistently *and* completely? $\rightarrow$ nothing better so far than reducing our requirements
- ... ?

Can we handle software better?

Obviously, no.

```
Jan 03 07:57:22 swallow kernel: BTRFS: error (device dm-0)
in btrfs_del_inode_extref:138: errno=-2 No such entry
Jan 03 07:57:22 swallow kernel: BTRFS info (device dm-0):
forced readonly
```

But we have to (some way).
We use maths for thousands years after all...

What's the problem with TPM?

Why is the following not all what is needed?

```
<tpm model="tpm-crb">
  <backend type="emulator" version="2.0"/>
</tpm>
```

What's the problem with TPM?

Why is the following not all what is needed?

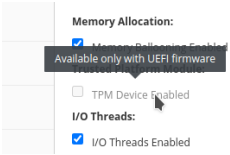
```
<tpm model="tpm-crb">
  <backend type="emulator" version="2.0"/>
</tpm>
```

It *is* all what is needed, **unless**:

- Software and VM configuration requirements are not satisfied.
- The VM is started on another (or reinstalled) host.
- The VM is redefined on the same host.
- Snapshots are used.
- There is API/UI needed.
- ...

Requirements: UEFI

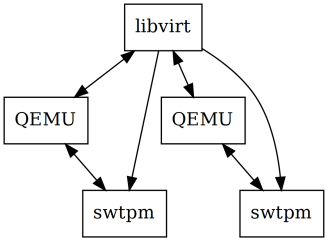
TPM emulation works only with UEFI firmware, not BIOS.
Not a big problem, but must be addressed in e.g. the UI etc.



Architecture differences: Concerns x86_64, not POWER.

Requirements: swtpm

swtpm: Software TPM emulator, used by QEMU.



Needed, but only for TPM ⇒ nothing depends on it
⇒ Vdsm must depend on it ⇒ new cluster compatibility version needed.
(Cluster compatibility version: Ensures all hosts provide certain set of features.)
It shouldn't be too restrictive ⇒ configuration option ⇒ users may override the requirement.

Let's define the device

```
<tpm>
  <backend type="emulator" version="2.0"/>
</tpm>
```

Let's define the device

```
<tpm>
  <backend type="emulator" version="2.0"/>
</tpm>
```

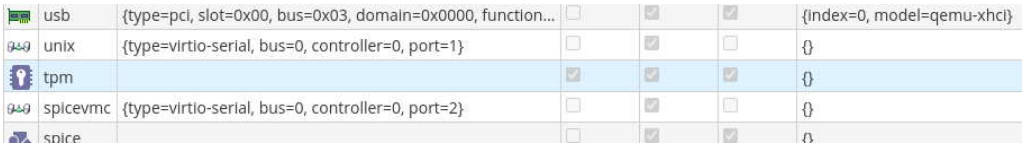
Wrong!

The implied model is `tpm-tis`, the older one. The modern one must be specified (for x86_64 architecture):

```
<tpm model="tpm-crb">
  <backend type="emulator" version="2.0"/>
</tpm>
```

Device type icon

What's the problem with one small icon?!



	usb	{type=pci, slot=0x00, bus=0x03, domain=0x0000, function...	☐	☑	☑	{index=0, model=qemu-xhci}
	unix	{type=virtio-serial, bus=0, controller=0, port=1}	☐	☑	☐	{}
	tpm		☑	☑	☑	{}
	spicevmc	{type=virtio-serial, bus=0, controller=0, port=2}	☐	☑	☐	{}
	spice		☐	☑	☑	{}

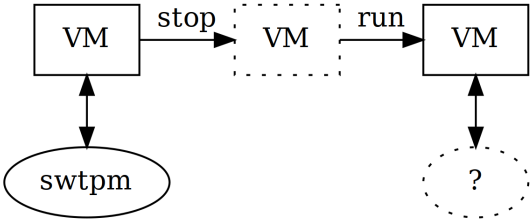
Surprisingly, not a small amount of work:

- Icon design is not a marginal task, must fit well into the UI.
- Cooperation with a graphic designer needed, several possible versions discussed.
- One-time little thing may be sometimes more difficult to get done than a bigger feature.

TPM data persistence

TPM is not very useful without data persistence.

But TPM/swtpm data persists only as long as the VM remains defined on the given host or the VM is migrated to another host.



The data must be saved and restored as needed and transferred between the host (Vdsm) and the manager (Engine).

How to keep the data persistent?

No shortage of ideas about transferring the data:

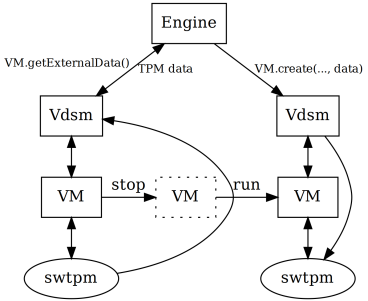
- New API calls and/or API call arguments
- libvirt domain XML metadata
- Shared storage
- Events

All with their pros & cons.

A complicated decision (but easier than variable naming!).

Actual data persistence implementation

Data is stored in the Engine database.
Engine → Vdsm: An additional argument to `VM.create` API call.
Vdsm → Engine: A new API call `VM.getExternalData`.



New cluster compatibility version needed again.

Retrieval of modified TPM data

`VM.getExternalData` is called **after** VM is stopped (shutdown from Engine, shutdown from the guest OS, VM crash) but **before** `VM.destroy` (VM cleanup on the host) is called.

It's important not to block `VM.destroy`, the VM cannot start again without it. As a safety break, `VM.getExternalData` is called at most three times before `VM.destroy` is eventually called (and contingent TPM data modifications are lost).

It's all only best effort. TPM data stored in a real hardware can also be lost if the hardware dies. Virtualization is not a replacement for proper backups.

Powering off a VM

Problem: On power off action (stopping a VM immediately, without proper shutdown), mighty `VM.destroy` Vdsm API call is used, which wipes everything about the VM from the host. Due to this implementation limitation, we cannot retrieve TPM data **after** the VM is down.

Cause: No separate API calls for powering off and cleaning up the VM — implementation shortcut. Difficult (although not impossible) to change.

~~Excuse~~ **Solution:** Power off is an unsafe operation by definition, let's retrieve the data **before** calling `VM.Destroy` while the VM is still running.

Lost VM

Problem: What if the host on which it is running gets lost? VM disks are on a shared storage but TPM data is stored locally and its modifications would be lost.

Workaround: Periodic retrieval of TPM data. To avoid unnecessary calls and data transfers, Vdsm reports cryptographic hashes of changed TPM data to Engine.

Are TPM data retrievals reliable?

No. If the VM is running then it may happen that we read swtpm data while it is being modified $\Rightarrow$ we may get corrupted data.

This can be prevented by a better way of writing the data. Suggested that to the swtpm developer one Friday afternoon and it got fixed in a couple of hours (an excellent response, Stefan!).

Are TPM data retrievals reliable?

No. If the VM is running then it may happen that we read swtpm data while it is being modified $\Rightarrow$ we may get corrupted data.

This can be prevented by a better way of writing the data. Suggested that to the swtpm developer one Friday afternoon and it got fixed in a couple of hours (an excellent response, Stefan!).

But: The fix is not available immediately downstream and secure boot NVRAM data suffers from a similar problem.

Workaround: Checking for consecutive data changes and reporting the data only once it is stable.

Are TPM data retrievals reliable?

No. If the VM is running then it may happen that we read swtpm data while it is being modified $\Rightarrow$ we may get corrupted data.

This can be prevented by a better way of writing the data. Suggested that to the swtpm developer one Friday afternoon and it got fixed in a couple of hours (an excellent response, Stefan!).

But: The fix is not available immediately downstream and secure boot NVRAM data suffers from a similar problem.

Workaround: Checking for consecutive data changes and reporting the data only once it is stable.

Another risk: Data format changes between software versions.

Snapshots

Where to store TPM data for snapshots? Decision: In snapshot OVF.

No problem with **offline** snapshots — the VM is not running and we have its data.

But how about **live** snapshots? How to get data corresponding to the snapshot?
They utilize periodic retrievals currently, users must take it into account.
This should be improved in future, but it's not easy to design something reliable since we work with a running system.

Snapshots with memory

Seemingly the same as with live snapshots without memory.
But not exactly:

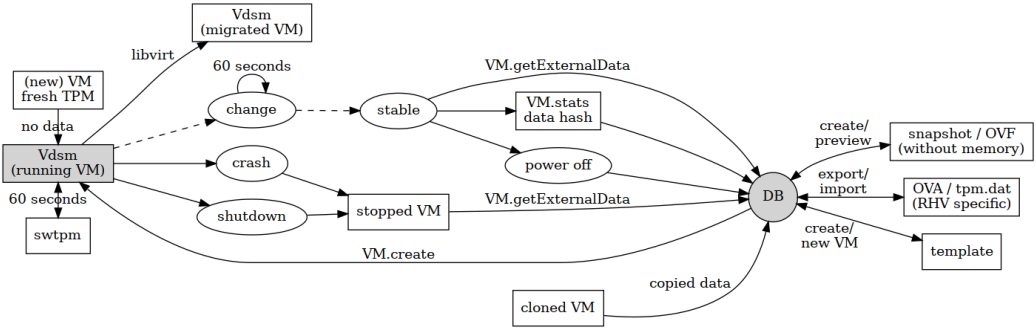
- The guest OS is frozen when dumping RAM (good).
- If TPM data is inconsistent with RAM then the guest OS can fall into trouble (bad).

Solution: Retrieve TPM data when the guest OS is frozen and the data cannot be modified.

Problem: libvirt doesn't have support for this.

Workaround: Snapshots with memory have to be disabled (ugly).

A complete picture of TPM data updates



Logging

Data transfers ⇒ data may appear in logs.
It is sensitive data ⇒ data may *not* appear in logs.

Sensitive data is usually not logged directly, **but**:

- It can be logged together with other data, e.g. when logging API call parameters or responses.
- Beware when logging exceptions!
- Double check your data logging prevention mechanism works as expected.

Other security considerations?

An emulated TPM device is not a sealed hardware device.

libvirt offers protecting the data by a passphrase:

```
<tpm model="tpm-crb">
  <backend type="emulator" version="2.0">
    <encryption secret="..."/>
  </backend>
</tpm>
```

But: Can we store the data anywhere without storing its passphrase in a similarly accessible location?
In oVirt, no. Untrusted people must not have access to the database, storage, logs, or root access to the hosts.

But it's not only about TPM data

It's also about **other data**.

We must write TPM data to a local file system on the host when starting a VM, which has some implications:

- We must ensure that the data provided by a Vdsm API call cannot be written elsewhere than to the intended destination — even when malicious call arguments are sent, such as an invalid VM id or specifically crafted tar archive contents.
- `shutil` Python library is insecure and can write data outside the destination directory when a tar archive is unpacked. GNU tar can prevent that and was used instead.

How about the guest OS?

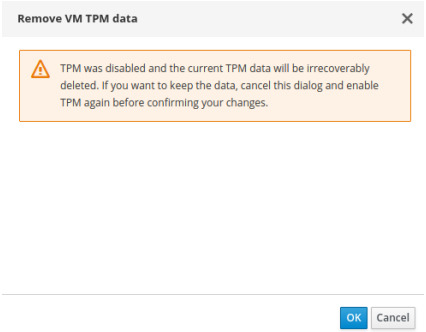
Not all guest OSes support virtual TPM properly.

Let's not give users false hope: TPM can be enabled only for RHEL >= 7 or Windows >= 8.1 / 2012 by default.
Other guest OSes may be added by changing oVirt Engine configuration.

```
os.other.devices.tpmAllowed.value = false
os.rhel_7x64.devices.tpmAllowed.value = true
os.rhel_7_ppc64.devices.tpmAllowed.value = true
os.windows_8.devices.tpmAllowed.value = true
os.windows_8x64.devices.tpmAllowed.value = true
os.windows_2012x64.devices.tpmAllowed.value = true
```

Other work

- VM clones, templates
- OVA export/import
- REST API
- Web UI (dependencies, confirmation dialogs about data deletions, . . .)




```
<template>
  <name>template-with-tpm</name>
  <vm id="..."/>
  <tpm_enabled>true</tpm_enabled>
</template>
```

Secure boot NVRAM

A similar feature is handling secure boot NVRAM data.

Thinking about this in advance allowed sharing most of the code for both the features and making the implementation easier.

In summary

TPM support was added to oVirt as a result of collaboration among many people. And it was possible only due to solid support in the underlying platform: libvirt, QEMU, swtpm, ...
It has been an interesting journey, full of challenges, there are still things to be done.
The amount of work wasn't surprising but there are always surprises in software development.

Some stats (oVirt only):

- ~30 patches.
- ~160 files modified, added or deleted.
- ~4000 lines of code modified, added or deleted.
- ~15 Bugzilla entries.
- ~50 e-mails sent by me.

So what?

Hopefully, the lessons learned from this story may be useful when adding new virtualization features, not necessarily only TPM, to other products.

- Read all the related documentation.
- Don't rely on undocumented features.
- Always look into logs, even when everything works.
- Never forget about security.
- Think about all possible scenarios.
- Consider possible implementation of similar features.
- Avoid implementation shortcuts.
- Make it easy for users; add extra options where needed.
- Communicate and be helpful when looking for solutions.
- Have the final goal on mind.

Thank you!

Questions?

<devel@ovirt.org>
<users@ovirt.org>