Software tpm alternative

We propose replicating much of PCR structure, but implementing it in software and using the results to generate a symmetric encryption key. This key will be used to encrypt sensitive material before storing it on disk.

The PCR usage table from the tpm 2.0 spec, as well as other references, can be found in the Appendix.

We replicate the tpm PCR measurements. However, each measurement is appended to the previous, following PCR-style extension. Each item is measured using sha256 hashing.

The tpm measurement algorithm is as follows:

```
PCR[N] = HASHalg( PCR[N] || ArgumentOfExtend )
```

If the original is `sha256:12345` , and the next measure is `sha256:aaabbcc` , then it concatenates the two together and hashes the result.

Without proper PCRs, and looking for a single measurement, we:

1. Take each element, e.g. CPU ID or EFI file or kernel or tar of a filesystem contents.
2. Digest it using sha256 hashing.
3. Concatenate it to the previous result.
4. Digest the result using sha256 hashing.

This is now used as the "previous result" for the next stage.

The elements to measure are the following, all in order. The items are optionally selectable. Different clients may choose to measure different items, the same way that clients on devices with tpm may choose to seal to only a certain subset of PCRs. If any is not selected by the client, that stage simply is skipped, and measurement moves to the next stage.

1. CPU ID or other hardware information to cover removal of disk with all its software
2. Firmware settings:
    a. EFI: from `/sys/firmware/efi`
        i. Equivalent of `efibootdump` , `efi-readvar` , etc.
        ii. Still probably need `dmidecode`
    b. BIOS: from `dmidecode` / `biosdecode` or equivalent in /sys/firmware see https://www.kernel.org/doc/Documentation/ABI/testing/sysfs-firmware-dmi-tables
        i. Serial, UUID, what else can be unique?
3. If EFI: firmware file from `/boot/EFI`
4. GPT partition table for boot disk
5. kernel command line from `/cmdline`
6. `initrd`
7. `kernel`
8. kernel modules `/sys/module/`
9. `init`
10. init config
11. Optionally other files or directories. Likely candidates are `systemd` , container directories, `runc` , `containerd` , etc.

For each stage:

1. If a directory or multiple files, `tar`
2. Hash with sha256
3. Add to previous stage

Use the final sha256 as an AES256 symmetric key.

Continuing our example above, the generated key would be used to encrypt the asymmetric key used to secure communications, as well as the symmetric key used to encrypt local data.

Managed Upgrades

The problem of OS upgrades applies here, as it does to devices with secrets sealed to PCR states. In the case of managed changes or upgrades, the process is as follows:

1. Device receives upgrade, which changes any measured element. As the device is running, the current measurement is in memory.
2. Before rebooting, device performs new measurements, creating a new key. It now has key for existing and new.
3. Device creates a new copy of each encrypted secret, saving it next to the existing ones. It now has both original and new ones.
4. Device reboots into new configuration.
5. On startup, it sees the new encrypted keys, uses them.
6. Device removes old keys.

Concerns

To be very clear, any such process has a root of trust issue. TPM uses the hardware itself as the Core Root of Trust Measurement (CRTM), specifically using Static Root of Trust. It also has special modes to do a late measurement with Dynamic Root of Trust. In our case, nothing is properly confirming the root of trust. We accept that.

This is not perfect. It has several shortcomings.

- The tpm measured boot process measures each next stage *before* executing it. This guarantees that the file hasn't changed. When we measure something *after* the fact, it might have moved. For example, if a later stage (malicious or unintentional) moves a kernel module file after loaded, we would be measuring the wrong thing.
- The tpm measured boot process has access to elements we do not, such as firmware or nvram.
- We do our calculations in memory in a multi-user space, which is vulnerable, whereas the measured boot process is performed by the primary executor in single-user space.

Nonetheless, on a continuum from “everything in the clear on disk” to “everything secured by tpm”, this is much more in the middle and provides a higher level of protection.

Appendix

According to the tpm 2.0 spec specifically Section 2.3.4, PCR numbers are as follows. The tpm 1.2 spec also is available.

A linux-specific list is here:

Another good reference from the tianocore docs is here.