# Towards Secure Unified Kernel Images for Generic Linux Distributions and Everyone Else

*Lennart Poettering, LPC 2022, Dublin*

# Status Quo Ante

- Kernel Image is signed, built by OS vendor
- Initial RAM disk (initrd) image is not signed (nor encrypted), generated locally
- Initrd generators are usually shell scripts that try to generate minimal images via parsing “ldd”, implementing a secondary, weaker form of packaging dependencies on individual file level
- Initrd mixes code and configuration
- Parameterization happens via kernel command line and by including configuration files in initrd image

*(This is the status quo ante in pretty much all major, popular general purpose Linux distributions, commercial ones or otherwise)*

# Problems

The status quo ante has many major problems:

- No integrity of initrd image 🤯
- No confidentiality of parameters in initrd image 😭
- Initrd mixes code and local configuration □
- Fragile because of the way it is built (parsing of `ldd` output, usually) 🥺
- If included in TPM PCR measurements: non-deterministic values, that cannot be pre-calculated by vendors, and thus not signed (for use in signed PCR policies) 😔
- Untestable due to combinatorial explosion 😢
- Changing low-level OS configuration requires re-running of initrd generator 😿

# Benefits

The status quo ante also has various benefits:

- Very flexible, hackable and adjustable to local needs
- Typically, more minimal than package manager based approaches
- Already implemented, tested, and deployed

# Goals

- Pre-built initrds that can be signed by vendor
- Secure parameterization of kernels/initrds (guaranteeing both integrity and confidentiality)
- Stable, deterministic TPM PCR measurements that can be pre-calculated
- Some degree of modularity
- Kernel/initrd updates as atomic single-file updates

# Approach

1. Let's pre-build basic initrds on OS vendors' build systems
2. Let's include them in a *unified kernel image*, and sign them along with it
3. Let's introduce an extension concept, to maintain some modularity to cover for less common storage, networking subsystems (and similar) without having to pull support in the basic initrd.
4. Let's add a *credentials* concept allowing secure local parameterization of unified kernel images
5. Let's measure the unified kernel image components into PCR 11
6. Let's include signed TPM2 policy for expected PCR 11 measurement value in kernel

# Building basic initrd cpio

⏭️ Next Talk, by Zbigniew

# Unified Kernel Images

Combination of the following components in a single UEFI PE image:

- systemd-stub EFI stub
- ELF kernel image
- Basic initrd cpio
- kernel command line
- /etc/os-release
- (boot splash .BMP, if desired)
- (devicetree, if desired)
- Signed TPM2 policy information based on expected PCR measurements of the kernel

# Unified Kernel Images

- All that, combined into one PE file through a simple “objcopy” invocation (see systemd-stub(7) man page for details)
- And then signed as one for SecureBoot
- Can be executed directly by UEFI firmware, since a PE binary like any other
- Can be automatically enumerated by systemd-boot boot loader, with the OS information automatically extracted from the PE image for display in the boot menu

# systemd-stub PE UEFI stub

- Small piece of code that runs in UEFI mode.
- Searches its own PE sections for the aforementioned components.
- Ultimately invokes the contained kernel + initrd + kernel command line
- May display a boot splash before
- Loads and activates Devicetree section
- Measures contained kernel + initrd into PCR 11 (which is otherwise unused; discussed later)
- Picks up credential files from ESP (discussed later)
- Picks up initrd extension images ESP (discussed later)
- Passes PCR signature from PE section as initrd to kernel (discussed later)

# Extension Images

- Basic initrd included in unified kernel image – not extensible because pre-built by vendor.
- systemd-sysext → a tool implementing a “system extension” concept, that allows overlaying a cryptographically secured disk image onto /usr/
- systemd-sysext can work in host OS, but also in initrd
- System extension images are GPT disk images, implementing the *Discoverable Partitions Specification* with three relevant partitions: /usr/ file system (e.g. squashfs), partition with dm-verity data for that file system, partition with PKCS#7 signature of root hash of dm-verity data
- PKCS#7 signature is verified by kernel on mounting, with possible integration with IMA/IPE
- /usr/ tree in extension images is merged into initrd /usr/ via read-only overlayfs, should be purely “additive”, but this is not enforced

# Extension Images (continued)

- Extension Images are supposed to add major subsystems at once (let's say: a complete volume manager such as LVM, instead of an individual library), i.e. much larger granularity than packages
- Version handling is simple: each base initrd image declare an ID identifying the initrd/distribution project and a system extension API level. Extension images declare the same. Only if both match they can be activated.
- In UEFI mode systemd-stub searches for system extension image files in the directory the unified kernel PE image is invoked from. It loads them one by one into memory, generating an on-the-fly cpio initrd image from them, placing them in /.extra/sysext/
- systemd-sysext picks them up from there during early initialization of the basic initrd and activates them via dm-verity, overlayfs, …

# Parameterization

1. In a SecureBoot environment, passing parameters to the kernel is problematic if done without authentication.
2. Kernel command line is locked down, and sourced only from unified kernel image section
3. Authenticating/decrypting in UEFI is nasty, since most likely requires embedding TPM2 and OpenSSL stack in UEFI code. We’d rather avoid that.

# Credentials

Solution:

- systemd's "service credentials" are a concept for passing identity information, certificates, key material, passwords, and similar to services
- They can also be passed into kernels (and then are called "system credentials")
- Credentials can be encrypted and authenticated (AES256-GCM)
- Symmetric key derived from combination of local TPM (and possibly key stored in local file system /var/)
- In UEFI mode systemd-stub will look in the directory the unified kernel is invoked from for credentials, load them into memory and pass them as auto-generated initrd cpio to the ELF kernel, so that they'll appear in /.extra/credentials/

# Credentials (continued)

- System credentials loaded that way can be propagated down to individual services
- Credentials are understood by many systemd components
- Services will receive decrypted/authenticated credentials in simple regular files in the file system, all below a directory indicated via the $CREDENTIALS_DIRECTORY environment variable, private to the service
- Both encryption/signing and decryption/authentication happens in Linux userspace at the moment the credentials are consumed by a specific service, not earlier, in particular not in UEFI mode
- Suitable for security sensitive stuff, such as PEM secret keys, trusted certificate database information, passwords, and more

# TPM2 PCR Measurements

- Now that kernel and initrd are pre-generated, they are also deterministic
- Hashes for them can be pre-calculated by vendors
- TPM2 PCRs hence too (if we start from a clean PCR)
- (Reminder: PCRs are special registers of the TPM that basically carry hashes of various components of the boot process, and that cannot be reset except via reboots)
- Resulting PCR values that can be pre-calculated can then also be signed by vendor
- Such a PCR signature can be used to build TPM security policies that allow locking secrets to kernels for which a suitable PCR signature can be presented.

# TPM2 PCR Measurements (continued)

- systemd-stub will measure kernel + initrd into PCR 11
- PCR 11 is otherwise unused on Linux systems, hence initially all zeroes
- systemd-measure tool can pre-calculate expected values for PCR 11 from unified kernel components
- It can also sign the resulting expected PCR 11 values
- This PCR value signature can also be included in the PE unified kernel image like the other sections.
- systemd-stub looks for this PCR signature in the PE sections, and generates an on-the-fly cpio initrd archive from it and passes it to the kernel, so that it appears in /.extra/tpm2-pcr-signature.json

# TPM2 PCR Measurements (continued)

- systemd-cryptsetup/systemd-cryptenroll look for this TPM2 PCR signature file, and if it exists automatically lock and unlock volumes with it.
- Similar, credentials encrypted/decrypted via the systemd-creds tool will also use this PCR signature information, to ensure encrypted credentials can only be unlocked if running on a kernel from the same vendor

# Results

- Everything is authenticated: kernel, initrd, extension images, credentials 🌟
- Confidential data is encrypted: credentials ✨
- Disk and credential encryption policies can be bound to kernel vendor, installation and TPM hardware → permitting safe encryption for unattended systems; comprehensive offline security 💫
- Robust updates as kernel images are single-file updates in the ESP 🎉
- Modularity through system extensions 🥳
- Reasonable factory reset: remove credential files from ESP (and possibly extension files) and system is back in pristine state 🎊
- Sane remote attestation possible covering kernel, initrd, system extension images and credentials 🎇

# Outlook

- Asymmetrically encrypted credentials
- Confidential Computing functionality

EOF