Extending EFI support in Linux to new architectures

Ard Biesheuvel <ardb@google.com>
Ilias Apalodimas <ilias.apalodimas@linaro.org>

EFI-related topics in Dublin this week

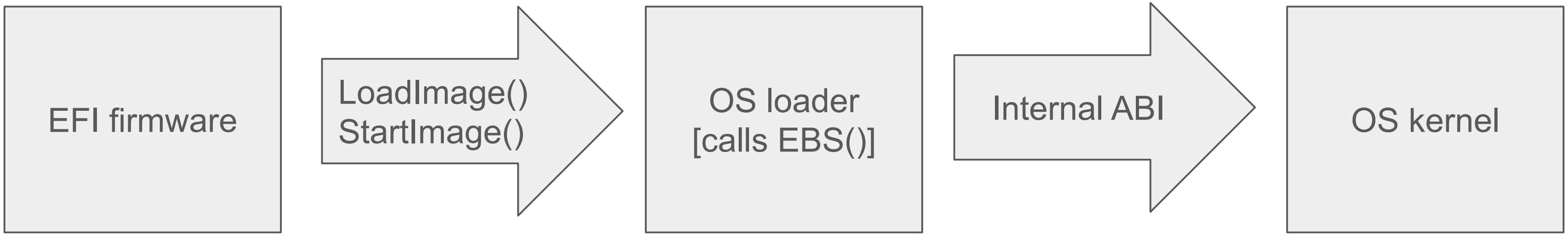
KVM forum:

- TDX, SEV-SNP, conditional attestation (Intel, AMD, IBM)

Plumbers

- Confidential computing MC
- System boot and security MC
- Service Management and systemd MC

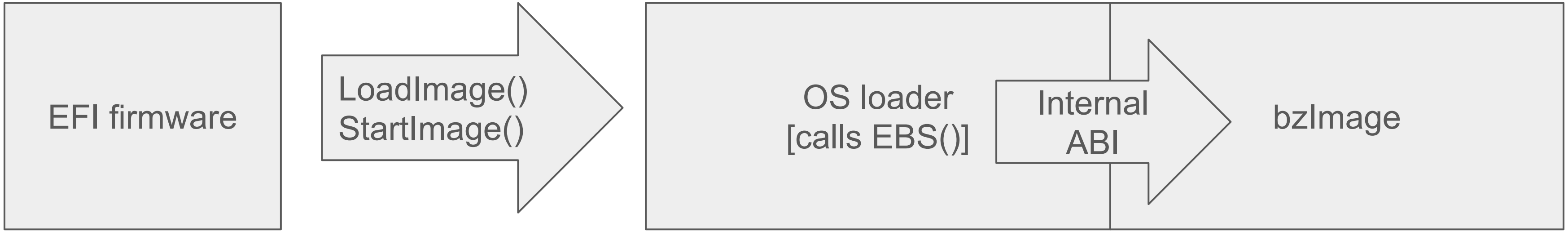
EFI boot flow according to the UEFI specification



EFI firmware exposes boot services used by the OS loader

- LoadImage() parses the PE/COFF headers and constructs the image in memory, zeroes BSS, remaps code and data sections, performs I-cache maintenance, verifies the signature and measures it into the TPM.
- StartImage() invokes the loaded image
- ExitBootServices() [EBS] shuts down all EFI boot services before jumping to the OS kernel

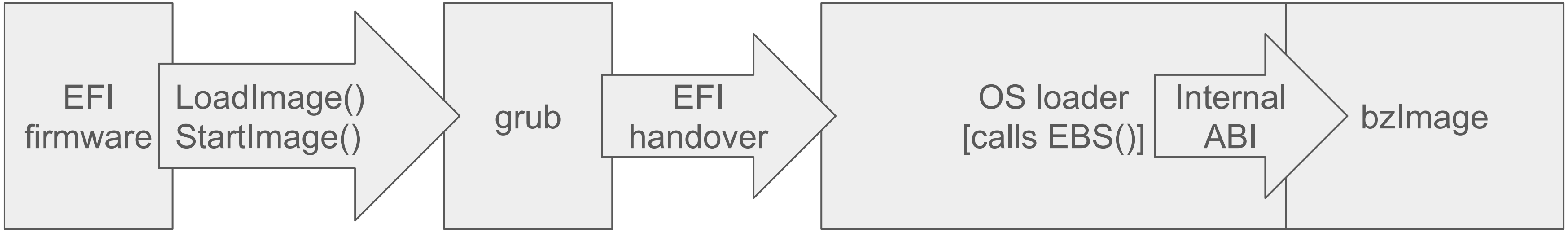
EFI boot flow on Linux with the EFI stub (x86 version)



The EFI stub is an OS loader built into the OS loadable image

- Removes the need for managing the internal ABI between the OS loader and loadee
- OS loader constructs boot_params/setup_data and passes them to the bzImage just like a legacy loader would
- Initrd can only be loaded from the same EFI ‘volume’ from which the OS loader/bzImage was loaded

EFI boot flow on Linux with EFI handover protocol



The EFI handover protocol permits a loader to pass bootparams/setup_data directly

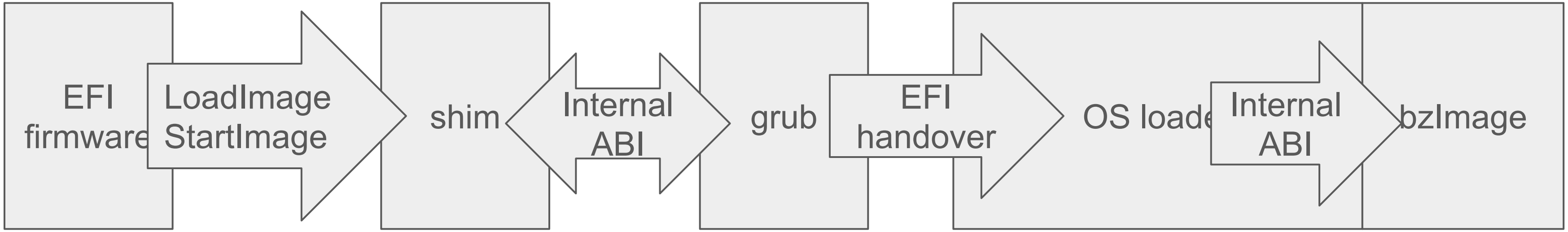
- GRUB can pass the initrd and command line in memory, but needs to understand Linux/x86 specific internals to decide where to load them and how to pass the info
- LoadImage() may be used, but StartImage() is never called, and so the Exit() boot service no longer works
- bootparams/setup_data only exist on x86
- Placement rules for command line and initrd are not the same across architectures

Interlude: the shim ~~problem~~ solution

Problem:

- X86 PC vendors don’t care about EFI compliance, they only care about the Windows sticker
- The easiest way to comply with the MS requirements regarding secure boot is to hardcode the MS certificates and to hardcode secure boot to ON, even if doing so violates the EFI specification.
- MS willing to collaborate on a solution by signing the first loader stage as long as it was not built from GPLv3 sources
- Solution: introduce shim as an intermediate boot stage

EFI boot flow on Linux with shim (distro version)



Shim is signed by MS, and authenticates subsequent stages against its own set of hashes and certificates

- The firmware’s LoadImage() may throw an error, so it cannot be used
- Instead, loading and starting an image is reimplemented from scratch
    - PE/COFF parsing, BSS zeroing, I-cache maintenance, etc need to be reimplemented too
- Shim exposes protocols that grub uses to load and authenticate the kernel
- Result: distro grub doesn’t work without shim, even when shim is unneeded

Porting EFI boot to other architectures

- Replace EFI handover protocol with LoadFile2 based initrd loading from the stub
  - EFI stub allocates the memory for the initrd, and invokes a callback provided by the firmware to load the initrd contents into it. (Recent kernels will then measure it into PCR9)
  - No need to pass bootparams/setup_data so Loadimage/StartImage can be used as before
- Measure the initrd and command line into the TPM from the EFI stub
  - Missing command line is an oversight in the TCG spec, initrd is Linux specific
  - No need to rely on GRUB or other boot stages
- Implement generic EFI zboot that uses LoadImage/StartImage
  - No need to set up bare metal execution context on each architecture
- Make shim truly optional
  - Hook LoadImage() and StartImage(), so subsequent stages don’t need to know the difference

Hooking LoadImage/StartImage has no basis in the spec

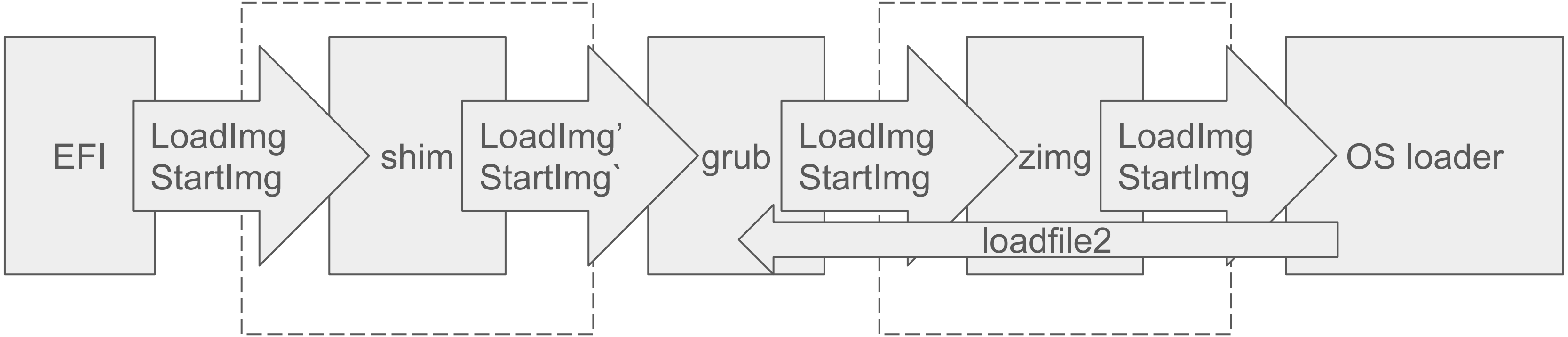
There are two ways for an OS loader to load and execute code:

- Use the LoadImage and StartImage abstractions as provided by the firmware
- Call ExitBootServices() and take ownership of the platform, MMU, page tables etc

Other variants turn out to be problematic:

- Recent x86 firmwares don’t map EFI page allocations as RWX
- Copying code into a fresh allocation and branching into it may not work
  - https://lore.kernel.org/all/cover.1662459668.git.baskov@ispras.ru/ addresses one such problem
  - EFI_MEMORY_ATTTRIBUTE_PROTOCOL added in UEFI v2.10 to manipulate permission attributes

Generic EFI boot flow on Linux



Provide a x86 legacy free EFI boot path for other architectures

- Rely only on EFI abstractions such as boot services and protocols
- Make shim optional and permit subsequent loaders to operate regardless of whether shim is there or not
- Provide optional decompressor stage that is fully generic
- Permit grub to be replaced with systemd-boot or U-Boot seamlessly