UEFI Secure boot in Linux

Sumeet Pawnikar

Who am I ?



Sumeet Pawnikar

- Linux enthusiast
- Experience:
  - Device drivers, Kernel and BSP
  - Low level Firmware/Bootloader
  - Active contributor in Linux Mainline Kernel and device driver development
  - Maintainer of Coreboot Firmware for X86 Thermal/Power Management subsystem
  - Patent holder
- Interest: Open source projects
- Company: Cisco, Ex-Intel, Ex-Samsung
- Hobby: Travelling and Badminton

Agenda

- UEFI Secure Boot Components
- Types of Keys and Database
- Kernel Dependencies and Configuration
- Binary Signing Tools and methodology
- Sign the UEFI binaries and Custom kernel with keys
- Enable Secure boot on Virtual Machine
- Verify Secure boot on the System

Secure Boot

Why Secure boot required in today's world?

- Ensure only authorized boot binaries load

Importance of Secure boot

- Malicious programs hide during boot process, runs before OS fully up and running
- Hard to detect and harder to remove

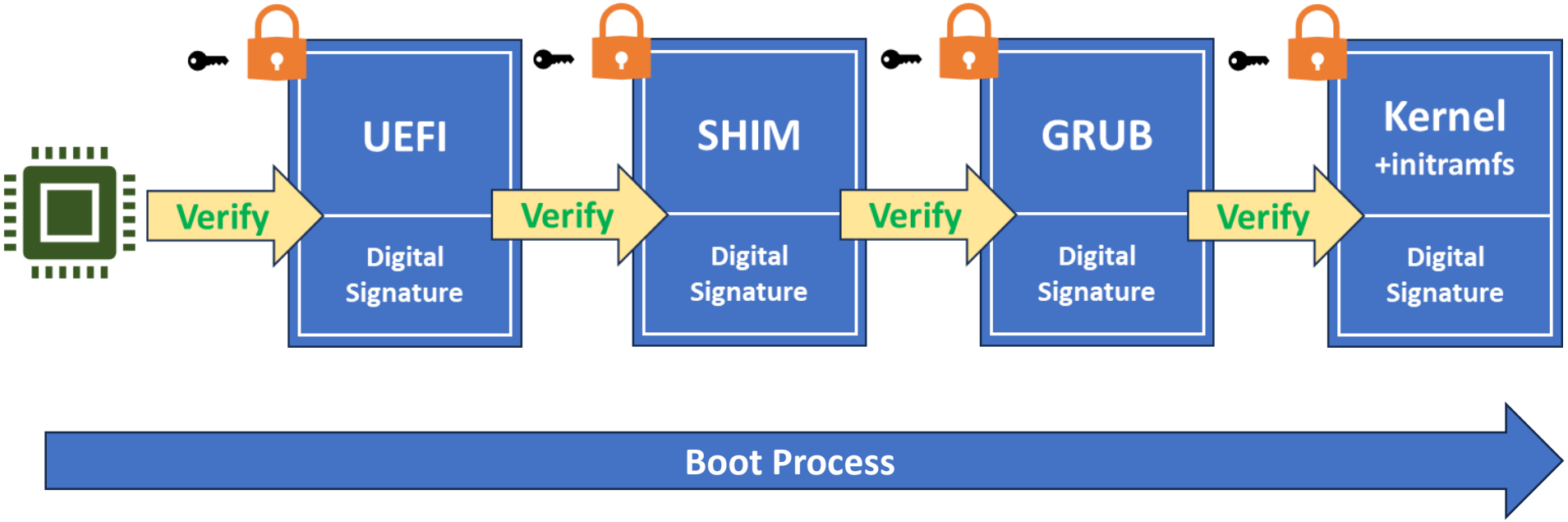
With Secure Boot

- Bootkits and rootkits (malwares) gets blocked from loading by the signature check
- Tampered or unauthorized programs prevented from affecting the boot process
- Users are alerted if something is wrong, so they can address potential issues before it become serious problems

Secure Boot

- **A Security Standard**
- **Boot process Safe**
  - First instruction to full OS load
- **Unified Extensible Firmware Interface (UEFI)**
  - Replacement of BIOS
- **Establish a root of trust**
  - Starts from the Hardware and next Software on the system
- **Trusted Boot Chain**
  - First code executed on hardware platforms is authentic and untampered
- **Foundation of establishing a full chain of trust**
  - Each Software Component validated by it’s previous stage with different Cryptographic keys

Chain of Trust



Types of Keys and Database

- **Keys and Database for Authentication process**
  - **Platform Key (PK):**
    - Master Key/certificate of root of trust for an entire platform
    - Generated by Vendors, Device Makers, OEMs
  - **Key Exchange Key (KEK):**
    - ESL (EFI Signature List) set of certificates
    - Verify the signature of entries in the database (db, dbx)
  - **DataBase (DB):**
    - Approved certificates list for software binary
  - **Blacklist DataBase (DBX):**
    - Unsafe software binary certificates list

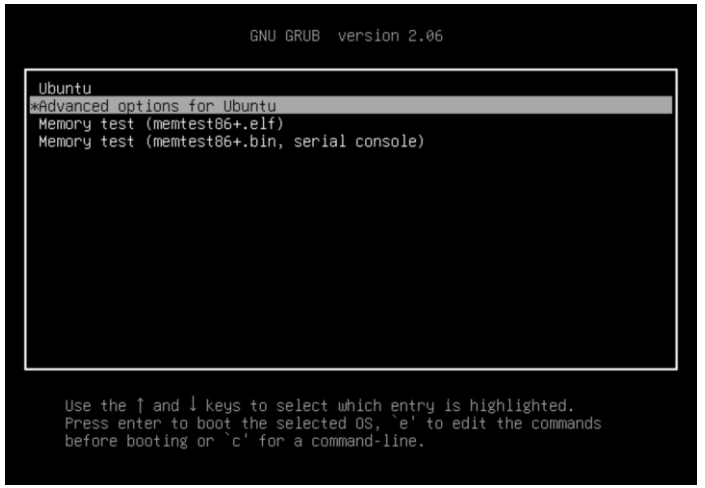
UEFI

- **Unified Extensible Firmware Interface (UEFI)**
  - Replacement of the traditional BIOS
  - POST : Power On Self Test
  - Boot order check
  - Scans hard disk images for partitions containing bootloader code
  - Dynamically find the hardware components and Initializes
- Enforces Secure Boot Policy
- Checks signatures of EFI binaries
- Loads and Verifies the Signature of SHIM [shim*.efi]

SHIM

- **Software Identity Management System**
- First stage bootloader [shim*.efi]
  - Signed by Microsoft and/or Linux Distribution’s Trusted Key [e.g. Ubuntu, Fedora, etc]
  - Bridge between UEFI and GRUB
  - Loads and Verifies the key/signature of GRUB
  - Pass control to GRUB on successful verification
- Builds Secure boot compatible Trust Chain
- Chain-load any EFI binary
- Enables Linux Distros to function seamlessly with UEFI Secure boot process
- Provides a secure and verifiable boot path for OS

GRUB



- **GRand Unified Bootloader**
- Second-stage bootloader [grubx64.efi]
- Signed by Distribution's Key (trusted by Shim)
- Once GRUB is verified and loaded by Shim,
- GRUB is responsible for:
  - Read configuration file (grub.cfg) for boot menu, kernel parameters
  - Locate the kernel and initial RAM disk (initramfs) on the file system
  - Verify the signatures of the linux kernel (`vmlinuz`) and Initial RAM Disk (`initramfs`)
  - Load the kernel and initramfs into memory
  - Pass control to the kernel

Linux Kernel

- **Kernel**
  - Core of the Operating System
  - Loaded, verified and executed by GRUB
  - Initializes the hardware and peripherals
  - Mounts the root filesystem
  - Starts init or systemd – first process
- **List of kernel/driver files for UEFI and Secure boot**
  - /drivers/firmware/efi/efi.c
    - Registers /sys/firmware/efi{,/efivars} when EFI is supported
    - Allows the efivarfs to be mounted or the efivars module to be loaded
    - /sys/firmware/efi used by user-space to determine, the system supports EFI
  - drivers/firmware/efi/libstub/x86-stub.c
  - arch/x86/boot/header.S
- If system is booted with EFI, registers EFI subsystem with firmware subsystem and efivars subsystem with EFI subsystem

Kernel Config

Kernel dependencies for UEFI

Kernel Configs:

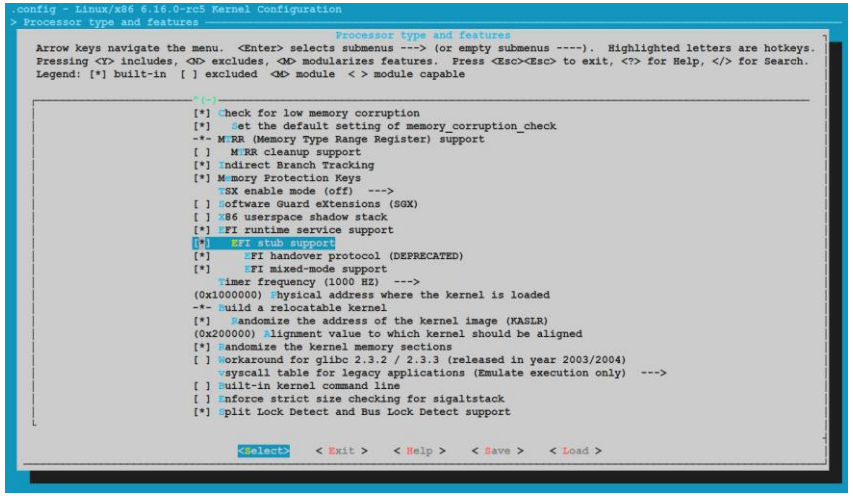
- CONFIG_EFI
- CONFIG_EFI_STUB

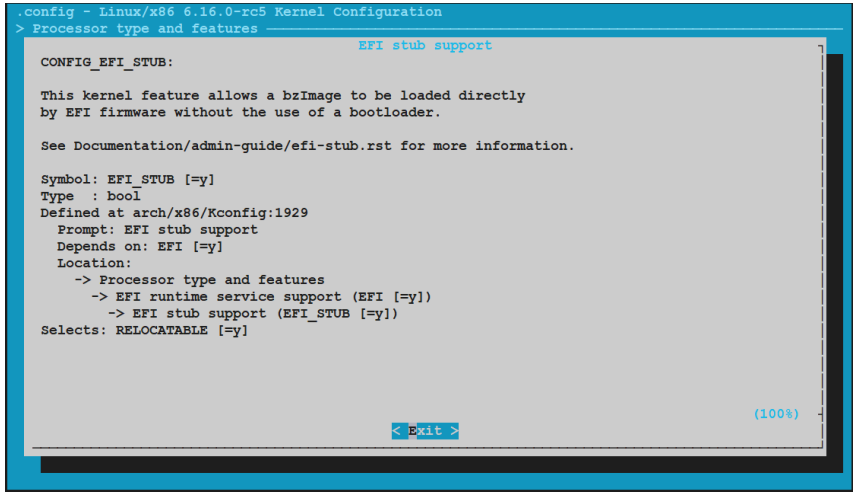
CONFIG_EFI : EFI runtime service support

- Enables usage of efivars (EFI Variables)
- Only useful on EFI firmware supported systems

CONFIG_EFI_STUB : EFI stub support

- Allows a bzImage to be loaded directly
- EFI firmware without the use of a bootloader





Signing Tools

Tools:

- OpenSSL :
  - Create private keys (.key), certificates (.crt/.pem)
  - Convert into EFI signature list format (.esl) and Authenticate update file (.auth) used with the SHIM OpenSSL engine to sign EFI binaries
- Mokutil :
  - MOK(Machine Owner Keys)
  - Add own MOKs in UEFI
- Sbsigntool :
  - Sign EFI binaries
  - Attach a digital signature using a private key and corresponding certificate

Binaries To Sign:

- GRUB image
- Kernel and initrd image

Generate Own Keys

Manually signing and updating keys:

- **Generate Own Keys:**
  - Generate Platform Key (PK)
    - openssl req --new -x509 -newkey rsa:2048 -keyout PK.key -out PK.crt -days 3650 -nodes -subj "/CN=MyPlatformKey/“ cert-to-efi-sig-list -g "$(uuidgen)" PK.crt PK.esl sign-efi-sig-list -g "$(uuidgen)" -k PK.key -c PK.crt PK PK.esl PK.auth
  - Generate Key Exchange Key (KEK)
    - openssl req -new -x509 -newkey rsa:2048 -keyout KEK.key -out KEK.crt -days 3650 -nodes -subj "/CN=MyKEK/“ cert-to-efi-sig-list -g "$(uuidgen)" KEK.crt KEK.esl sign-efi-sig-list -g "$(uuidgen)" -k PK.key -c PK.crt KEK KEK.esl KEK.auth
  - Generate Signature Database Key (db)
    - openssl req -new -x509 -newkey rsa:2048 -keyout db.key -out db.crt -days 3650 -nodes -subj "/CN=MyDbKey/“ cert-to-efi-sig-list -g "$(uuidgen)" db.crt db.esl sign-efi-sig-list -g "$(uuidgen)" -k KEK.key -c KEK.crt db db.esl db.auth
- **Enroll keys in UEIF**
    - sudo mokutil --import db.crt
    - sudo mokutil --import KEK.crt
    - sudo mokutil --import PK.crt
  - Get prompt to set a password
  - Need this password during the next boot in the MOK manager to confirm the key enrollment

Create Signed Binaries

Manually signing and updating keys:

- **Sign EFI Binaries (Bootloader and Kernel):**
  - **sbsign** --key ~/**db.key** --cert ~/**db.crt** --output /boot/efi/EFI/GRUB/**grubx64.efi** /boot/efi/EFI/GRUB/grubx64.efi
  - **sbsign** --key ~/**db.key** --cert ~/**db.crt** --output /boot/**vmlinuz-$(uname -r)-signed** /boot/vmlinuz-$(uname -r)
    - **NOTE**: May need to sign any out-of-tree kernel modules if being used (e.g., Graphics drivers, Virtual Machine drivers).
- **Configure Bootloader:**
  - Load signed EFI binary and Kernel image
  - Reboot
  - Enable Secure Boot in UEFI [UEFI/BIOS setup → Security → Secure Boot]
  - MOK manager will appear at boot, prompting you to confirm the key enrollment using the password you set

*This above is only for advance users or Custom Kernels/Users

VM Creation

- **Pre-requisites:**
  - qemu-kvm , libvirt-daemon-system , libvirt-clients , virt-manager
- **Firmware dependency:**
  - OVMF (Open Virtual Machine Firmware) package need to install on x86 Host
    - Enable UEFI support on KVM [QEMU]
- **Create a Virtual Machine on KVM Virtualization:**

1: command-line mode (virt-install command)

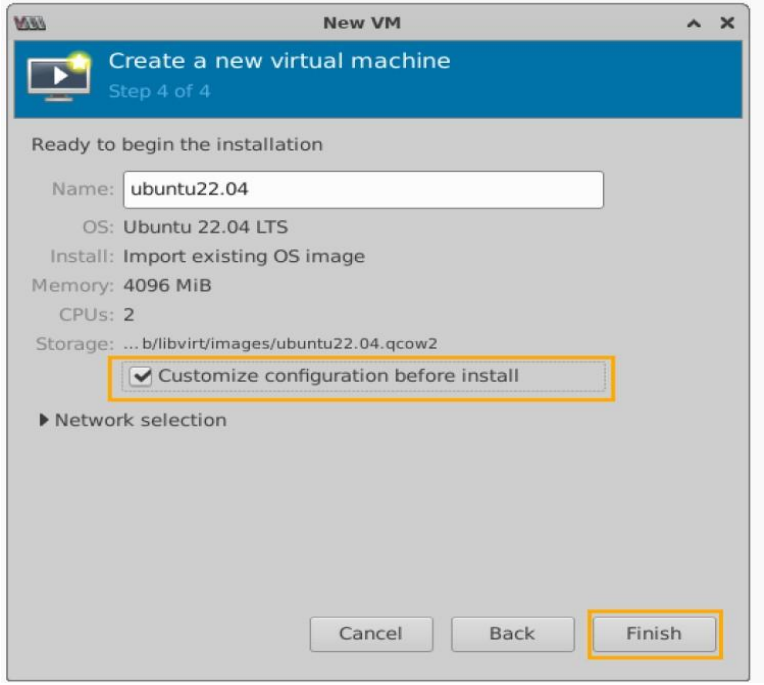
- install ubuntu using virt-install:

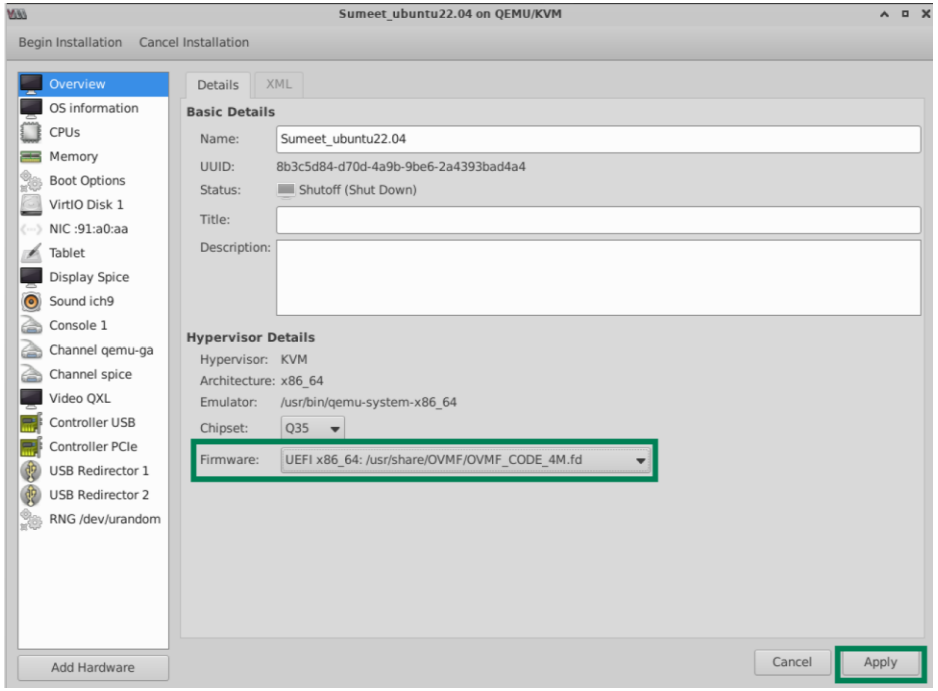
```
$ virt-install --name vm_name --ram 2048 --vcpus 2 \
--disk path=/path/to/vm_image.qcow2,format=qcow2,bus=virtio \
--os-type linux --os-variant ubuntu22.04  --network bridge=virbr0,model=virtio  --graphics spice,listen=0.0.0.0 \
--cdrom /path/to/ubuntu-24.04-live-server-amd64.iso  --boot uefi
```

- Boot VM → UEFI settings → OVMF settings → select image
- virsh start/shutdown/console/list vm_name

GUI Based VM Creation

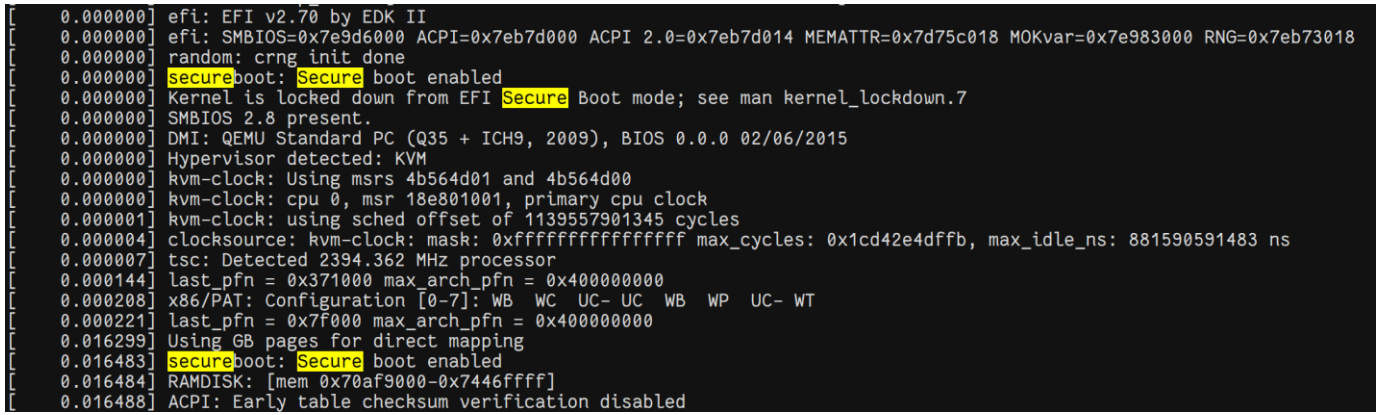
Virtual Machine Manager

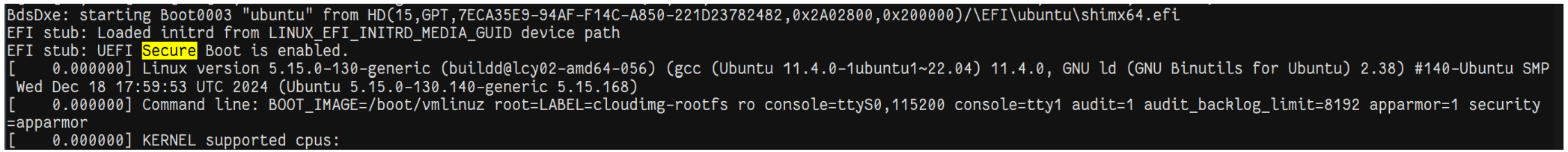




Verify Secure Boot

- **UEFI/BIOS level**
  - UEFI/BIOS setup
    → Security settings
      → Secure Boot
- **Kernel level**
  - **dmesg print**
  - $ sudo dmesg | grep Secure
  - [0.000000] secureboot: Secure boot enabled
  - [0.000000] Kernel is locked down from EFI Secure Boot; see man kernel_lockdown.7
- **System level**
  - **$ mokutil --sb-state**
  - SecureBoot enabled





References

- https://www.gnu.org/software/grub/manual/grub/grub.html#UEFI-secure-boot-and-shim
- https://docs.kernel.org/admin-guide/efi-stub.html
- https://www.rodsbooks.com/efi-bootloaders/controlling-sb.html
- https://www.kernel.org/doc/html/latest/arch/x86/x86_64/uefi.html

Questions ?

