Secure boot on Linux

Dominik Stipić

04-09-2023

1 Motivation

Every good lesson starts with motivation. Why would somebody waste their most important asset - time, in order to study the bootloading topic? Bootloading is the first chain of action which initializes the computing system and it very important to protect it. Usually the systems are usally most sensitive in the beginng of their creation. This statement is true nit only for computing systems but also for society, relationship, child raising and so on. Lets understand bootloading process and protect our computing systems from explotation.

2 Introduction

In this article we want to describe the functioning of the boot loading process which happens on most modern and secure systems. First we will start with definition of the key concepts which we will use in this article.

We can broadly classfiy the system software in 4 categories.

- **Application software**. Application source code or binaries are stored on hard disk. When user wants to run the program the binaries are transfered to the main memory from where they are fetched by the CPU.
- **OS software**. Stored on the hard disk. During bootloading process, the kernel binaries are transfered to the main memory. Operating system is responsible for overall system management and application code is using the OS services.
- **Drivers**. Operates or controls a particular type of device that is attached to a computer. Drivers are also stored on computer's hard drive.
- **Firmware**. They are stored on the device itself. Controls the basic function of the device. The drivers translate the command from the OS into commands that the device can understand. Firmware, on the other hand, controls the basic function of the device and controlls the device low-level operations with low-level languages. Firmware is usually stored on the system *Read-only memory(ROM)* or device flash memory.

Structure, placement and distribution of the described system software can be found on the image 1.

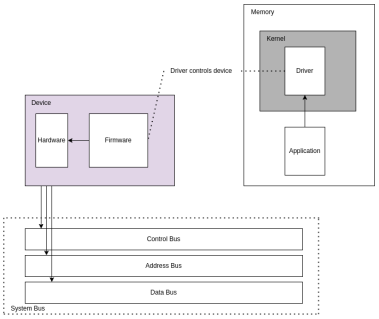

Figure 1: The diagram shows the simplified architecture of the computing system and the placement of the different software types. A foreign device is connected to the computing system on the system bus. The Device contains firmware code that controls the system's I/O peripherals. Inside the main memory, we can find the loaded kernel and application process. In general, the kernel is used for hardware management and it contains hardware drivers that control specific hardware units which are connected to the system bus. An application that the user manages uses the services and libraries exposed by the system OS.

We will also use the security requirements concepts which are usually well established and used in operating system and security community:

- **Identity.** Set of attributes which belong to physical person. Atrributes can be physical, psychological, property and social relationships and so on.
- **Authentification.** The process of verifying the user identity. The user or the candidate who wants to enter into some system needs to pass the process of registration. In the process of registration, the candidate gives to the system representitive unique parts of his identity. In information technology this part of his identity are called *credentials*. When the user wants to use the systems resources he need to present his credentials to the system representitive or owner.
- **Authorization.** System contains the resources which people like to consume. Some people have bigger privileges than other people. When authentificated user wants to consume some resource he needs to pass the process of authorization where the system determines if he is able to consume this resource.
- **Integrity.** This requirement checks if the reosurce is immutable over time. This requirement is usually very important during communication between sender and recipent which are separated by noisy or unsafe channel. This

requirement ensures that recipient can be sure that the message wasn't changed by the attacker.

Booting or bootstraping. The process of initializing the computing system. System initialization starts with the power button press. The goal of the initialization is to load system binaries to the main memory and do software and device quality control (QC) checks. With QC checks, system ensures that hardware works and booting process can move to the next stages. Those series of QC tests are called *power-on self-test(POST)*. Their aim is to verify the hardware components and peripherals and carry out tests to ensure that the computer is in proper working condition.

On image 2 we can see diagram of bootstrapping activities which happen after the power button is pressed. When the button power is pressed the CPU loads and executes the boot loader process which is stored on ROM. Purpose of this code is to redirect the execution thread to the place where bootstapping code is stored. Real bootstrapping code is stored on hard disk partition which is called master boot record partition (MBR). This record contains the bootloading code whose puropose is to initialize the system, load the kernel and kernel modules to the main memory. Information where kernel is located is also contained in the MBR, it is contained in the MBR's partition table. Picture 3 shows the logical organization of the disk. Disk is divided on independent partitions and we can find MBR on the first partition. MBR is located at the very beginning of partitioned computer storage. The MBR code is usaually referred to as a boot loader.

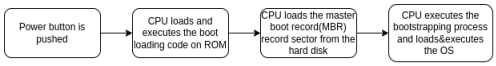

Figure 2: Activity graph which shows the standard booting activites

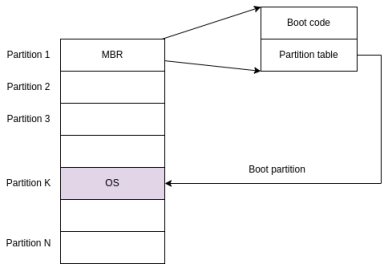

Figure 3: Schema of the hard disk divided on N partitions. The first partition is reserved by master booting record(MBR) which contains booting code and partition table. From partition table, booting process can find out where the system kenel is located. After that is known, the bootloading process loads the kernel to the main memory and start it.

Secure boot. Secure boot is a type of bootloading process that addition to

the booting, it implements software integrity and authentification requirements. This is important because this prevents malicious activities and modification of the pre-boot code. The integrity requirement is implemented by public key cryptography. It is done by verification of the module signatures when loading them into the main memory in the boot process. Let's say that we have an array of binaries B, which are executable code for some kernel module. If we want to sign those binaries, first we need to create a pair of public and private keys, which we will denote with K_d and K_e. We will also need to decide what type of asymmetric cryptographic algorithm and hashing algorithm we will use. Digital signature of binaries B is created as follows:

$$Sign(B) = E(H(B), K_e)$$

In other words, the signature of a message is an encrypted hash of the message content. By verifying this signature with the original message, we are sure that the message hasn't lost its integrity. In our case, the user will sign the module that he wants to connect to the kernel with his private key only once, delete this key and save the public key in the system database so that he can verify module integrity on every system boot. Secure boot reduces the risk of preboot malware attacks such as rootkits.

2.1 Booting process

On the figure **??**, we can see the state diagram which shows the step done when Linux system starts booting process. In practice, there exist two version or orgaizations of the bootloading process: BIOS and UEFI. BIOS uses the, already described, Master Boot Record and UEFI uses GUID Partition table.

Machine Owner Key. Public key which user generates and uses to verify the binaries during bootloading process. The process of signing is used for implementing the message integrity requirement. The seneder of message m signs this message with his private key E_K. The receiver of the message receives the message and performs verfication. The pseudocode of signature and verification functions is as follows:

```
def sign(message, private_key, encrypt=RSA, digest=sha256):
    return (message, encrypt(digest(message), private_key)

def verify(message, signature, public_key, decrypt=RSA.inv, digest=
    sha256):
    h1 = decrypt(signature, public_key)
    h2 = digest(message)
    return h1 == h2
```

Listing 1: Signing and verification functions in Python

When kernel module is created, user needs to sign it only once with private key and he needs to save public key in the Linux MOK database. Every time the system perfoms secure bootloading, the bootloader will check the currently loaded module signature with the module's public key in the MOK. MOKs are

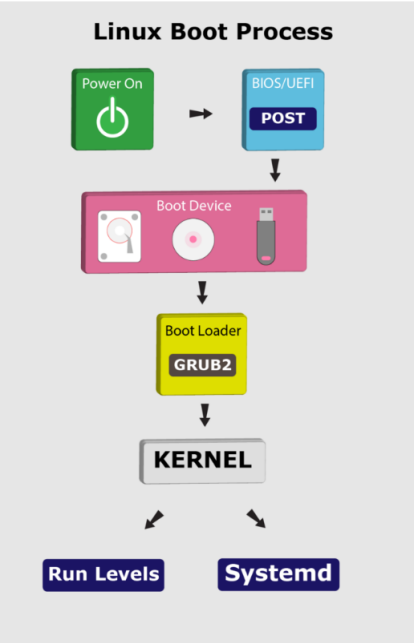

Figure 4:

part of the Secure Boot and they are used to sign EFI binaries. They are managed by mokutil process in order to list, export, add and delete MOKs.

Most of the x86 hardware devices come with the pre-loaded Microsoft keys in their firmware. We are talking about public keys which are loaded

Kernel module. Piece of code that can be loaded and unloaded into the kernel. They extend the functionality of the kernel without the need to reboot the system.

shim.

GRUB = type of bootloader bootloader = transfers Linux kernel and its modules to the RAM and starts its execution

Commands

- **modprobe.** Add the loadable kernel module to the Linux Kernel or to remove it.
- **lsmod**. Lists currently-loaded kernel modules
- **modinfo**. Information about specific module
- **mokutil**. Utility for managing the MOKs. With *mokutil –list-enrolled* we can find out

modprobe module-name: starts the module

2.2 Secure Boot

The goal of the secure boot process is to provide authentification of the Linux kernel modules in order to prevent tempering of the modules. With Secure boot active, the firmware checks the presence of a cryptographic signature on any EFI program that it executes.