The Linux Boot Process

Peter Ashford

Ashford Computer Consulting Service

5/10/2018

What We’ll Cover

This presentation covers the Linux boot process with the Grub2 bootloader. In the addition, we’ll touch on Legacy and UEFI BIOS, MBR and GPT disk partitioning, `/etc/init` and systemd, Secure Boot, CDROM/DVD Boot, USB Boot and Network Boot.

This presentation is Distribution-agnostic.

The intended audience for this presentation is end-users and System Administrators.

What We Won’t Cover

In an effort to keep this presentation down to a reasonable length, some limitations had to be imposed on what would be discussed:

- We won’t cover dual-boot or Windows boot
- We won’t cover the low-level details of many of the steps
- We won’t focus on any one distribution

This presentation isn’t designed for programmers writing BIOS, bootloaders or kernel modules.

Overview

There are several steps in the Linux boot process. These will be addressed individually:

- Legacy BIOS / UEFI
- Master Boot Record / GPT
- Second Stage Bootloader
- Grub2
- Kernel Initialization
- Init / systemd
- Runlevel / default target

Legacy BIOS

The system BIOS performs several tasks during the boot process:

- BIOS is started when the system power stabilizes
- Initializes hardware - CPU(s), memory controller(s) and integrated I/O controllers
- Performs some system integrity checks (POST - Power On Self Test)
- Initialize connected controllers with their own BIOS
- Removes the POST portion of BIOS from the memory map

Legacy BIOS Continued

The system BIOS continues the boot process:

- Checks the BIOS-specified devices for an MBR (Master Boot Record)
- Upon finding an MBR with a partition set to “Active”, the BIOS loads the MBR into memory and gives it control of the system

What is EFI/UEFI

- EFI (Extensible Firmware Interface) replaces the Legacy BIOS
- UEFI (Unified EFI) is EFI 2.0
- EFI 1.x was deprecated in 2005, in favor of UEFI, but manufacturers required several years to change
- UEFI firmware supports a “Legacy BIOS” option for backward compatibility
- UEFI can be extended by the manufacturer with modules for forward compatibility
- UEFI is effectively a mini-OS

Why Use EFI/UEFI

- CPU independent architecture and drivers
  - Supports x86, x86-64, AArch32, AArch64 and Itanium
  - Currently limited to “little-endian” processors, but projects are underway to change that
- Flexible pre-OS environment, including networking
- Can read Bootloader or OS from FAT partitions
- Can validate software signatures
- Required to boot from GPT partition tables
- Required for “Secure Boot”
- Required to advertise Windows 8/10 compatibility

How to Use EFI/UEFI

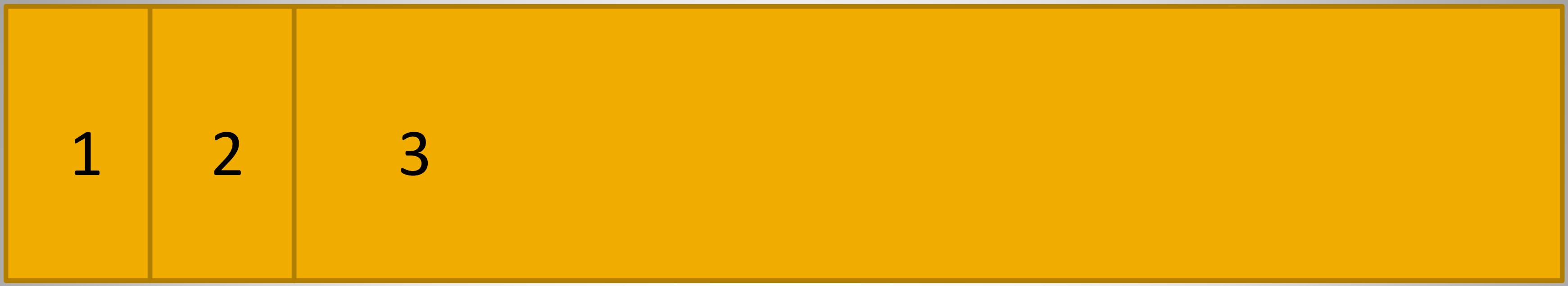
- Configure the system firmware to operate in UEFI mode instead of Legacy BIOS mode (default on systems/motherboards with Windows 8 or Windows 10 certification)
- Configure `CONFIG_EFI_PARTITION` during kernel configuration (most distributions have this by default)
- To boot to the UEFI shell, go through the BIOS and select that option

UEFI BIOS

From the point of view of the Linux kernel, a UEFI BIOS functions similarly to a Legacy BIOS. The exceptions are as follows:

- The UEFI BIOS starts a few Boot Services
- When the kernel boots, these Boot Services are terminated by calling the BIOS "`ExitBootServices`" function
- In a UEFI system, there are additional critical files under "`/boot/efi`" (the BIOS Boot Partition / EFI System Partition)

MBR Disk Layout

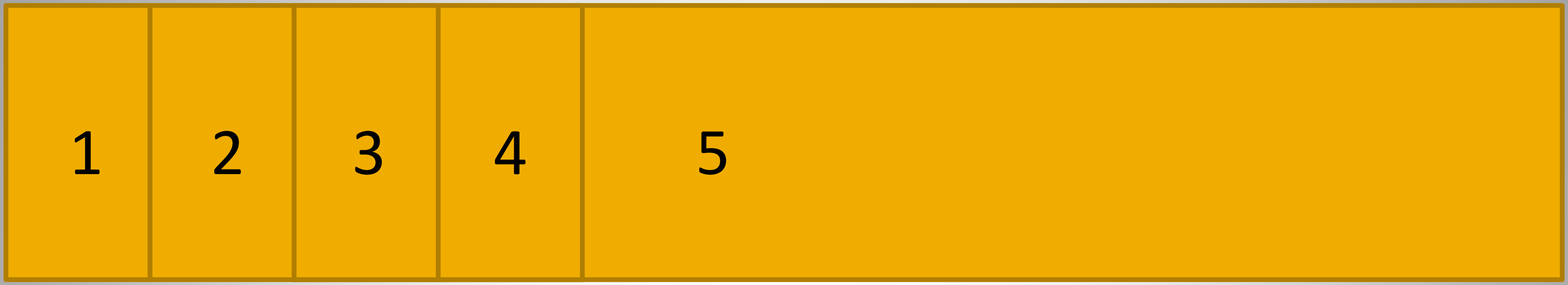


1. MBR - 1 block
2. Reserved space - 62 blocks
3. Partitions - Remainder of disk

Master Boot Record

- The first stage of the Grub2 bootloader is at the start of the MBR, followed by the partition table
- The first-stage bootloader is a program with a maximum size of 446 bytes
- The binary program is installed from "`/boot/grub/boot.img`"
- The first stage bootloader uses BIOS calls to read the second stage bootloader into memory and gives it control of the system

GPT Disk Layout



1. Protective / Hybrid MBR - 1 block
2. Partition table header - 1 block
3. Partition entries - 32 blocks
4. BIOS Boot Partition / EFI System Partition - As required (usually ends at 1MB on disk)
5. Other partitions - Remainder of disk

GPT Disk Layout

- Protective / Hybrid MBR
  - Stage 1 bootloader (not used)
  - Entire disk (up to 2TB) in one partition of type 0xEE
- Partition entries
  - Up to 128 partition entries of 128 bytes each
- BIOS Boot Partition / EFI System Partition
  - FAT partition
  - Contains bootloader
  - Usually first partition
  - Usually ends at 1MB to optimize alignment

Second Stage Bootloader

- The second stage bootloader occupies the 62 blocks between the MBR and the first partition, giving it a maximum size of 31,744 bytes
- The program is installed from "`/boot/grub/core.img`".
- The second stage bootloader can read a Linux EXT file-system (and a few others) on a physical volume. It looks for the "`/boot/grub`" directory, loads the Grub2 configuration file and any necessary modules and performs as directed

Grub2

Once the second stage boot loader has loaded the configuration files, it has access to the pluggable modules in the “`/boot/grub`” directory. These pluggable modules are similar to Linux pluggable modules, in that they can dynamically expand the capabilities of a minimal core OS.

Grub2 displays a menu allowing selection of the OS or utility to boot.

Grub2 Continued

- Every menu item has an executable file (a kernel for Linux) associated with it
- There may be parameters associated with the menu item
- Almost every Linux has an InitRamFS parameter associated with it
- Other parameters may be included if needed/desired
- The kernel file is loaded into memory
- The parameters in the Grub configuration are passed to the kernel

Grub2 Continued

- You can break out of the menu by pressing the “c” key, which will give you a Grub2 command prompt
- You can temporarily edit a menu entry by pressing the “e” key while on the entry to be edited

Kernel

When control is passed to the kernel, it performs the following steps:

- Load the InitRamFS file into memory
- Extract/Uncompress itself
- Initialize and configure memory
- Create a "`tmpfs`" file-system
- Copy the InitRamFS file into the tmpfs file-system
- Release the memory containing the initial InitRamFS
- Extract the InitRamFS file into the tmpfs file-system

InitRamFS

- The InitRamFS is a compressed (usually gzip) CPIO file containing the modules and files needed to get the kernel to the point where it can use the ROOT file-system
- The InitRamFS usually includes a BusyBox instance for debugging boot problems
- The contents of the InitRamFS file are controlled by the "`/etc/initramfs-tools/initramfs.conf`" file

Back to the Kernel

- After the InitRamFS has been extracted, the kernel probes the hardware to see what’s attached to the system
- Any device that has a module in the InitRamFS will be initialized and made available
- The kernel then initializes any virtual devices (e.g. LVM and RAID) that have modules in the InitRamFS

Kernel Continued

- Now that the devices are ready to communicate, the kernel looks for the ROOT file-system that was specified in the boot command (from Grub2) and mounts it read-only
- The change of ROOT is called a “pivot”
- After switching to the real ROOT file-system, the tmpfs file-system containing the InitRamFS is no longer needed and is released

Kernel Continued

- At this point, the kernel has access to all the drivers
- If any hardware wasn’t able to be initialized previously, the driver is now loaded and the hardware is initialized
- The ROOT file-system is now remounted read/write, and the system manager process is executed
- The system manager can be either systemd or legacy init

Legacy Init

The legacy init software brings the system to the run state specified in the `initdefault` line of the `/etc/inittab` file, using the scripts in the `/etc/rc?.d` directories, which are links to scripts in `/etc/init.d`.

Systemd

Systemd brings the system to the state defined by the default target.

```
systemctl get-default
```

Secure Boot Concepts

- Secure boot requires a UEFI BIOS and a GPT disk
- Secure boot uses signed binaries to establish a chain of trust via x.509 certificates
- All PC hardware manufacturers who support Secure Boot accept Microsoft's signing key (part of Windows 10 certification), but very few will accept other keys
- Not all Linux Distributions are available with Secure Boot
- Depending on the Distribution, Secure Boot might be 64-bit only or 32/64-bit

Secure Boot - First Stage

- The first stage bootloader is a “shim” that resides in the BIOS Boot Partition
- Due to the signature and the need to verify other signatures, the shim is far too large to fit in the MBR
- The UEFI BIOS verifies that the shim is signed with an accepted KEK (Key Exchange Key), which, in practice, requires that the shim be signed by Microsoft (about $100)
- The shim is very stable, and only needs to be resigned when the cert expires

Secure Boot - Shim

- The keychain includes the KEK (from the firmware) and any keys added during the boot process (transient - stored in RAM)
- The shim adds a distribution-specific signing key and optional MOKs (Machine Owner Key) to the keychain
- The MBR may still contain a normal first stage bootloader

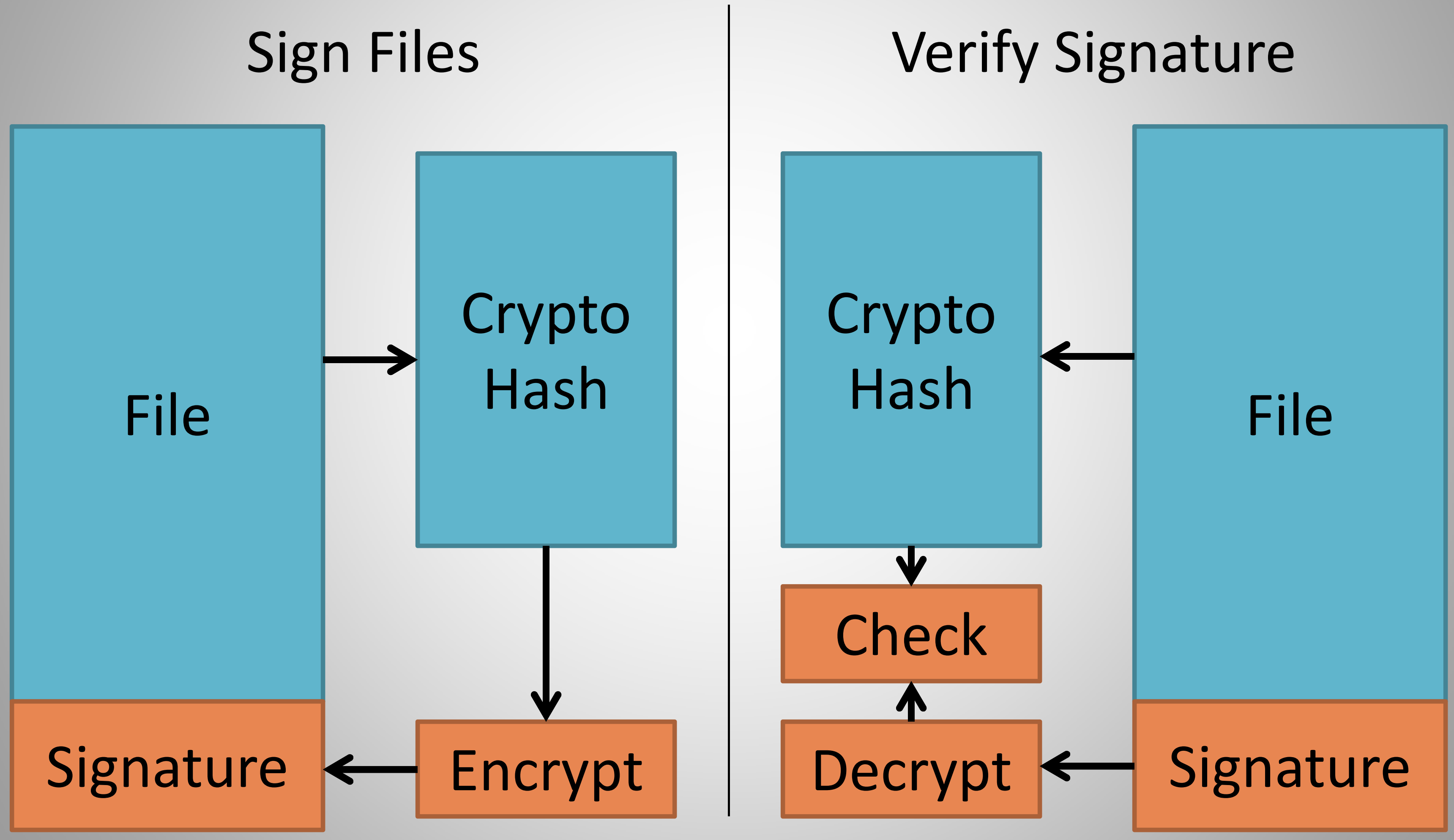
Secure Boot - Second Stage

- The second stage bootloader (Grub2) resides in the BIOS Boot Partition, and must be signed
- The kernel must be signed
- Signatures are checked against the keyring by requesting verification from the UEFI BIOS

Secure Boot - Kernel

- The InitRamFS doesn’t need to be signed
- All modules loaded by the kernel must be signed, whether from the InitRamFS or from the ROOT partition
- Signatures are checked against the keyring by calling the UEFI BIOS
- This is the end of the trust chain

Signed Files



System State

- Was system booted through UEFI?
  - ls /sys/firmware/efi
- Was Secure Boot used to boot the system?
  - mokutil --sb-state
- How do you sign a custom module?
  - Generate a MOK
  - Install the MOK
  - Sign the modules with the MOK
  - Details at: https://www.suse.com/documentation/sled11/book_sle_admin/data/sec_uefi_secboot.html

Summary

MBR	GPT	Secure Boot
Load/Run Boot Block		Load/Check/Run Shim
Load/Run 2nd-stage Bootloader	Load/Run Grub2	Load/Check/Run Grub2
Load Grub2 Configuration/Modules	Load Grub2 Configuration/Modules	Load/Check Grub2 Configuration/Modules
Load/Run Kernel/InitRamFS	Load/Run Kernel/InitRamFS	Load/Check/Run Kernel/InitRamFS
Load Kernel Modules	Load Kernel Modules	Load/Check Kernel Modules
Run System Manager	Run System Manager	Run System Manager

References (Distro Specific)

- https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Kernel_Administration_Guide/sect-signing-kernel-modules-for-secure-boot.html
- https://docs-old.fedoraproject.org/en-US/Fedora/18/html/UEFI_Secure_Boot_Guide/
- https://wiki.debian.org/SecureBoot (doesn’t work yet)
- https://wiki.ubuntu.com/UEFI/SecureBoot
- https://www.suse.com/documentation/sled11/book_sle_admin/data/sec_uefi_secboot.html

CDROM/DVD Drive Boot

Bootable CD and DVD drives are formatted according to the “El Torito” specification. This is an enhancement to the ISO 9660 format.

CDROM/DVD sectors are 2048 bytes long.

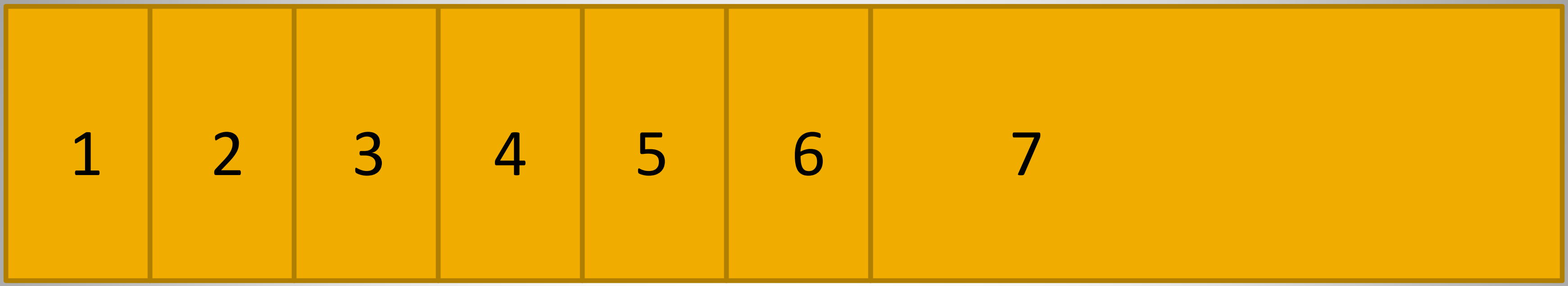
CD/DVD Drive Boot

The BIOS does the following to boot from optical media:

- Read and verify Primary Volume Descriptor
- Read and verify Boot Volume Descriptor
- Read and verify Boot Catalog
- Read and boot from the selected (or default) disk image

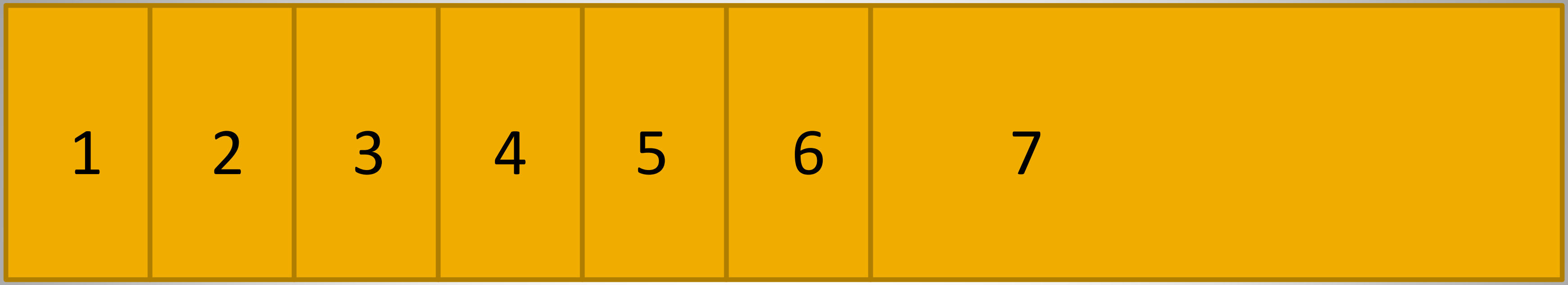
Disk image can be a floppy image (1.2MB, 1.44MB or 2.88MB) or a disk image (any size, 1 MBR partition).

El Torito Layout



1. System reserved (unused) - 16 sectors
2. Primary Volume Descriptor- 1 sector
3. Boot Volume Descriptor - 1 sector
4. Other Volume Descriptors - 1 sector per Volume
5. Volume Descriptor Set Terminator - 1 sector

El Torito Layout



6. Boot Catalog containing Boot Record(s) - as required
7. Disk Images - as required

CD/DVD Drive Boot

- Due to the size of the Linux boot files (kernel and InitRamFS), floppy images are not used
- To boot using legacy BIOS, the boot image is usually an EXT file-system
- To boot using UEFI, the disk image is usually GPT-partitioned

It is likely that Secure Boot will have to be disabled during installation, even on those distributions that support Secure Boot.

USB Flash Drive Boot

When a USB flash drive is used to boot a computer, the BIOS causes it to be seen as either a disk drive or an optical drive. This allows a bootable USB drive to be formatted as either a disk drive (MBR or GPT) or as an optical drive ("El Torito").

Because of this, it is sometimes possible to use the "`dd`" command to copy a bootable disk or `.ISO` file directly to the USB drive and have it boot. This depends on the BIOS and boot software in the image.

Network Boot

A network boot uses the PXE (Preboot eXecution Environment) software in the NIC (Network Interface Controller), working in coordination with the BIOS and network services.

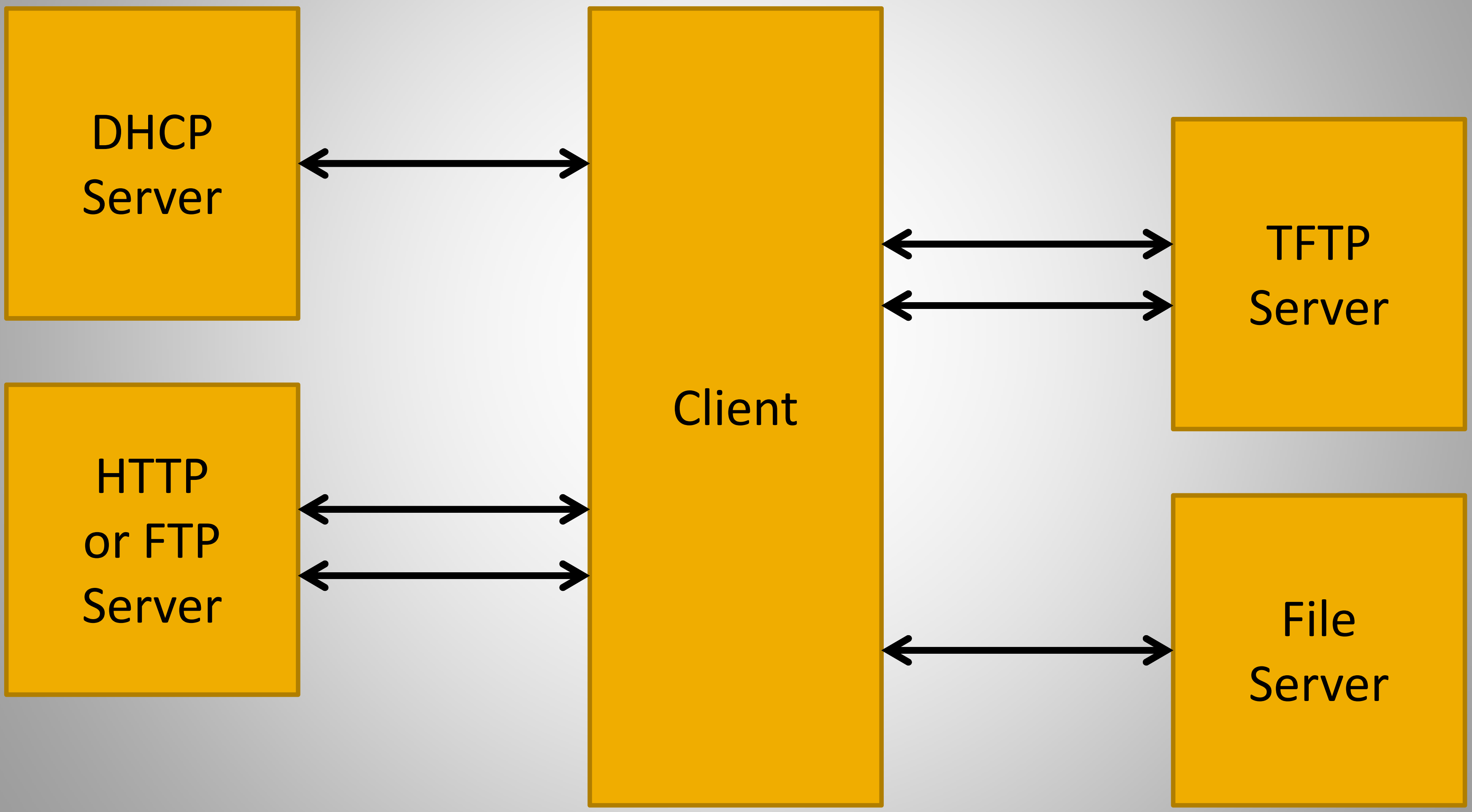
Network Boot

- DHCP is used to get the IP address and parameters
  - `filename` (option 67) contains the file to boot from
  - `next-server` (option 66) contains the IP address of the TFTP server
  - Some configurations will require additional parameters
  - Some distributions may require additional parameters
- TFTP is used to download the file (`pxelinux.0`)
- pxelinux.0 uses TFTP to download configuration file
  - Configuration file generates a menu (like Grub1)
  - Configuration file points to kernels, InitRamFS, ROOT and provides optional parameters

Network Boot

- The pxelinux.0 configuration file is in the "`pxelinux.cfg`" directory of the TFTP server
- In order to simplify configuration functions, the following names are checked, in order:
  - Client UUID - e.g. `b8945908-d6a6-41a9-611d-74a6ab80b83d`
  - Client NIC MAC address, prefixed by the interface type - e.g. `01-88-99-aa-bb-cc-dd`
  - Client IP address in upper-case Hex - e.g. `C0A8025B`, `C0A8025`, `C0A802`, `C0A80`, `C0A8`, `C0A`, `C0`, `C`
  - "`default`"

Network Boot



Network Boot

For EFI/UEFI, the boot file changes from `pxelinux.0` to `syslinux.efi`.

At this time, network boot doesn't support Secure Boot.

References

- http://www.rodsbooks.com/efi-bootloaders/secureboot.html
- http://www.rodsbooks.com/efi-bootloaders/controlling-sb.html
- https://pdos.csail.mit.edu/6.828/2014/readings/boot-cdrom.pdf
- https://wiki.osdev.org/El-Torito

Feedback

My goal is to improve this presentation. To that end, I would appreciate feedback to:

- ashford@accs.com
- www.linkedin.com/in/peterashford