OpenCore

Reference Manual (0.9.4.5)

[2023.09.10]

Failsafe: `false`
Description: Protect UEFI services from being overridden by the firmware.

Some modern firmware, including on virtual machines such as VMware, may update pointers to UEFI services during driver loading and related actions. Consequently, this directly obstructs other quirks that affect memory management, such as `DevirtualiseMmio`, `ProtectMemoryRegions`, or `RebuildAppleMemoryMap`, and may also obstruct other quirks depending on the scope of such.

GRUB ~~shim~~ Shim makes similar on-the-fly changes to various UEFI image services, which are also protected against by this quirk.

Note 1: On VMware, the need for this quirk may be determined by the appearance of the "Your Mac OS guest might run unreliably with more than one virtual core." message.

Note 2: This quirk is needed for correct operation if OpenCore is chainloaded from GRUB with BIOS Secure Boot enabled.

14. `ProvideCustomSlide`
Type: `plist boolean`
Failsafe: `false`
Description: Provide custom KASLR slide on low memory.

This option performs memory map analysis of the firmware and checks whether all slides (from `1` to `255`) can be used. As `boot.efi` generates this value randomly with `rdrand` or pseudo randomly `rdtsc`, there is a chance of boot failure when it chooses a conflicting slide. In cases where potential conflicts exist, this option forces macOS to select a pseudo random value from the available values. This also ensures that the `slide=` argument is never passed to the operating system (for security reasons).

Note: The need for this quirk is determined by the `OCABC: Only N/256 slide values are usable!` message in the debug log.

15. `ProvideMaxSlide`
Type: `plist integer`
Failsafe: `0`
Description: Provide maximum KASLR slide when higher ones are unavailable.

This option overrides the maximum slide of 255 by a user specified value between 1 and 254 (inclusive) when `ProvideCustomSlide` is enabled. It is assumed that modern firmware allocates pool memory from top to bottom, effectively resulting in free memory when slide scanning is used later as temporary memory during kernel loading. When such memory is not available, this option stops the evaluation of higher slides.

Note: The need for this quirk is determined by random boot failures when `ProvideCustomSlide` is enabled and the randomized slide falls into the unavailable range. When `AppleDebug` is enabled, the debug log typically contains messages such as `AAPL: [EB|'LD:LKC] } Err(0x9)`. To find the optimal value, append `slide=X`, where `X` is the slide value, to the `boot-args` and select the largest one that does not result in boot failures.

16. `RebuildAppleMemoryMap`
Type: `plist boolean`
Failsafe: `false`
Description: Generate macOS compatible Memory Map.

The Apple kernel has several limitations on parsing the UEFI memory map:

- The Memory map size must not exceed 4096 bytes as the Apple kernel maps it as a single 4K page. As some types of firmware can have very large memory maps, potentially over 100 entries, the Apple kernel will crash on boot.
- The Memory attributes table is ignored. `EfiRuntimeServicesCode` memory statically gets `RX` permissions while all other memory types get `RW` permissions. As some firmware drivers may write to global variables at runtime, the Apple kernel will crash at calling UEFI runtime services unless the driver `.data` section has a `EfiRuntimeServicesData` type.

To workaround these limitations, this quirk applies memory attribute table permissions to the memory map passed to the Apple kernel and optionally attempts to unify contiguous slots of similar types if the resulting memory map exceeds 4 KB.

they have no memory. Using the `non-volatile` flag will cause the log to be written to NVRAM flash after every printed line.

To obtain UEFI variable logs, use the following command in macOS:

```
nvram 4D1FDA02-38C7-4A6A-9CC6-4BCCA8B30102:boot-log |
  awk '{gsub(/%0d%0a%00/,"");gsub(/%0d%0a/,"\n")}1'
```

Warning 1: Certain firmware appear to have defective NVRAM garbage collection. As a result, they may not be able to always free space after variable deletion. Do not enable `non-volatile` NVRAM logging on such devices unless specifically required.

While the OpenCore boot log already contains basic version information including build type and date, this information may also be found in the `opencore-version` NVRAM variable even when boot logging is disabled.

File logging will create a file named `opencore-YYYY-MM-DD-HHMMSS.txt` (in UTC) under the EFI volume root with log contents (the upper case letter sequence is replaced with date and time from the firmware). Please be warned that some file system drivers present in firmware are not reliable and may corrupt data when writing files through UEFI. Log writing is attempted in the safest manner and thus, is very slow. Ensure that `DisableWatchDog` is set to `true` when a slow drive is used. Try to avoid frequent use of this option when dealing with flash drives as large I/O amounts may speed up memory wear and render the flash drive unusable quicker.

Warning 2: It is possible to enable fast file logging, which requires a fully compliant firmware FAT32 driver. On drivers with incorrect FAT32 write support (e.g. APTIO IV, but maybe others) this setting can result in corruption up to and including an unusable ESP filesystem, therefore be prepared to recreate the ESP partition and all of its contents if testing this option. This option can increase logging speed significantly on some suitable firmware, but may make little speed difference on some others.

When interpreting the log, note that the lines are prefixed with a tag describing the relevant location (module) of the log line allowing better attribution of the line to the functionality.

The list of currently used tags is as follows.

Drivers and tools:

- `BMF` — OpenCanopy, bitmap font
- `BS` — Bootstrap
- `GSTT` — GoptStop
- `HDA` — AudioDxe
- `KKT` — KeyTester
- `LNX` — OpenLinuxBoot
- `MMDD` — MmapDump
- `OCPAVP` — PavpProvision
- `OCRST` — ResetSystem
- `OCUI` — OpenCanopy
- `OC` — OpenCore main, also OcMainLib
- `OLB` — OpenLegacyBoot
- `VMOPT` — VerifyMemOpt

Libraries:

- `AAPL` — OcLogAggregatorLib, Apple EfiBoot logging
- `OCABC` — OcAfterBootCompatLib
- `OCAE` — OcAppleEventLib
- `OCAK` — OcAppleKernelLib
- `OCAU` — OcAudioLib
- `OCA` —- OcAcpiLib
- `OCBP` — OcAppleBootPolicyLib
- `OCB` — OcBootManagementLib
- `OCLBT` — OcBlitLib
- `OCCL` — OcAppleChunkListLib
- `OCCPU` — OcCpuLib
- `OCC` — OcConsoleLib

11 UEFI

11.1 Introduction

UEFI (Unified Extensible Firmware Interface) is a specification that defines a software interface between an operating system and platform firmware. This section allows loading additional UEFI modules as well as applying tweaks to the onboard firmware. To inspect firmware contents, apply modifications and perform upgrades UEFITool and supplementary utilities can be used.

11.2 Drivers

Depending on the firmware, a different set of drivers may be required. Loading an incompatible driver may lead the system to unbootable state or even cause permanent firmware damage. Some of the known drivers are listed below:

`AudioDxe*`	HDA audio support driver in UEFI firmware for most Intel and some other analog audio controllers. Staging driver, refer to acidanthera/bugtracker#740 for known issues in AudioDxe.
`btrfs_x64`	Open source BTRFS file system driver, required for booting with OpenLinuxBoot from a file system which is now quite commonly used with Linux.
`BiosVideo*`	CSM video driver implementing graphics output protocol based on VESA and legacy BIOS interfaces. Used for UEFI firmware with fragile GOP support (e.g. low resolution). Requires `ReconnectGraphicsOnConnect`. Included in OpenDuet out of the box.
`CrScreenshotDxe*`	Screenshot making driver saving images to the root of OpenCore partition (ESP) or any available writeable filesystem upon pressing `F10`. Accepts optional driver argument `--enable-mouse-click` to additionally take screenshot on mouse click. (It is recommended to enable this option only if a keypress would prevent a specific screenshot, and disable it again after use.) This is a modified version of `CrScreenshotDxe` driver by Nikolaj Schlej.
`EnableGop{Direct}*`	Early beta release firmware-embeddable driver providing pre-OpenCore non-native GPU support on MacPro5,1. Installation instructions can be found in the `Utilities/EnableGop` directory of the OpenCore release zip file - proceed with caution.
`ExFatDxe`	Proprietary ExFAT file system driver for Bootcamp support commonly found in Apple firmware. For Sandy Bridge and earlier CPUs, the `ExFatDxeLegacy` driver should be used due to the lack of `RDRAND` instruction support.
`ext4_x64`	Open source EXT4 file system driver, required for booting with OpenLinuxBoot from the file system most commonly used with Linux.
`HfsPlus`	Recommended. Proprietary HFS file system driver with bless support commonly found in Apple firmware. For Sandy Bridge and earlier CPUs, the `HfsPlusLegacy` driver should be used due to the lack of `RDRAND` instruction support.
`HiiDatabase*`	HII services support driver from `MdeModulePkg`. This driver is included in most types of firmware starting with the Ivy Bridge generation. Some applications with GUI, such as UEFI Shell, may need this driver to work properly.
`EnhancedFatDxe`	FAT filesystem driver from `FatPkg`. This driver is embedded in all UEFI firmware and cannot be used from OpenCore. Several types of firmware have defective FAT support implementation that may lead to corrupted filesystems on write attempts. Embedding this driver within the firmware may be required in case writing to the EFI partition is needed during the boot process.
`NvmExpressDxe*`	NVMe support driver from `MdeModulePkg`. This driver is included in most firmware starting with the Broadwell generation. For Haswell and earlier, embedding it within the firmware may be more favourable in case a NVMe SSD drive is installed.
`OpenCanopy*`	OpenCore plugin implementing graphical interface.
`OpenRuntime*`	OpenCore plugin implementing `OC_FIRMWARE_RUNTIME` protocol.
`OpenLegacyBoot*`	OpenCore plugin implementing `OC_BOOT_ENTRY_PROTOCOL` to allow detection and booting of legacy operating systems from OpenCore on Macs, OpenDuet and systems with a CSM.
`OpenLinuxBoot*`	OpenCore plugin implementing `OC_BOOT_ENTRY_PROTOCOL` to allow direct detection and booting of Linux ~~distributiuons~~ distributions from OpenCore, without chainloading via GRUB.

Predefined labels are saved in the `\EFI\OC\Resources\Label` directory. Each label has `.lbl` or `.l2x` suffix to represent the scaling level. Full list of labels is provided below. All labels are mandatory.

- `EFIBoot` — Generic OS.
- `Apple` — Apple OS.
- `AppleRecv` — Apple Recovery OS.
- `AppleTM` — Apple Time Machine.
- `Windows` — Windows.
- `Other` — Custom entry (see `Entries`).
- `ResetNVRAM` — Reset NVRAM system action or tool.
- `SIPDisabled` — Toggle SIP tool with SIP disabled.
- `SIPEnabled` — Toggle SIP tool with SIP enabled.
- `Shell` — Entry with UEFI Shell name (e.g. `OpenShell`).
- `Tool` — Any other tool.

Note: All labels must have a height of exactly 12 px. There is no limit for their width.

Label and icon generation can be performed with bundled utilities: `disklabel` and `icnspack`. Font is Helvetica 12 pt times scale factor.

Font format corresponds to AngelCode binary BMF. While there are many utilities to generate font files, currently it is recommended to use dpFontBaker to generate bitmap font (using CoreText produces best results) and fonverter to export it to binary format.

11.5 OpenRuntime

`OpenRuntime` is an OpenCore plugin implementing `OC_FIRMWARE_RUNTIME` protocol. This protocol implements multiple features required for OpenCore that are otherwise not possible to implement in OpenCore itself as they are needed to work in runtime, i.e. during operating system functioning. Feature highlights:

- NVRAM namespaces, allowing to isolate operating systems from accessing select variables (e.g. `RequestBootVarRouting` or `ProtectSecureBoot`).
- Read-only and write-only NVRAM variables, enhancing the security of OpenCore, Lilu, and Lilu plugins, such as VirtualSMC, which implements `AuthRestart` support.
- NVRAM isolation, allowing to protect all variables from being written from an untrusted operating system (e.g. `DisableVariableWrite`).
- UEFI Runtime Services memory protection management to workaround read-only mapping (e.g. `EnableWriteUnprotector`).

11.6 OpenLegacyBoot

OpenLegacyBoot is an OpenCore plugin implementing `OC_BOOT_ENTRY_PROTOCOL`. It aims to detect and boot legacy installed operating systems.

Usage:

- Add `OpenLegacyBoot.efi` and also optionally (see below) `OpenNtfsDxe.efi` to the `config.plist` `Drivers` section.
- Install Windows or another legacy operating system as normal if this has not been done earlier – OpenLegacyBoot is not involved in this stage and may be unable to boot from installation media such as a USB device.
- Reboot into OpenCore: the installed legacy operating system should appear and boot directly from OpenCore when selected.

OpenLegacyBoot does not require any additional filesystem drivers such as `OpenNtfsDxe.efi` to be loaded for base functionality, but loading them will enable the use of `.contentDetails` and `.VolumeIcon.icns` files for boot entry customisation.

11.6.1 Configuration

No additional configuration should work well in most circumstances, but if required the following options for the driver may be specified in `UEFI/Drivers/Arguments`:

- `--hide-devices` - String value, no default.

  When this option is present and has one or more values separated by semicolons (e.g. `--hide-devices=PciRoot(0x0)/Pci(0x1F,0x2)/Sata(0x0,0xFFFF,0x0)/HD(2,GPT,...)`), it disables scanning the specified disks for legacy operating system boot sectors.

11.7 OpenLinuxBoot

OpenLinuxBoot is an OpenCore plugin implementing `OC_BOOT_ENTRY_PROTOCOL`. It aims to automatically detect and boot most Linux distros without additional configuration.

Usage is as follows:

- Add `OpenLinuxBoot.efi` and also typically (see below) `ext4_x64.efi` to the `config.plist` `Drivers` section.
- Make sure `RequestBootVarRouting` and `LauncherOption` are enabled in `config.plist`; it is also recommended to enable `HideAuxiliary` in order to hide older Linux kernels except when required (they are added as auxiliary entries and so may then be shown by pressing the `Spacebar` key in the OpenCore boot menu).
- Install Linux as normal if this has not been done earlier – OpenLinuxBoot is not involved in this stage.
- Reboot into OpenCore: the installed Linux distribution should just appear and boot directly from OpenCore when selected, which it does without chainloading via GRUB.

If OpenCore has already been manually set up to boot Linux, e.g. via `BlessOverride` or via `Entries` then then these settings may be removed so that the Linux distribution is not displayed twice in the boot menu.

It is recommended to install Linux with its default bootloader, even though this will not be actively used when booting via OpenLinuxBoot. This is because OpenLinuxBoot has to detect the correct kernel options to use, and does so by looking in files left by the default bootloader. If no bootloader was installed (or these options cannot be found) booting is still possible, but the correct boot options must be manually specified before OpenLinuxBoot will attempt to start the distro.

OpenLinuxBoot typically requires filesystem drivers that are not available in firmware, such as EXT4 and BTRFS drivers. These drivers can be obtained from external sources. Drivers tested in basic scenarios can be downloaded from OcBinaryData. Be aware that these drivers are not tested for reliability in all scenarious, nor did they undergo tamper-resistance testing, therefore they may carry potential security or data-loss risks.

Most Linux distros require the `ext4_x64` driver, a few may require the `btrfs_x64` driver, and a few may require no additional file system driver: it depends on the filesystem of the boot partition of the installed distro, and on what filesystems are already supported by the system's firmware. LVM is not currently supported - this is because it is not believed that there is currently a stand-alone UEFI LVM filesystem driver.

Be aware of the `SyncRuntimePermissions` quirk, which may need to be set to avoid early boot failure (typically halting with a black screen) of the Linux kernel, due to a firmware bug of some firmware released after 2017. When present and not mitigated by this quirk, this affects booting via OpenCore with or without OpenLinuxBoot.

After installing OpenLinuxBoot, it is recommended to compare the options shown in the OpenCore debug log when booting (or attempting to boot) a given distro against the options seen using the shell command `cat /proc/cmdline` when the same distro has been booted via its native bootloader. In general (for safety and security of the running distro) these options should match, and if they do not it is recommended to use the driver arguments below (in particular `LINUX_BOOT_ADD_RO`, `LINUX_BOOT_ADD_RW`, `autoopts:{PARTUUID}` and `autoopts`) to modify the options as required. Note however that the following differences are normal and do not need to be fixed:

- If the default bootloader is GRUB then the options generated by OpenLinuxBoot will not contain a `BOOT_IMAGE=...` value where the GRUB options do, and will contain an `initrd=...` value where the GRUB options do not.
- OpenLinuxBoot uses `PARTUUID` rather than filesystem `UUID` to identify the location of `initrd`, this is by design as UEFI filesystem drivers do not make Linux filesystem `UUID` values available.
- Less important graphics handover options (such as discussed in the Ubuntu example given in `autoopts` below) will not match exactly, this is not important as long as distro boots successfully.

If using OpenLinuxBoot with Secure Boot, users may wish to ~~use the `shim-to-cert.tool` included in OpenCore utilities, which can be used to extract the public key needed to boot a distro's kernels directly, as done when using OpenCore with OpenLinuxBoot, rather than via GRUB shim. For non-GRUB distros, the required public key must be found by user research.~~ install a user built, user signed Shim bootloader giving SBAT and MOK integration, as explained in OpenCore ShimUtils.

11. `RequestBootVarRouting`
Type: `plist boolean`
Failsafe: `false`
Description: Request redirect of all `Boot` prefixed variables from `EFI_GLOBAL_VARIABLE_GUID` to `OC_VENDOR_VARIABLE_GUID`.

    This quirk requires `OC_FIRMWARE_RUNTIME` protocol implemented in `OpenRuntime.efi`. The quirk lets default boot entry preservation at times when the firmware deletes incompatible boot entries. In summary, this quirk is required to reliably use the Startup Disk preference pane in firmware that is not compatible with macOS boot entries by design.

    By redirecting `Boot` prefixed variables to a separate GUID namespace with the help of `RequestBootVarRouting` quirk we achieve multiple goals:

    - Operating systems are jailed and only controlled by OpenCore boot environment to enhance security.
    - Operating systems do not mess with OpenCore boot priority, and guarantee fluent updates and hibernation wakes for cases that require reboots with OpenCore in the middle.
    - Potentially incompatible boot entries, such as macOS entries, are not deleted or corrupted in any way.

12. `ResizeUsePciRbIo`
Type: `plist boolean`
Failsafe: `false`
Description: Use PciRootBridgeIo for `ResizeGpuBars` and `ResizeAppleGpuBars`

    The quirk makes `ResizeGpuBars` and `ResizeAppleGpuBars` use `PciRootBridgeIo` instead of PciIo. This is needed on systems with a buggy `PciIo` implementation where trying to configure Resizable BAR results in `Capability I/O Error`. Typically this is required on older systems which have been modified with ReBarUEFI.

13. `ShimRetainProtocol`
Type: `plist boolean`
Failsafe: `false`
Description: Request Linux Shim to keep protocol installed for subsequent image loads.

    This option is only required if chaining OpenCore from Shim. It must be set in order to allow OpenCore to launch items which are verified by certificates present in Shim, but not in the system Secure Boot database.

14. `ResizeGpuBars`
Type: `plist integer`
Failsafe: `-1`
Description: Configure GPU PCI BAR sizes.

    This quirk sets GPU PCI BAR sizes as specified or chooses the largest available below the `ResizeGpuBars` value. The specified value follows PCI Resizable BAR spec. Use `0` for 1 MB, `1` for 2 MB, `2` for 4 MB, and so on up to `19` for 512 GB.

    Resizable BAR technology allows to ease PCI device programming by mapping a configurable memory region, BAR, into CPU address space (e.g. VRAM to RAM) as opposed to a fixed memory region. This technology is necessary, because one cannot map the largest memory region by default, for the reasons of backwards compatibility with older hardware not supporting 64-bit BARs. Consequentially devices of the last decade use BARs up to 256 MB by default (4 remaining bits are used by other data) but generally allow resizing them to both smaller and larger powers of two (e.g. from 1 MB up to VRAM size).

    Operating systems targeting x86 platforms generally do not control PCI address space, letting UEFI firmware decide on the BAR addresses and sizes. This illicit practice resulted in Resizable BAR technology being unused up until 2020 despite being standardised in 2008 and becoming widely available in the hardware soon after.

    Modern UEFI firmware allow the use of Resizable BAR technology but generally restrict the configurable options to failsafe default (`OFF`) and maximum available (`ON`). This quirk allows to fine-tune this value for testing and development purposes.

    Consider a GPU with 2 BARs:

    - `BAR0` supports sizes from 256 MB to 8 GB. Its value is 4 GB.
    - `BAR1` supports sizes from 2 MB to 256 MB. Its value is 256 MB.

6. Sign all the installed drivers and tools with the private key. Do not sign tools that provide administrative access to the computer, such as UEFI Shell.
7. Vault the configuration as explained Vaulting section.
8. Sign all OpenCore binaries (`BOOTX64.efi`, `BOOTIa32.efi`, `OpenCore.efi`, custom launchers) used on this system with the same private key.
9. Sign all third-party operating system (not made by Microsoft or Apple) bootloaders if needed. For Linux there is an option to install ~~Microsoft-signed Shim bootloader as explained on e.g. Debian Wiki~~a user built, user signed Shim bootloader giving SBAT and MOK integration, as explained in the `/Utilities/ShimUtils` directory of OpenCore source or releases.
10. Enable UEFI Secure Boot in firmware preferences and install the certificate with a private key. Details on how to generate a certificate can be found in various articles, such as this one, and are out of the scope of this document. If Windows is needed one will also need to add the Microsoft Windows Production CA 2011. To launch option ROMs or to use signed Linux drivers if not using a user build of Shim, Microsoft UEFI Driver Signing CA will also be needed.
11. Password-protect changing firmware settings to ensure that UEFI Secure Boot cannot be disabled without the user's knowledge.

12.3 Windows support

Can I install Windows?

While no official Windows support is provided, 64-bit UEFI Windows installations (Windows 8 and above) prepared with Boot Camp are supposed to work. Third-party UEFI installations as well as systems partially supporting UEFI boot, such as Windows 7, might work with some extra precautions. Things to consider:

- MBR (Master Boot Record) installations are legacy and ~~will not be supported~~ are only supported with the OpenLegacyBoot driver.
- All the modifications applied (to ACPI, NVRAM, SMBIOS, etc.) are supposed to be operating system agnostic, i.e. apply equally regardless of the OS booted. This enables Boot Camp software experience on Windows.
- macOS requires the first partition to be EFI System Partition, and does not support the default Windows layout. While OpenCore does have a workaround for this, it is highly recommend not to rely on it and install properly.
- Windows may need to be reactivated. To avoid it consider setting SystemUUID to the original firmware UUID. Be aware that it may be invalid on old firmware, i.e., not random. If there still are issues, consider using HWID or KMS38 license or making the use `Custom UpdateSMBIOSMode`. Other nuances of Windows activation are out of the scope of this document and can be found online.

What additional software do I need?

To enable operating system switching and install relevant drivers in the majority of cases Windows support software from Boot Camp is required. For simplicity of the download process or when configuring an already installed Windows version a third-party utility, Brigadier, can be used successfully. Note, that 7-Zip may be downloaded and installed prior to using Brigadier.

Remember to always use the latest version of Windows support software from Boot Camp, as versions prior to 6.1 do not support APFS, and thus will not function correctly. To download newest software pass most recent Mac model to Brigadier, for example `./brigadier.exe -m iMac19,1`. To install Boot Camp on an unsupported Mac model afterwards run PowerShell as Administrator and enter `msiexec /i BootCamp.msi`. If there is a previous version of Boot Camp installed it should be removed first by running `msiexec /x BootCamp.msi` command. `BootCamp.msi` file is located in `BootCamp/Drivers/Apple` directory and can be reached through Windows Explorer.

While Windows support software from Boot Camp solves most of compatibility problems, the rest may still have to be addressed manually:

- To invert mouse wheel scroll direction `FlipFlopWheel` must be set to `1` as explained on SuperUser.
- `RealTimeIsUniversal` must be set to `1` to avoid time desync between Windows and macOS as explained on SuperUser (this is typically not required).