OpenCore

Reference Manual (0.9.5.6)

[2023.11.03]

8. `EnableWriteUnprotector`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Permit write access to UEFI runtime services code.

   This option bypasses `W^X` permissions in code pages of UEFI runtime services by removing write protection (`WP`) bit from `CR0` register during their execution. This quirk requires `OC_FIRMWARE_RUNTIME` protocol implemented in `OpenRuntime.efi`.

   Note: This quirk may potentially weaken firmware security. Please use `RebuildAppleMemoryMap` if the firmware supports memory attributes table (MAT). Refer to the `OCABC: MAT support is 1/0` log entry to determine whether MAT is supported.

9. `FixupAppleEfiImages`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Fix errors in early Mac OS X boot.efi images.

   Modern secure PE loaders will refuse to load `boot.efi` images from Mac OS X 10.4 and 10.5 due to these files containing `W^X` errors and illegal overlapping sections.

   This quirk detects these issues and pre-processes such images in memory, so that a modern loader can accept them.

   Pre-processing in memory is incompatible with secure boot, as the image loaded is not the image on disk, so you cannot sign files which are loaded in this way based on their original disk image contents. Certain firmware will offer to register the hash of new, unknown images - this would still work. On the other hand, it is not particularly realistic to want to start such early, insecure images with secure boot anyway.

   Note 1: The quirk is only applied to Apple-specific 'fat' (both 32-bit and 64-bit versions in one image) `.efi` files, and is never applied during the Apple secure boot path for newer macOS.

   Note 2: The quirk is only needed for loading Mac OS X 10.4 and 10.5, and even then only if the firmware itself includes a modern, more secure PE COFF image loader. This includes current builds of OpenDuet.

10. `ForceBooterSignature`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Set macOS `boot-signature` to OpenCore launcher.

    Booter signature, essentially a SHA-1 hash of the loaded image, is used by Mac EFI to verify the authenticity of the bootloader when waking from hibernation. This option forces macOS to use OpenCore launcher SHA-1 hash as a booter signature to let OpenCore shim hibernation wake on Mac EFI firmware.

    Note: OpenCore launcher path is determined from `LauncherPath` property.

11. `ForceExitBootServices`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Retry `ExitBootServices` with new memory map on failure.

    Try to ensure that the `ExitBootServices` call succeeds. If required, an outdated `MemoryMap` key argument can be used by obtaining the current memory map and retrying the `ExitBootServices` call.

    Note: The need for this quirk is determined by early boot crashes of the firmware. Do not use this option without a full understanding of the implications.

12. `ProtectMemoryRegions`
    Type: `plist boolean`
    Failsafe: `false`
    Description: Protect memory regions from incorrect access.

    Some types of firmware incorrectly map certain memory regions:

    - The CSM region can be marked as boot services code, or data, which leaves it as free memory for the XNU kernel.

This setting allows matching the latest `kernelcache` with a suitable architecture when the `kernelcache` without suffix is unavailable, improving macOS 10.6 boot performance on several platforms.

3. `KernelArch`
   Type: `plist string`
   Failsafe: `Auto` (Choose the preferred architecture automatically)
   Description: Prefer specified kernel architecture (`i386`, `i386-user32`, `x86_64`) when available.

   On macOS 10.7 and earlier, the XNU kernel can boot with architectures different from the usual `x86_64`. This setting will use the specified architecture to boot macOS when it is supported by the macOS and the configuration:

   - `i386` — Use `i386` (32-bit) kernel when available.
   - `i386-user32` — Use `i386` (32-bit) kernel when available and force the use of 32-bit userspace on 64-bit capable processors if supported by the operating system.
     - On macOS, 64-bit capable processors are assumed to support `SSSE3`. This is not the case for older 64-bit capable Pentium processors, which cause some applications to crash on macOS 10.6. This behaviour corresponds to the `-legacy` kernel boot argument.
     - This option is unavailable on macOS 10.4 and 10.5 when running on 64-bit firmware due to an uninitialised 64-bit segment in the XNU kernel, which causes AppleEFIRuntime to incorrectly execute 64-bit code as 16-bit code.
   - `x86_64` — Use `x86_64` (64-bit) kernel when available.

   The algorithm used to determine the preferred kernel architecture is set out below.

   (a) `arch` argument in image arguments (e.g. when launched via UEFI Shell) or in `boot-args` variable overrides any compatibility checks and forces the specified architecture, completing this algorithm.
   (b) OpenCore build architecture restricts capabilities to `i386` and `i386-user32` mode for the 32-bit firmware variant.
   (c) Determined EfiBoot version restricts architecture choice:
   - 10.4-10.5 — `i386` or `i386-user32` (only on 32-bit firmware)
   - 10.6 — `i386`, `i386-user32`, or `x86_64`
   - 10.7 — `i386` or `x86_64`
   - 10.8 or newer — `x86_64`

   (d) If `KernelArch` is set to `Auto` and `SSSE3` is not supported by the CPU, capabilities are restricted to `i386-user32` if supported by EfiBoot.
   (e) Board identifier (from SMBIOS) based on EfiBoot version disables `x86_64` support on an unsupported model if any `i386` variant is supported. `Auto` is not consulted here as the list is not overridable in EfiBoot.
   (f) `KernelArch` restricts the support to the explicitly specified architecture (when not set to `Auto`) if the architecture remains present in the capabilities.
   (g) The best supported architecture is chosen in this order: `x86_64`, `i386`, `i386-user32`.

   Unlike macOS 10.7 (where certain board identifiers are treated as ~~the~~ `i386` only machines), and macOS 10.5 or earlier (where `x86_64` is not supported by the macOS kernel), macOS 10.6 is very special. The architecture choice on macOS 10.6 depends on many factors including not only the board identifier, but also the macOS product type (client vs server), macOS point release, and amount of RAM. The detection of all these is complicated and impractical, as several point releases had implementation flaws resulting in a failure to properly execute the server detection in the first place. For this reason when `Auto` is set, OpenCore on macOS 10.6 falls back ~~on~~ to the `x86_64` architecture ~~whenever~~ when it is supported by the board, as ~~it is~~ on macOS 10.7. The 32-bit `KernelArch` options can still be configured explicitly however.

   A 64-bit Mac model compatibility matrix corresponding to actual EfiBoot behaviour on macOS 10.6.8 and 10.7.5 is outlined below.

Model	10.6 (minimal)	10.6 (client)	10.6 (server)	10.7 (any)
Macmini	4,x (Mid 2010)	5,x (Mid 2011)	4,x (Mid 2010)	3,x (Early 2009)
MacBook	Unsupported	Unsupported	Unsupported	5,x (2009/09)
MacBookAir	Unsupported	Unsupported	Unsupported	2,x (Late 2008)
MacBookPro	4,x (Early 2008)	8,x (Early 2011)	8,x (Early 2011)	3,x (Mid 2007)
iMac	8,x (Early 2008)	12,x (Mid 2011)	12,x (Mid 2011)	7,x (Mid 2007)
MacPro	3,x (Early 2008)	5,x (Mid 2010)	3,x (Early 2008)	3,x (Early 2008)
Xserve	2,x (Early 2008)	2,x (Early 2008)	2,x (Early 2008)	2,x (Early 2008)

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 on supported systems, such as OpenDuet and Mac models capable of legacy booting.

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`:

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 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 are only supported with the OpenLegacyBoot driver on legacy systems.
- 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).