OpenCore

Reference Manual (1.0~~.1~~.2)

[2024.10.06]

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~~ permissions and section errors in macOS `boot.efi` images.

   ~~Modern secure PE loaders will refuse to load `boot.efi` images from~~ Mac OS X ~~10.4 to macOS 10.12 due to these files containing~~ `boot.efi` images contain `W^X` ~~errors (~~permissions errors in all versions~~) and illegal overlapping sections (in~~ , and 10.4 and 10.5 32-bit versions ~~only~~also contain illegal overlapping sections. Modern, strict PE loaders will refuse to load such images unless additional mitigations are applied. The image loader which matters here is the one provided by the system firmware, or by OpenDuet if OpenDuet is providing the UEFI compatibility layer. Image loaders which enforce these stricter rules include the loader provided with current versions of OpenDuet, the loader in OVMF if compiled from audk, and possibly the image loaders of some very recent 3rd party firmware (e.g. Microsoft).

   This quirk detects these issues and pre-processes such images in memory ~~,~~ so that a ~~modern~~ stricter loader will 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 these early, insecure images with secure boot anyway~~On a system with such a modern, stricter loader this quirk is required to load Mac OS X 10.4 to macOS 10.12, and is required for all newer macOS when `SecureBootModel` is set to `Disabled`.

   Note 1: The quirk is never applied during the Apple secure boot path for newer macOS. The Apple secure boot path in OpenCore includes its own separate mitigations for `boot.efi` `W^X` issues.

   Note 2: When enabled, and when not processing for Apple secure boot, this quirk is applied to:

   - All images from Apple Fat binaries (32-bit and 64-bit versions in one image).
   - All Apple-signed images.
   - All images at `\System\Library\CoreServices\boot.efi` within their filesystem.

   Note 3: ~~This quirk is needed for Mac OS X 10.4 to macOS 10.12 (and higher, if Apple secure bootis not enabled), but only when the firmware itself includes a modern, more secure PE COFF image loader. This applies to current builds of OpenDuet, and to OVMF if built from audk source code~~Pre-processing in memory is incompatible with UEFI 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 for future secure boot - this would still work. On the other hand, it is not particularly realistic to want to start these early, insecure images with secure boot anyway.

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.

UEFI audio support provides a way for upstream protocols to interact with the selected audio hardware and resources. All audio resources should reside in `\EFI\OC\Resources\Audio` directory. Currently the supported audio file formats are MP3 and WAVE PCM. While it is driver-dependent which audio stream format is supported, most common audio cards support 16-bit signed stereo audio at 44100 or 48000 Hz.

Audio file path is determined by audio type, audio localisation, and audio path. Each filename looks as follows: `[audio type]_[audio localisation]_[audio path].[audio ext]`. For unlocalised files filename does not include the language code and looks as follows: `[audio type]_[audio path].[audio ext]`. Audio extension can either be `mp3` or `wav`.

- Audio type can be `OCEFIAudio` for OpenCore audio files or `AXEFIAudio` for macOS bootloader audio files.
- Audio localisation is a two letter language code (e.g. `en`) with an exception for Chinese, Spanish, and Portuguese. Refer to `APPLE_VOICE_OVER_LANGUAGE_CODE` definition for the list of all supported localisations.
- Audio path is the base filename corresponding to a file identifier. For macOS bootloader audio paths refer to `APPLE_VOICE_OVER_AUDIO_FILE` definition. For OpenCore audio paths refer to `OC_VOICE_OVER_AUDIO_FILE` definition. The only exception is OpenCore boot chime file, which is `OCEFIAudio_VoiceOver_Boot.mp3`.

Audio localisation is determined separately for macOS bootloader and OpenCore. For macOS bootloader it is set in `preferences.efires` archive in `systemLanguage.utf8` file and is controlled by the operating system. For OpenCore the value of `prev-lang:kbd` variable is used. When native audio localisation of a particular file is missing, English language (`en`) localisation is used. Sample audio files can be found in OcBinaryData repository.

4. `ConnectDrivers`
   Type: `plist boolean`
   Failsafe: `false`
   Description: Perform UEFI controller connection after driver loading.

   This option is useful for loading drivers following UEFI driver model as they may not start by themselves. Examples of such drivers are filesystem or audio drivers. While effective, this option may not be necessary for drivers performing automatic connection, and may slightly slowdown the boot.

   Note: Some types of firmware, particularly those made by Apple, only connect the boot drive to speed up the boot process. Enable this option to be able to see all the boot options when running multiple drives.

5. `Drivers`
   Type: `plist array`
   Failsafe: Empty
   Description: Load selected drivers from `OC/Drivers` directory.

   To be filled with `plist dict` values, describing each driver. Refer to the Drivers Properties section below.

6. `Input`
   Type: `plist dict`
   Description: Apply individual settings designed for input (keyboard and mouse) in the Input Properties section below.

7. `Output`
   Type: `plist dict`
   Description: Apply individual settings designed for output (text and graphics) in the Output Properties section below.

8. `ProtocolOverrides`
   Type: `plist dict`
   Description: Force builtin versions of certain protocols described in the ProtocolOverrides Properties section below.

   Note: all protocol instances are installed prior to driver loading.

9. `Quirks`
   Type: `plist dict`
   Description: Apply individual firmware quirks described in the Quirks Properties section below.

10. `ReservedMemory`
    Type: `plist array`

Failsafe: Empty
Description: To be filled with `plist dict` values, describing memory areas exclusive to specific firmware and hardware functioning, which should not be used by the operating system. Examples of such memory regions could be the second 256 MB corrupted by the Intel HD 3000 or an area with faulty RAM. Refer to the ReservedMemory Properties section below for details.

11. `Unload`
Type: `plist array`
Failsafe: Empty
Description: Unload specified firmware drivers.

    To be filled with `plist string` entries containing the names of firmware drivers to unload before loading the `Drivers` section. This setting is typically only required if a user-provided driver is a variant of an existing system firmware driver, and if the new driver would detect itself as partially loaded, or otherwise fail to operate correctly, if the old driver is not unloaded first.

    Warning: Unloading system firmware drivers is usually not required and not recommended. Poorly written drivers may crash when unloaded, or cause subsequent crashes (e.g by allowing themselves to be unloaded even though they have active dependencies). However standard UEFI network stack drivers should unload cleanly.

    Note 1: See `SysReport/Drivers/DriverImageNames.txt` for the list of drivers which this option can attempt to unload. The relevant name is the driver component name. Drivers are only listed if they implement `DriverBindingProtocol` and `LoadedImageProtocol`, and have an available component name.

    Note 2: The NVRAM `Lang` and `PlatformLang` variables are ignored when determining the driver component names recognised by this option, and listed in the `SysReport` file. This is in order to make unloading images stable across changes in these variables. The UEFI Shell `dh` command takes account of these variables, so in some circumstances may display different driver component names from those listed for this option, unless these variables are cleared.

11.12 APFS Properties

1. `EnableJumpstart`
Type: `plist boolean`
Failsafe: `false`
Description: Load embedded APFS drivers from APFS containers.

    An APFS EFI driver is bundled in all bootable APFS containers. This option performs the loading of signed APFS drivers (consistent with the `ScanPolicy`). Refer to the "EFI Jumpstart" section of the Apple File System Reference for details.

2. `GlobalConnect`
Type: `plist boolean`
Failsafe: `false`
Description: Perform full device connection during APFS loading.

    Every handle is connected recursively instead of the partition handle connection typically used for APFS driver loading. This may result in additional time being taken but can sometimes be the only way to access APFS partitions on certain firmware, such as those on older HP laptops.

3. `HideVerbose`
Type: `plist boolean`
Failsafe: `false`
Description: Hide verbose output from APFS driver.

    APFS verbose output can be useful for debugging.

4. `JumpstartHotPlug`
Type: `plist boolean`
Failsafe: `false`
Description: Load APFS drivers for newly connected devices.

    Permits APFS USB hot plug which enables loading APFS drivers, both at OpenCore startup and during OpenCore picker display. Disable if not required.