Kernel Debug Kit for macOS - Read Me

Use the Kernel Debug Kit (KDK) to debug kernel-level code, such as kernel extensions you create.

Performing Two-Machine Debugging

The KDK supports the debugging of kernel-level code, such as kernel extensions, from a second Mac.

- The *target* device is the Mac that runs the code you want to debug.

- The *host* device is the Mac that runs the debugger.

Identify Device Compatibility

On Apple silicon, perform two-machine debugging using your Mac's built-in Ethernet ports.

On Intel-based Macs, perform two-machine debugging using your Mac's built-in Ethernet ports, the Ethernet port on the Apple Thunderbolt display, or the Apple Thunderbolt to Gigabit Ethernet adapter. You may also use third-party Thunderbolt Ethernet adapters that support one of the following chipsets:

- Broadcom C-IV
- Aquantia AQC107/113
- Intel 82574L

Note: You cannot perform two-machine debugging using USB Ethernet adapters or wireless networking on any Mac.

You must connect the host and target device to the same network, but there are no other restrictions on how the devices connect to that network.

Configure Apple silicon as a Target Device

Use the following steps to configure your Apple silicon as a target device for debugging.

Step 1: Modify the Security Configuration of Your Mac
Modify the security configuration of your system as follows:

1. Reboot your Mac in Recovery Mode.
2. Launch Terminal and run the following command to disable System Integrity Protection (SIP):
    - `csrutil disable`
3. If prompted to lower your system security to "permissive", enter `y` to accept.
4. Reboot your Mac.

Step 2: Identify the Correct Ethernet Device
Run the `ifconfig` tool in Terminal to identify which Ethernet device your target device uses to connect to the network. In the following example, `en1` is the Ethernet device connected to the

network.

```
en0: flags=8963 mtu 1500
   options=60
   ether 32:00:13:er:19:e0
   media: autoselect
   status: inactive
en1: flags=8863 mtu 1500
   options=10b
   ether 40:6c:8f:5b:a2:96
   inet6 fe80::426c:8fff:fe5b:a296%en2 prefixlen 64 scopeid 0x4
   inet6 2620::1b07:114:426c:8fff:fe5b:a296 prefixlen 64 autoconf
   inet6 2620::1b07:114:88d6:bbba:7ac9:b0a7 prefixlen 64 autoconf
temporary
   inet 10.128.19.135 netmask 0xfffff800 broadcast 10.128.23.255
   nd6 options=1
   media: autoselect (1000baseT )
   status: active
```

Step 3: Set the boot-args

Run the `nvram` tool in Terminal to add the following arguments to your target device's `boot-args` key:

- `debug=0x44`—Tells the kernel to wait for a debugger to attach to the device, when the kernel receives a non-maskable interrupt (NMI).
- `kdp_match_name=enX`—Set the value of this key to the Ethernet device (en0, en1, etc.) identified in *Step 2: Identify the Correct Ethernet Device*.
- `wdt=-1`—Disables watchdog monitoring.

For example:

```
sudo nvram boot-args="debug=0x44 kdp_match_name=en1 wdt=-1"
```

Step 4: Reboot the Device

Upon reboot, you may connect to your target device from the host.

Configure an Intel-based Mac as a Target Device

Use the following steps to configure your target device for debugging.

Step 1: Modify the Security Configuration of Your Mac

Modify the security configuration of your system as follows:

1. Reboot your Mac in Recovery Mode.
2. If your device has the Apple T2 Security Chip, set the Secure Boot policy to "Medium Security".
3. Launch Terminal and run the following command to disable System Integrity Protection (SIP):
   - `csrutil disable`
4. Reboot your Mac.

Step 2: Identify the Correct Ethernet Device

Run the `ifconfig` tool in Terminal to identify which Ethernet device your target device uses to connect to the network. In the following example, `en1` is the Ethernet device connected to the

network.

```
en0: flags=8963 mtu 1500
   options=60
   ether 32:00:13:er:19:e0
   media: autoselect
   status: inactive
en1: flags=8863 mtu 1500
   options=10b
   ether 40:6c:8f:5b:a2:96
   inet6 fe80::426c:8fff:fe5b:a296%en2 prefixlen 64 scopeid 0x4
   inet6 2620::1b07:114:426c:8fff:fe5b:a296 prefixlen 64 autoconf
   inet6 2620::1b07:114:88d6:bbba:7ac9:b0a7 prefixlen 64 autoconf
temporary
   inet 10.128.19.135 netmask 0xfffff800 broadcast 10.128.23.255
   nd6 options=1
   media: autoselect (1000baseT )
   status: active
```

Step 3: Set the boot-args

Run the `nvram` tool in Terminal to add the following arguments to your target device's `boot-args` key:

- `debug=0x44`—Tells the kernel to wait for a debugger to attach to the device, when the kernel receives a non-maskable interrupt (NMI).
- `kdp_match_name=enX`—Set the value of this key to the Ethernet device (en0, en1, etc.) identified in *Step 2: Identify the Correct Ethernet Device*.
- `wdt=-1`—Disables watchdog monitoring.

For example:

```
sudo nvram boot-args="debug=0x44 kdp_match_name=en1 wdt=-1"
```

Step 4: Reboot the Device

Upon reboot, you may connect to your target device from the host.

Configure the Host Device

Configure your host device to initiate debugging using the following steps:

Step 1: Install Xcode

Install Xcode. Then install these additional Python packages for kernel debugging:

```
$ xcrun python3 -m pip install --user --ignore-installed macholib
$ xcrun python3 -m pip install --user --ignore-installed future
```

Step 2: Install the KDK

Install this KDK on the host device.

Trigger the Target Device to Wait for the Debugger

After you configure the target device, it halts and waits for an external debugger to attach in the following situations:

- If the target device panics, it automatically halts and waits for an external debugger to

attach.

- If you trigger an NMI on the target device, the device halts and waits for an external debugger to attach.

Connect the Debugger to the Target Device

Run `lldb` from Terminal and use the `kdp-remote` command to connect to the target device. That command accepts the hostname or IP address of the target device, and creates a debugger connection to it.

`(lldb)kdp-remote` *`{name_or_ip_address}`*

The `lldb` tool automatically searches any spotlight-indexed directories and the `/Library/Developer/KDKs/` directory on the local system for symbol information. If lldb is unable to find a kernel binary automatically, you can specify it on the command line when you launch `lldb`. For example:

```
lldb /Library/Developer/KDKs/<KDK Version>/System/Library/Kernels/kernel
```

Note: Apple silicon doesn't support active kernel debugging. You may inspect the current state of the kernel when it is halted due to a panic or NMI. However, you cannot set breakpoints, continue code execution, step into code, step over code, or step out of the current instruction.

Finish Up Your Debugging Session

When you finish debugging the target device, quit `lldb` to disconnect the host from the target machine.

When you no longer need to debug the target device, return the device to normal operation:

Step 1: Remove the boot-args
Run the `nvram` tool in Terminal to remove the `boot-args` key.

```
sudo nvram -d boot-args
```

Step 2: Restore the Security Configuration of Your Mac
Modify the security configuration of your Mac by disabling SIP and enabling permissive security:

1. Reboot your Mac in Recovery Mode.
2. Connect your Mac to the internet, via a cable or Wi-Fi (internet access may be required to restore full security).
3. Launch Terminal from the Utilities menu and run the following command to enable System Integrity Protection (SIP):
   - `csrutil enable`
4. If prompted to raise the security to "full", enter `y` to accept.
5. If your device is an Intel-based Mac that has the Apple T2 Security Chip, set the Secure Boot policy to "Full Security".
6. Reboot your Mac.

Installing Kernel and Kernel Extension Variants

The KDK includes several variants of the kernel binary and kernel extensions. These variants

include additional assertions and error checking beyond what is present in the shipping version of the macOS kernel, and may be useful for debugging in various situations. Choose the variant that best suits your needs:

- The *kernel* (release) variant matches the shipping kernel for users.
- The *kernel.development* (development) variant is safe for everyday use during development, and has minimal performance overhead.
- The *kernel.kasan* (kasan) variant has more significant performance and memory overhead, and enables address sanitizer features that will, for example, panic the kernel upon memory safety violations.

When performing two-machine debugging, install these kernel variants on the target device.

Note: Apple silicon doesn't support installing the kernel and kernel extension variants from the KDK.

Install the Variants on an Intel-based Mac

To install new kernel and kernel extension variants on the target device, perform the following steps:

Step 1: Install the KDK
Install this KDK on the target device.

Step 2: Modify the Security Configuration of Your Mac
Modify the security configuration of your Mac by disabling SIP and enabling permissive security:

1. Reboot your Mac in Recovery Mode.
2. If your device has the Apple T2 Security Chip, set the Secure Boot policy to "Medium Security".
3. Launch Terminal from the Utilities menu and run the following commands to disable System Integrity Protection (SIP) and Authenticated Root protection:
   - `csrutil disable`
   - `csrutil authenticated-root disable`
4. Reboot your Mac.

Step 3: Identify the System Volume Disk Device Name
Run the `mount` command in Terminal to identify the devices of your system volume snapshot. Look for the device mounted at "/", which identifies the system volume disk. In the following example, this disk is `/dev/disk4s5s1`.

```
/dev/disk4s5s1 on / (apfs, sealed, local, read-only, journaled)
devfs on /dev (devfs, local, nobrowse)
/dev/disk4s4 on /System/Volumes/VM (apfs, local, noexec, journaled, noatime, nobrowse)
/dev/disk4s2 on /System/Volumes/Preboot (apfs, local, journaled, nobrowse)
/dev/disk4s1 on /System/Volumes/Data (apfs, local, journaled, nobrowse)
```

To get the actual name of the system volume disk, remove the final "s*X*" from the device. In the preceding example, the name of the system volume disk is `/dev/disk4s5`.

Step 4: Mount a Live Version of the System Volume

Run the `mount` command in Terminal to mount the system volume disk to a temporary location. When running the mount command, always include the `nobrowse` mount option to prevent Spotlight from indexing the volume.

```
mkdir /Users/jappleseed/livemount
```

```
sudo mount -o nobrowse -t apfs  /dev/disk4s5 /Users/jappleseed/
livemount
```

Step 5: Add the Variant Files to the Live Mount

Add the kernel variant files to the newly mounted disk. The easiest way to copy the files is to copy the entire `/System` directory of the KDK into the `/System` directory of your mounted disk, as shown in the following example:

```
sudo ditto /Library/Developer/KDKs/<KDK Version>/System /Users/
jappleseed/livemount/System
```

Step 6: Rebuild the Kernel Collections and Bless the Changes

Run `kmutil` in Terminal to rebuild the kernel collections for the variants you added to your mounted disk. Run `bless` to authorize booting from your modified kernels.

```
sudo kmutil install --volume-root /Users/jappleseed/livemount --update-
    all
```

```
sudo bless --mount /Users/jappleseed/livemount --bootefi --create-
    snapshot
```

Step 7: Set the boot-args

Run the `nvram` tool in Terminal to add the following argument to your device's `boot-args` key:

- `kcsuffix` — Set the value of this key to either `development` or `kasan`, based on which kernel variant you want to use. Omit this argument if you want the release kernel.

For example:

```
  sudo nvram boot-args="debug=0x44 kdp_match_name=en1 wdt=-1
kcsuffix=development"
```

Step 8: Reboot the Device and Verify the Variant is Running

Upon rebooting, the chosen kernel variant will be active. You can verify this by running the `sysctl` command in Terminal to check the `kern.osbuildconfig` property:

```
sysctl kern.osbuildconfig
```

```
kern.osbuildconfig: development
```

Removing the Variants

When you are finished debugging, perform the following steps to remove the kernel variants and restore your device security:

Step 1: Remove the boot-args

Run the `nvram` tool in Terminal to remove the `boot-args` key.

```
  sudo nvram -d boot-args
```

Step 2: Identify the System Volume Disk Device Name

Run the `mount` command in Terminal to identify the devices of your system volume snapshot. Look for the device mounted at "/", which identifies the system volume disk. In the following example, this disk is `/dev/disk4s5s1`.

```
/dev/disk4s5s1 on / (apfs, sealed, local, read-only, journaled)
devfs on /dev (devfs, local, nobrowse)
/dev/disk4s4 on /System/Volumes/VM (apfs, local, noexec, journaled, noatime, nobrowse)
/dev/disk4s2 on /System/Volumes/Preboot (apfs, local, journaled, nobrowse)
/dev/disk4s1 on /System/Volumes/Data (apfs, local, journaled, nobrowse)
```

To get the actual name of the system volume disk, remove the final "s*X*" from the device. In the preceding example, the name of the system volume disk is `/dev/disk4s5`.

Step 3: Mount a Live Version of the System Volume

Run the `mount` command in Terminal to mount the system volume disk to a temporary location. When running the `mount` command, always include the `nobrowse` mount option to prevent Spotlight from indexing the volume.

```
mkdir /Users/jappleseed/livemount
sudo mount -o nobrowse -t apfs  /dev/disk4s5 /Users/jappleseed/livemount
```

Step 4: Restore the System Sealed Snapshot

Run `bless` to restore booting from the system sealed snapshot.

```
sudo bless --mount /Users/jappleseed/livemount --bootefi --last-sealed-snapshot
```

Step 5: Restore the Security Configuration of Your Mac

Modify the security configuration of your Mac by disabling SIP and enabling permissive security:

1. Reboot your Mac in Recovery Mode.
2. Connect your Mac to the internet, via a cable or Wi-Fi (internet access may be required to restore full security).
3. Launch Terminal from the Utilities menu and run the following commands to restore Authenticated Root protection and System Integrity Protection (SIP):
   - `csrutil authenticated-root enable`
   - `csrutil enable`
4. If your device has the Apple T2 Security Chip, set the Secure Boot policy to "Full Security".
5. Reboot your Mac.

Step 6: Verify the Release Variant is Running

Verify you have restored the "release" kernel variant by running the `sysctl` command in Terminal to check the `kern.osbuildconfig` property:

```
sysctl kern.osbuildconfig
kern.osbuildconfig: release
```

Configuring a Core Dump Server

You can use another Mac to capture core dumps from your system automatically over the network. This may be useful for kernel panics that occur early in the boot cycle, or when the kernel fails to write the core-dump file to disk after a panic. You can use a core dump server simultaneously with two-machine debugging. Use the following terminology to refer to the Macs associated with core dumps:

- The *server* device is the Mac that receives the core-dump files.

- The *target* device is the Mac that generates the core-dump file.

Configure the Server Device

Configure the server device to receive the core dump files by performing the following steps:

Step 1: Create the PanicDumps Directory

Create a `/var/tmp/PanicDumps` directory on the server, and configure it to be writable by any user. This directory is where the server writes the core-dump files.

```
mkdir /var/tmp/PanicDumps
chmod 1777 /var/tmp/PanicDumps
```

Step 2: Load the kdumpd Launch Daemon

Run the `launchctl` command in Terminal to load the kdumpd launch daemon. This service listens for incoming core dumps and writes them to `/var/tmp/PanicDumps`.

```
sudo launchctl load -w /System/Library/LaunchDaemons/
com.apple.kdumpd.plist
```

Configure the Target Device

The same compatibility requirements apply to the target device as for two-machine debugging. You must also configure the target device as if you're using it for two-machine debugging. (See: "Configure Apple silicon as a Target Device" or "Configure an Intel-based Mac as a Target Device" in the Performing Two-Machine Debugging section above)

After performing those configuration steps, perform the following additional steps:

Step 1: Set additional boot-args

Run the `nvram` tool in Terminal to add the following arguments to your target device's `boot-args` key:

- `debug=0xc44`—Tells the kernel to create a core dump when a panic or NMI occurs.
- `_panicd_ip=10.0.40.2`—Set the value of this key to the IP address of the server device.

```
sudo nvram boot-args="debug=0xc44 kdp_match_name=en1 wdt=-1
_panicd_ip=10.0.40.2"
```

Step 2: Reboot the Device

After you reboot the target device, it sends core dumps to the server in the following situations:

- If the target device panics.
- If you trigger an NMI on the target device.

Core dumps can be very large, so it may take some time for the core dump to transfer over the network. Additionally, ensure the server device has sufficient free disk space.

Finish Up

When you no longer want to gather core dumps from the target device, return that device to normal operation (See: “Finish Up Your Debugging Session” in the Performing Two-Machine Debugging section above)

When you no longer wish to capture core dumps on the server device, run the following to unload the `kdumpd` launch daemon:

```
sudo launchctl unload -w /System/Library/LaunchDaemons/
com.apple.kdumpd.plist
```