Debugging macOS Kernel using VirtualBox

Posted on April 10, 2017

Update: In the HN discussion (https://news.ycombinator.com/item?id=14079163), awalton mentioned you can set CPUID flags in VMWare. Simply adding `cpuid.7.ebx = "-----------0--------------------"` to the vmx file will disable SMAP.

Late last year, I upgraded my old MBP to the 2016 model with a Skylake processor. As I was debugging a kernel exploit, it turned out that SMAP (https://lwn.net/Articles/517475/) was enabled inside my VMWare Fusion VM. I wanted to avoid dealing with SMAP, but couldn't figure out how to disable it in Fusion. Luckily, VirtualBox VMs do not support SMAP (yet?).

This post will be a step-by-step guide on how to setup macOS kernel source-level debugging using VirtualBox. Though all the step examples are geared toward VirtualBox, this guide can also be used to setup kernel debugging on VMWare Fusion since it's even more straightforward in Fusion.

Installing VirtualBox and Sierra

If you don't already have a macOS VirtualBox VM, we must first install the target macOS on a VM. You can either provide the vmdk from a VMWare Fusion VM, or create a fresh VM. VirtualBox requires an ISO image to install the OS for newly created VMs. The commands below can be used to create an ISO from the Sierra install app obtained from the Mac app store (https://itunes.apple.com/us/app/macos-sierra/id1127487414?ls=1&mt=12).

```
$ hdiutil attach /Applications/Install\ macOS\ Sierra.app/Contents/SharedSupport/Instal
$ hdiutil create -o /tmp/Sierra -size 8g -type SPARSE -layout SPUD -fs HFS+J
$ hdiutil attach /tmp/Sierra.sparseimage -noverify -nobrowse -mountpoint /Volumes/insta
$ asr restore -source /Volumes/installesd/BaseSystem.dmg -target /Volumes/install -nopr
$ rm /Volumes/OS\ X\ Base\ System/System/Installation/Packages
$ cp -rp /Volumes/installesd/Packages /Volumes/OS\ X\ Base\ System/System/Installation/
$ cp -rp /Volumes/installesd/BaseSystem.dmg /Volumes/OS\ X\ Base\ System/BaseSystem.dmg
$ cp -rp /Volumes/installesd/BaseSystem.chunklist /Volumes/OS\ X\ Base\ System/BaseSyst
$ hdiutil detach /Volumes/installesd
$ hdiutil detach /Volumes/OS\ X\ Base\ System/
$ hdiutil resize -sectors min /tmp/Sierra.sparseimage
$ hdiutil convert /tmp/Sierra.sparseimage -format UDTO -o /tmp/Sierra
$ rm /tmp/Sierra.sparseimage
$ mv /tmp/Sierra.cdr /tmp/Sierra.iso
```

Networking

If you are using a bridged adapter, there isn't anything special you need to do.

If you decide to go with NAT, you'll need to enable port forwarding for KDP to work with the VM. In the adapter settings, choose *Advanced*→*Port Forwarding*. We need to reach 41139/UDP on the debugee VM, so I forward localhost 41139/UDP to the VM's 41139/UDP.

Installing XCode

Install XCode on your host machine. The easiest way is to install it from the Mac app store (https://itunes.apple.com/us/app/xcode/id497799835?ls=1&mt=12). After installing, accepting the XCode license is required either by opening XCode and accepting, or through command line.

```
$ sudo xcodebuild -license accept
```

Install Kernel Debug Kit (KDK) on

On our host debugger machine, we need to install the KDK from the Apple Developer site (https://developer.apple.com/download/more/) corresponding to our debugee macOS version and build. In this guide, I used 10.12 build 16A323.

The KDK installs to `/Library/Developer/KDKs` and provides RELEASE, DEVELOPMENT, and DEBUG kernels for macOS, as well as symbols for these kernels and various Apple kexts. The difference between the different kernels is that the DEVELOPMENT and DEBUG kernels have additional assertions and error checking compared to RELEASE with the DEBUG build having even more than DEVELOPMENT.

Note: The debugee system does not need to have the KDK installed.

Update nvram boot-args

In order to debug the VM, we must set the `debug` option of `boot-args` in nvram on our debugee VM. There are numerous options in addition to `debug` that we can use. Below are a few that could be of interest including `debug`.

- `-v`: Always boot the system in verbose mode.
- `kcsuffix`: Specifies which kernel to boot using a given suffix.
- `pmuflags`: Many people still seem to recommend setting this option to 1. However, Apple's Kernel Programming Guide (https://developer.apple.com/library/content/documentation/Darwin/Conceptual/KernelProgramming/ says the power management watchdog timer "is only present in G4 and earlier desktops and laptops and in early G5 desktops", and the other primary watchdog timer is "normally only enabled in OS X Server." Thus, this option doesn't seem to do anything, though setting it doesn't hurt.
- `-zc zlog1=<zone_name>`: `zc` in conjunction with `zlog#` logs both allocations and frees to the specified zone where # is 1-5.
- `debug`: This option allows us to perform remote kernel debugging. Available flags are listed in the Apple docs (https://developer.apple.com/library/content/documentation/Darwin/Conceptual/KernelProgramming/ CH221-BABDGEGF). I usually use `DB_LOG_PI_SCRN | DB_ARP | DB_NMI`.
  - Non-maskable interrupts (NMI) can be triggered by pressing *control + option + command + shift + escape*. Triggering an NMI will break in the debugger which is super convenient. This key combo does not play well with VirtualBox when it covers the host key combo so I rebound the host key to *right command + right option*.

Modifying nvram

In VMWare Fusion, you modify nvram using the `nvram` command like so:

```
$ sudo nvram boot-args="-v debug=0x144"
```

On VirtualBox, you'll find it's not so easy. After a reboot, the nvram modifications will have disappeared. VirtualBox User Manual §3.13.2 (https://www.virtualbox.org/manual/ch03.html#idm1685) sheds some light:

> *It is currently not possible to manipulate EFI variables from within a running guest (e.g., setting the "boot-args" variable by running the nvram tool in a Mac OS X guest will not work). As an alternative way, "VBoxInternal2/EfiBootArgs" extradata can be passed to a VM in order to set the "boot-args" variable. To change the "boot-args" EFI variable:*

Thus, we need to shutdown our VM and run the commands below on our host.

```
$ VBoxManage list vms # take the UUID to use in the next command
"macOS 10.12.0" {9ad936f8-9360-44a6-ba3e-c4d92b4243e8}
$ VBoxManage setextradata 9ad936f8-9360-44a6-ba3e-c4d92b4243e8 VBoxInternal2/EfiBootArg
```

Swapping Kernels

I alluded to debugging different builds of kernels previously, mentioning that the `kcsuffix` option specifies which kernel build to use. The kernel file must be at `/System/Library/Kernels` on the debugee VM. It should not be a surprise that this directory is protected by System Integrity Protection (SIP) (https://support.apple.com/en-us/HT204899). Therefore, if you want to use a KDK kernel or a self-compiled kernel (http://shantonu.blogspot.ca), you must first boot into recovery, copy the target kernel to the above directory, invalidate the kext cache, and then reboot.

Reliably Booting into Recovery

In Fusion, booting into recovery mode using *cmd+R* is as easy as doing so on a physical machine. VirtualBox, on the other hand, requires a few more steps (http://anadoxin.org/blog/disabling-system-integrity-protection-from-guest-el-capitan-under-virtualbox-5.html).

When booting the VM, hit F12, and select *Boot Manager→EFI Internal Shell*. You will be greeted by an EFI shell. To boot into recovery, type:

```
FS2:\com.apple.recovery.boot\boot.efi
```

Once the recovery GUI loads, launch a terminal, move the target kernels, then invalidate the kextcache.

```
# mv /path/to/kernels/kernel.development /System/Library/Kernels
# kextcache -invalidate /Volumes/Macintosh\ HD
# reboot
```

Before reboot, you can optionally disable SIP if desired.

```
# csrutil disable
Successfully disabled System Integrity Protection. Please restart the machine for the c
```

Source-level Debugging

Download the XNU source code (https://opensource.apple.com) corresponding to the debuggee XNU version. To gain source-level debugging, LLDB will look in `/Library/Caches/com.apple.xbs/Sources/xnu/xnu-...` for the kernel source. You can either place the downloaded source there, or create a symlink there that points to the source. Alternatively, you can also set `target.source-map` in LLDB.

```
lldb> settings set target.source-map /Library/Caches/com.apple.xbs/Sources/xnu/xnu-3789
```

Previous versions of macOS like Yosemite, you had to place source code in `/SourceCache/xnu/`.

Setting up LLDB

Finally now, we can break out the debugger. The example below sets the target file to the RELEASE kernel build.

To use the XNU LLDB macros in Sierra KDK, the `macholib` Python module is required now. A simple `pip install macholib` should do the trick. To use the nifty LLDB macros, copy paste the KDK debug script command that is prompted when you first set the target file to a KDK kernel.

After triggering an NMI (or waiting for debugger to halt the boot process if you chose `DB_HALT` flag), connect to the debugee with the command `kdp-remote <ip>` where `<ip>` is the IP address (localhost if you used the NAT port forwarding).

```
$ lldb /Library/Developer/KDKs/KDK_10.11.2_15C50.kdk/System/Library/Kernels/kernel
(lldb) target create "/Library/Developer/KDKs/KDK_10.11.2_15C50.kdk/System/Library/Kern
warning: 'kernel' contains a debug script. To run this script in this debug session:

    command script import "/Library/Developer/KDKs/KDK_10.11.2_15C50.kdk/System/Library

To run all discovered debug scripts in this session:

    settings set target.load-script-from-symbol-file true

Current executable set to '/Library/Developer/KDKs/KDK_10.11.2_15C50.kdk/System/Library
(lldb) command script import "/Library/Developer/KDKs/KDK_10.11.2_15C50.kdk/System/Libr
Loading kernel debugging from /Library/Developer/KDKs/KDK_10.11.2_15C50.kdk/System/Libr
LLDB version lldb-370.0.40
  Swift-3.1
settings set target.process.python-os-plugin-path "/Library/Developer/KDKs/KDK_10.11.2_
settings set target.trap-handler-names hndl_allintrs hndl_alltraps trap_from_kernel hnd
command script import "/Library/Developer/KDKs/KDK_10.11.2_15C50.kdk/System/Library/Ker
xnu debug macros loaded successfully. Run showlldbtypesummaries to enable type summarie


(lldb) kdp-remote 192.168.149.184
Version: Darwin Kernel Version 15.2.0: Fri Nov 13 19:56:56 PST 2015; root:xnu-3248.20.5
Kernel UUID: 17EA3101-D2E4-31BF-BDA9-931F51049F93
Load Address: 0xffffff8007a00000
Kernel slid 0x7800000 in memory.
Loaded kernel file /Library/Developer/KDKs/KDK_10.11.2_15C50.kdk/System/Library/Kernels
Target arch: x86_64
Instantiating threads completely from saved state in memory.
Loading 82 kext modules warning: Can't find binary/dSYM for com.apple.kec.corecrypto (4
.warning: Can't find binary/dSYM for com.apple.kec.pthread (0888BA0A-49EE-394A-AEB1-1E5

(omitted...)

. done.
kernel was compiled with optimization - stepping may behave oddly; variables may not be
Process 1 stopped
* thread #2, name = '0xffffff800db8b000', queue = '0x0', stop reason = signal SIGSTOP
    frame #0: 0xffffff8007bd655e kernel`Debugger(message=<unavailable>) at model_dep.c:
   1017
   1018     doprnt_hide_pointers = old_doprnt_hide_pointers;
   1019     __asm__("int3");
-> 1020     hw_atomic_sub(&debug_mode, 1);
   1021 }
   1022
   1023 char *
(lldb)
```

Voila, source-level debugging macOS kernel!

← **PREVIOUS POST (HTTPS://KLUE.GITHUB.IO/BLOG/2017/03/PEGASUS_SHIFT/)**

NEXT POST → (HTTPS://KLUE.GITHUB.IO/BLOG/2017/09/HEVD_STACK_GS/)

I

irad

8 years ago

How about debugging my kernel kext from xcode (or any other gui wrapper for lldb) ? is it possible to activate the lldb in kdp mode from xcode? I couldn't find a way to open lldb command line in order to connect the machine under debug.

2 0 Reply

liangweidarth

2 years ago

When I trigger the nmi, screen log "waiting for remote debugger connection. kdp_poll: no debugger device", and I try to lldb>kdb-remote ip, it says "KDP_REATTACH failed", could you give me any clue to solve this?

0 0 Reply

S

swarm3d

6 years ago edited

I had an issue where I got stuck on a kernel panic on boot after trying to switch back to the original kernel. After a lot of trial and error, I solved this by deleting ALL of the prelinked kernels using Recovery mode. The location is:

```
/System/Library/PrelinkedKernels
```

"touch"ing the files may have worked as well, but I didn't try it since they're presumably generated on boot if not found. Not sure why this was necessary, since the prelinked kernels are ostensibly specific to the kernel filenames (e.g. "kernel.development"), or what I could have done to avoid it, but it solved the issue for me.

0 0 Reply

memleak

8 years ago edited

Hi, **@klue**. I'm new to Mac.

"This key combo does not play well with VirtualBox when it covers the host key combo so I rebound the host key to right command + right option"

How to rebind host key to right command + right option?
Can anyone else also help me out, :-D

0 0 Reply

K

klue Mod memleak

8 years ago

VirtualBox Preferences -> Input -> Virtual Machine -> Host Key Combination.

0 0 Reply

Steven Irby

8 years ago

Huh I can't get macOS to boot up in virtualbox. I keep getting this error:

gIOScreenLockState

Virtualbox v5.1.18

0 0 Reply

freme Steven Irby
8 years ago

me2 does anyone have a fix for this - I have no clue where to look . . .