

Running iOS in QEMU to an interactive bash shell

(1): tutorial

By [Jonathan Afek \(@JonathanAfek\)](#)

June 17, 2019

For better reading experience please follow the link to original blog posts [\(1\)](#) [\(2\)](#):

While wanting to do some iOS security research and inspired by the work done by [zhuowei](#), I decided to try and get this emulation project further along the boot process. The goal was to get the system to boot without having to patch the kernel beforehand or during the boot process, have new modules that extend QEMU's capabilities to execute arm64 XNU systems and, get an interactive bash shell. This post is the first post in a 2-post series, in which I will present instructions for executing iOS on QEMU and launching an interactive bash shell. In the second post, I will detail some of the research that was required in order to get there. For this project, the iOS version and device that were chosen are iOS 12.1 and iPhone 6s Plus, because this specific iOS 12 image comes with a lot of symbols exported in the kernel image compared to other iOS kernel images that are usually stripped of most symbols. This presented some more challenges, because it is a non-KTRR device that uses a secure monitor image, and that required changes in the stages of the solution already done by zhuowei. Another change is that I wanted this feature to be in external modules that could later be extended and used to create modules for other iOS devices and versions, instead of having the code inside the core QEMU code.

Current Status

The project is now available at [qemu-aleph-git](#) with the required scripts at [qemu-scripts-aleph-git](#). The current status allows booting to user mode with a read only mounted ram disk, to which new executables and launchd items can be added (before boot), and those can use the [dyld cache](#) from the main disk image copied to the ram disk and communicate with the user via the emulated UART channel. Here is a demonstration of running an interactive bash shell with this project:

This lets you execute whatever user mode process you want as root, with whatever entitlements you choose, and debug the process and/or the kernel with a kernel debugger:



Current Limitations

1. There is a long suspension where nothing happens for a few seconds before mounting the ram disk.
2. It only works with a ram disk image which is mounted as read only, and is limited to 2GB in size.
3. We can only communicate with the guest iOS over UART, and no other communication channels are currently available.
4. No devices emulation: screen, touch, wifi, BT or anything else.
5. Only a single emulated CPU is currently supported.

Tutorial

To start the process we first need to prepare a kernel image, a secure monitor image, a device tree, a static trust cache, and ram disk images. To get the images we first need to get the update file from Apple: [iOS 12.1 update file](#). This is actually a zip file which we can extract:

```
Downloads jonathanafek$ unzip iPhone_5.5_12.1_16B92_Restore.ipsw
Archive: iPhone_5.5_12.1_16B92_Restore.ipsw
  creating: Firmware/
  inflating: Restore.plist
  creating: Firmware/usr/
  creating: Firmware/usr/local/
  inflating: BuildManifest.plist
  inflating: Firmware/Mav10-7.21.00.Release.plist
  creating: Firmware/all_flash/
  inflating: Firmware/all_flash/DeviceTree.n66ap.im4p.plist
  inflating: Firmware/all_flash/LLB.n56.RELEASE.im4p
  inflating: Firmware/all_flash/batterylow1@3x~iphone.im4p
  inflating: Firmware/all_flash/batterycharging0@3x~iphone.im4p
  inflating: Firmware/all_flash/LLB.n66.RELEASE.im4p
  inflating: Firmware/all_flash/sep-firmware.n56.RELEASE.im4p.plist
  inflating: Firmware/all_flash/iBoot.n56.RELEASE.im4p.plist
  inflating: Firmware/all_flash/batteryfull@3x~iphone.im4p
  inflating: Firmware/all_flash/iBoot.n66m.RELEASE.im4p
  inflating: Firmware/all_flash/iBoot.n56.RELEASE.im4p
  inflating: Firmware/all_flash/DeviceTree.n66ap.im4p
  inflating: Firmware/all_flash/sep-firmware.n66m.RELEASE.im4p.plist
  inflating: Firmware/all_flash/applelogo@3x~iphone.im4p
```

```

inflating: Firmware/all_flash/recoverymode@1920~iphone-lightning.im4p
creating: Firmware/dfu/
inflating: Firmware/dfu/iBSS.n56.RELEASE.im4p.plist
inflating: Firmware/all_flash/glyphplugin@1920~iphone-lightning.im4p
inflating: Firmware/all_flash/batterylow@3x~iphone.im4p
inflating: Firmware/dfu/iBEC.n66m.RELEASE.im4p.plist
inflating: Firmware/dfu/iBSS.n66.RELEASE.im4p
inflating: Firmware/048-32459-105.dmg.trustcache
inflating: Firmware/dfu/iBSS.n66m.RELEASE.im4p
inflating: Firmware/dfu/iBEC.n56.RELEASE.im4p.plist
inflating: Firmware/all_flash/sep-firmware.n56.RELEASE.im4p
inflating: Firmware/Mav13-5.21.00.Release.bbfw
inflating: Firmware/all_flash/sep-firmware.n66m.RELEASE.im4p
inflating: Firmware/all_flash/LLB.n66m.RELEASE.im4p.plist
inflating: Firmware/all_flash/iBoot.n66.RELEASE.im4p.plist
inflating: Firmware/dfu/iBSS.n56.RELEASE.im4p
inflating: Firmware/all_flash/DeviceTree.n66map.im4p.plist
inflating: Firmware/all_flash/DeviceTree.n56ap.im4p.plist
inflating: Firmware/all_flash/LLB.n66.RELEASE.im4p.plist
creating: Firmware/AOP/
inflating: Firmware/AOP/aopfw-s8000aop.im4p
inflating: Firmware/dfu/iBEC.n56.RELEASE.im4p
inflating: Firmware/all_flash/LLB.n66m.RELEASE.im4p
inflating: Firmware/all_flash/iBoot.n66.RELEASE.im4p
inflating: Firmware/all_flash/sep-firmware.n66.RELEASE.im4p
inflating: Firmware/048-31952-103.dmg.trustcache
inflating: Firmware/all_flash/sep-firmware.n66.RELEASE.im4p.plist
inflating: Firmware/dfu/iBSS.n66.RELEASE.im4p.plist
inflating: Firmware/all_flash/DeviceTree.n66map.im4p
inflating: Firmware/dfu/iBSS.n66m.RELEASE.im4p.plist
inflating: Firmware/all_flash/batterycharging1@3x~iphone.im4p
inflating: Firmware/all_flash/iBoot.n66m.RELEASE.im4p.plist
inflating: 048-32651-104.dmg
inflating: Firmware/all_flash/LLB.n56.RELEASE.im4p.plist
inflating: Firmware/dfu/iBEC.n66.RELEASE.im4p
inflating: Firmware/dfu/iBEC.n66.RELEASE.im4p.plist
inflating: Firmware/dfu/iBEC.n66m.RELEASE.im4p
inflating: kernelcache.release.ip hone7
inflating: Firmware/048-32651-104.dmg.trustcache
inflating: Firmware/Mav13-5.21.00.Release.plist
inflating: Firmware/all_flash/DeviceTree.n56ap.im4p
inflating: Firmware/Mav10-7.21.00.Release.bbfw
inflating: 048-32459-105.dmg
inflating: kernelcache.release.n66
extracting: 048-31952-103.dmg

```

Next, we need to clone the supporting scripts repository:

```

Downloads jonathanafek$ git clone git@github.com:alephsecurity/xnu-qemu-arm64-scripts.git
Cloning into 'xnu-qemu-arm64-scripts'...
remote: Enumerating objects: 16, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 16 (delta 4), reused 16 (delta 4), pack-reused 0
Receiving objects: 100% (16/16), 5.16 KiB | 5.16 MiB/s, done.
Resolving deltas: 100% (4/4), done.

```

And extract the ASN1 encoded kernel image:

```

Downloads jonathanafek$ python xnu-qemu-arm64-scripts/asn1kerneldecode.py kernelcache.release.n66
kernelcache.release.n66.asn1decoded

```

This decoded image now includes the compressed kernel and the secure monitor image.
To extract both of them:

```

Downloads jonathanafek$ python xnu-qemu-arm64-scripts/decompress_lzss.py
kernelcache.release.n66.asn1decoded kernelcache.release.n66.out
Downloads jonathanafek$ python xnu-qemu-arm64-scripts/kernelcompressedextractmonitor.py
kernelcache.release.n66.asn1decoded securemonitor.out

```

Now let's prepare a device tree which we can boot with (more details about the device tree in the second post). First, extract it from the ASN1 encoded file:

```
Downloads jonathanafek$ python xnu-qemu-arm64-scripts/asn1dtredecode.py  
Firmware/all_flash/DeviceTree.n66ap.im4p Firmware/all_flash/DeviceTree.n66ap.im4p.out
```

Then, parse it and modify it to make our kernel boot on QEMU:

```
Downloads jonathanafek$ python xnu-qemu-arm64-scripts/read_device_tree.py  
Firmware/all_flash/DeviceTree.n66ap.im4p.out Firmware/all_flash/DeviceTree.n66ap.im4p.out.mod
```

Now we have to set up the ram disk. First, ASN1 decode it:

```
Downloads jonathanafek$ python xnu-qemu-arm64-scripts/asn1rdskdecode.py ./048-32651-104.dmg ./048-32651-  
104.dmg.out
```

Next, resize it so it has room for the [dynamic loader cache file](#) (needed by bash and other executables), mount it, and force usage of file permissions on it:

```
Downloads jonathanafek$ hdiutil resize -size 1.5G -imagekey diskimage-class=CRawDiskImage 048-32651-  
104.dmg.out  
Downloads jonathanafek$ hdiutil attach -imagekey diskimage-class=CRawDiskImage 048-32651-104.dmg.out  
Downloads jonathanafek$ sudo diskutil enableownership /Volumes/PeaceB16B92.arm64UpdateRamDisk/
```

Now let's mount the regular update disk image by double clicking on it: [048-31952-103.dmg](#). Create a directory for the dynamic loader cache in the ram disk, copy the cache from the update image and chown it to root:

```
Downloads jonathanafek$ sudo mkdir -p  
/Volumes/PeaceB16B92.arm64UpdateRamDisk/System/Library/Caches/com.apple.dyld/  
Downloads jonathanafek$ sudo cp  
/Volumes/PeaceB16B92.N56N66OS/System/Library/Caches/com.apple.dyld/dyld_shared_cache_arm64  
/Volumes/PeaceB16B92.arm64UpdateRamDisk/System/Library/Caches/com.apple.dyld/  
Downloads jonathanafek$ sudo chown root  
/Volumes/PeaceB16B92.arm64UpdateRamDisk/System/Library/Caches/com.apple.dyld/dyld_shared_cache_arm64
```

Get precompiled user mode tools for iOS, including bash, from [rootlessJB](#) and/or [iOSBinaries](#). Alternatively, compile your own iOS console binaries as described [here](#).

```
Downloads jonathanafek$ git clone https://github.com/jakeajames/rootlessJB  
Cloning into 'rootlessJB'...  
remote: Enumerating objects: 6, done.  
remote: Counting objects: 100% (6/6), done.  
remote: Compressing objects: 100% (6/6), done.  
remote: Total 253 (delta 2), reused 0 (delta 0), pack-reused 247  
Receiving objects: 100% (253/253), 7.83 MiB | 3.03 MiB/s, done.  
Resolving deltas: 100% (73/73), done.
```

```
Downloads jonathanafek$ cd rootlessJB/rootlessJB/bootstrap/tars/  
tars jonathanafek$ tar xvf iosbinpack.tar  
tars jonathanafek$ sudo cp -R iosbinpack64 /Volumes/PeaceB16B92.arm64UpdateRamDisk/  
tars jonathanafek$ cd -
```

Configure launchd to not execute any services:

```
Downloads jonathanafek$ sudo rm /Volumes/PeaceB16B92.arm64UpdateRamDisk/System/Library/LaunchDaemons/*
```

And now, configure it to launch the interactive bash shell by creating a new file under [/Volumes/PeaceB16B92.arm64UpdateRamDisk/System/Library/LaunchDaemons/com.apple.bash.plist](#) with the following contents:

```
<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">  
<plist version="1.0">  
<dict>  
    <key>EnablePressuredExit</key>
```

```

<false/>
<key>Label</key>
<string>com.apple.bash</string>
<key>POSIXSpawnType</key>
<string>Interactive</string>
<key>ProgramArguments</key>
<array>
    <string>/iosbinpack64/bin/bash</string>
</array>
<key>RunAtLoad</key>
<true/>
<key>StandardErrorPath</key>
<string>/dev/console</string>
<key>StandardInPath</key>
<string>/dev/console</string>
<key>StandardOutPath</key>
<string>/dev/console</string>
<key>Umask</key>
<integer>0</integer>
<key>UserName</key>
<string>root</string>
</dict>
</plist>

```

As a side note, you can always convert the binary plist files that you find natively in iOS images to text xml format and back to binary format with:

```

Downloads jonathanafek$ plutil -convert xml1 file.plist
Downloads jonathanafek$ vim file.plist
Downloads jonathanafek$ plutil -convert binary1 file.plist

```

For launch daemon, iOS accepts both xml and binary plist files.

Since the new binaries are signed, but not by Apple, they need to be trusted by the static trust cache that we will create. To do this, we need to get [jtool](#) (also available via [Homebrew](#): `brew cask install jtool`). Once we have the tool, we have to run it on every binary we wish to be trusted, extract the first 40 characters of its CDHash, and put it in a new file named `tchashes`. A sample execution of `jtool` looks like this:

```

Downloads jonathanafek$ jtool --sig --ent /Volumes/PeaceB16B92.arm64UpdateRamDisk/iosbinpack64/bin/bash
Blob at offset: 1308032 (10912 bytes) is an embedded signature
Code Directory (10566 bytes)
    Version: 20001
    Flags: none
    CodeLimit: 0x13f580
    Identifier:
/Users/jakejames/Desktop/jelbreks/multi_path/multi_path/iosbinpack64/bin/bash (0x58)
    CDHash: 7ad4d4c517938b6fdc0f5241cd300d17fbb52418b1a188e357148f8369bacad1 (computed)
        # of Hashes: 320 code + 5 special
        Hashes @326 size: 32 Type: SHA-256
Empty requirement set (12 bytes)
Entitlements (279 bytes) :
-- 
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>platform-application</key>
    <true/>
    <key>com.apple.private.security.container-required</key>
    <false/>
</dict>
</plist>

```

In the above case, we need to write down `7ad4d4c517938b6fdc0f5241cd300d17fbb52418` in `tchashes`. For convenience, the following command will extract the correct part of the hash from each of the binaries we put in the image:

```
Downloads jonathanafek$ for filename in $(find /Volumes/PeaceB16B92.arm64UpdateRamDisk/iosbinpack64 -type f); do jtool --sig --ent $filename 2>/dev/null; done | grep CDHash | cut -d' ' -f6 | cut -c 1-40
ebe945ddb4dbeb1ee9624e6ba1932d2ec61cfde
7ad4d4c517938b6fdc0f5241cd300d17fbb52418
0cf1b00e3bf76ab51c56da7ca888e89359f1d1c4
c9c1e21c3f3593c99f4e7c91c64d7f3106ad29ce
522ddaa7f40fe6aa6e2038bc66c9cb31660a43429
dc040d340f1fcfb493394e77d9944aa164e23ca3
f975cd0eec230299d1b8d9b0e3b54ae7cf660d92
728be7f7a78f400742e887f7ac93306145f822c0
4f4ca5aa3e506d145f344d59504630b85ddefffc
0d274c72cefbbf705db0ed0fda29fb6f4cacf4c9
ebc9073fd59db7c59a5212b0824faf1d7b30e39
cf784ea216e6b49f66a3cc81aeceaf7ac39b71d7
9d625c7eaadc8fd3eb57d9facca294b1a5afab8a
90c02c153e636cac74ca09e7e3dc89c0508a1393
59cba1c5ce169d4cd454d43e3a3c6fa824cf2764
9ff1194d135e979a632033ec2df63ba0cfe4682a
d11b49576e0f6645c4c9f234497f51219173dce8
7a01f3e7bcd18b26297c3936c9e256ddf8f9fe3
b7fd47df9b6652f2810cc789d5903a082af2570d
68a32f0a35bbb23f4f272ca99186521618c08d21
e04fa65a33c4b69d2338688ee72ea13d624a4255
b400373e16a7f82fa56d318038ec7b4b28e2593f
65859385e11b910de3841e53a833ab4c4b855282
2eae1b42c4f6bb95e3226aff8cb93a539c0a6263
c305e094747ba274f37e3063b826a5e41e5e2549
41620d4632bf6f071388033f8cf267123df16489
3bf1f6c49e3bcd775041864085893bf9b1ab3870
bb2d9c166635fc693e99355e84984aa61692c6f3
3bb79fd3568c3620a2bd7bad004ab759bec4e331
7c60ae6060d7bf2772c6b4b0c04b605c4e62a7a7
b904a692d548c3323621c17212121aca0c733088
6fe1d88bcbdd97d273533d695c04279f8ddf5e32
4165a869f1b35bdff90b74116499c1c210f27ddb
414ebc5e48c94d60b2018e4c83a323426bc0ac74
62b2b303c31e5fc9d5210b736d8d632eee28d24f
871e0ea84b71cd01e45e261542e9b2dd08fb81ab
0912c647e222bd04f05b837a8286519bd8ae2393
bd6d7d7f51b639da99e0581096534273b4f040ed
27ed9a3b21392bc459619293a6b36fe2c3b8ddac
e92565cbfdb0bd41d069384689ffae715e61b216
164fc2d96f9decfd643ac33fc279b2078e51f5c88
3e0529b705d666af4f25c8c18fc7992f6934cf6f
176f273cb276085052519054d042508dc8d562b4
18762f5c54d935759f02248b032576bdc93be260
22d2f02d3be49da4819534553ad5ac37c0ace28c
e76bf6e8e84b656ee61b1ff10b38eab23607ae82
84bbc455477d6737f738b649c5af3d4a069abee
57fe14db863b48f19cdec3c884c5dfad1bf6a12
e6ee59194bd768c3e3cc140009b6a729c7700a11
f1c25d5ac4e3924deaa3418a9ba309e15c09f502
e962bfdead7da46f23b6f4dc448df085e946940
26d34ca63bc69c8e81c15672258f3b8cbaf4ba4c
7fc69d2fc1f57ca555b07d6de51c82f74915c6bd
85f3c5263835d90b776886f92e8536ceb2f46036
0f1214d8a6138f170c2654a6f81c40586fbeaac
dc995e91bc0b67c52b969c91c1d68b09bbf94ec2
5d46a9681b4a3cc84a69083288e76aa969ec3a43
3c0db01f7aaaf0a5b935dfcc51f6b2534013795ad
8422f07e41b2951e4138b88e013eab5773ae52f7
f9c4cca6b141064b7ae97131ff3969386d624718
259733b48f2f4fa88ba4f2e5f519bd40a6a3750d
8e06a919d28c3c0376b1207981d70b3bda99b6bc
68cd528c435b417c6f0022a132d459fc25d6e039
d176fa07a7ea5bfe88b9d2d703f3c65b4298b2e6
30f3d6e1d00614a0a9e8e8a3d4f31b8c68066091
698587325d71b9d51c22ae26e0c2de8ca70f6dc8
ccf27e4d7b62f1f839cfb9d70340efd1a2b77532
928a02f17cef27a5528ae055a467a18528f2aff5
4d24ada94fa70d27a684867541266f264261ce36
ab3e7808ee41f4536ece24091d1f166c5f0e9b63
e492332b87adc07406503ca857b6f3e2a3f0625d
```

d121b2de1778563183087238c4675316176f159d
12fe31a31132f7c0bab2857c0b3ac3c71cdb9dae
d6bc5428d129dd76695519b9b7f201daa9eb87de
685660477e1f851a90ace593670e5288d2168a24
94a493c2909f8b563e0076956bec7a1941455ed3
13c2e0251ba0469f2e1ec3d61da61c664822c791
e6332fc916f9b06f4987ecbaa23bbf4fa374c68f
1f6f82bcc994a4559d891d3a9e187268632da0b9
f864bd7891b9a0970f3ea05f13f7769289e62803
ba84abbeb198b91cbefec678096c8fd17387657d
d537ff6ab7d2bf38b0f18e964ad3525f2761b535
1acf88c15c1a08b3387b62969a34a95196632932
345d3b92a7f8a11c0872ec9ec439b5a6a2ada104
067b54e23cd6bc5b007113929dc4e2d2868228b6
11794790670afe1b651ed838362bb955e1503706
973674b1cf5f51119fa655ad2393df3dee9f44cc
c59738382faa4b7f803359d0c92dd53d6479ffb8
e3285e8252c44404675876ae0104f02cdc36574c
41c139fa86a3e67d49566d11a7d1d14fe375b564
b52692291cc4d9c9f09bc0ba650904d889674218
65713ffe304718b3b6a8b710b7db0467e52ca5aa
f2e77f5600970036ffd5a06067491c5799a2ebd
cb08034d4647f2cc921b62ea648a76b5635fcc13
a9fc0262a6925ec1c18b0bf627c04c60fa5b5ecf
3736f93cc5f88d138f58016fdce2c3c3af979c43
183cd29cea8ba53f6e5d28d87e37b0cc603106c6
cd0281c8fa808c3f0f0b74db8c262a6997f52d03
e3016edd7acfa4d24d2eacec4918f3018d9d2449
ddd943f2a4192b3eabb0580c64ff23ea7c31387
e3285e8252c44404675876ae0104f02cdc36574c
41c139fa86a3e67d49566d11a7d1d14fe375b564
b52692291cc4d9c9f09bc0ba650904d889674218
8af0e498ca73e05155f10fe7c26cfbdd9762ff24
73657606cb288c85f909da3ec4b92d7f8819ae79
918a3cf30a9c9d6ee2872c670421e528883221ae
dcf5eeaefc7ec3e7a0166676f6ee564761f78bc6
994ada738587ba622bfe36b987e9bfa246ff3858
d6f9c9107eb6dc237040d18debd4244c3e4c1320
f0e0c6a7e5c4545bac0d9ebf7811997f5c7076ad
38a790a40cca659fb8a0942ba140aa07309a17aa
070472831955773d78c9f33aff696c0a67b06bda
4ca98aac5e3b9174beaa2e4175e33fdcddee6866
44bd100692ded0637a763d324490db7435216f8c
a28a364092033230a6045fd288cb503aedbdd072
bbbe8ea84bdc4f3004398895ee58979a55b744c0
a09ee84582821397aa68d81350ed07b9902d09cb
8f8f612996a91e4fb26deacf2c88b8eda42da7a2
504d7c5b0a0e72a3dc5177ec571f591f3dae2ade
c0b0dea10a283f9d904bad52c53e20b129ae278c
5b089432710347242dfb6ccfdfea6fc523d9fe60
40af3f97ae3dc743f638c82f4ed78bce13687c83
7b3d463b62ce306c86d88e7ec0e52964c073c223
580eb965a96782a1fd005bd8a27100abc8430e1
330efc667ea608575d863b10a41a73e49f31d1c6
5827c3ef16144d298fd04342fc7041dd3b20d35e
f9bce1706a98b2492750aaa977806549f7d010f7
eeeaaeb163512c31c6462f41c6bc3b6a228224bee
2ae51c0fac8b5656ec91693e7f9846a9c4af8069
92c89c47a734cad1a36756155ea3043e406ae565
be0e71c532033d79d519951f0450cdca44f835c3
feff0ce891c71c69f581b19a70b30fffd4c407205
8b0f3f0c620f008d4b85b7aff69933d3aae6098e
296124c76c9f0201480678a012a1df2e6835c521
a1876907ad59843dc5ed1390c78c88698504b9d8
e3190fc3865f02092ab6725b25c485ea5c143e3b
8bbd9944ebc23ce2001a4837732ba082c040d0f4
6408ed0d9df71e7bdde2faa985e5c07911a43503
ca2b47f582135e00a9720215cc09881dd9b49b85
e7e478f2e7f9715d9b540c9f8d12993c83ece0c1
25ac265b51c484680decaf8903b0b3c12c5ff81c
5a37eb16c2eaba8dc55d9edb3ba98a0ee09af0

The above output should be saved in `tchashes`, and then we can create the static trust cache blob:

```
Downloads jonathanafek$ python xnu-qemu-arm64-scripts/create_trustcache.py tchashes static_tc
```

Now is a good time to unmount both volumes. We now have all the images and files prepared. Let's get the modified QEMU code (more detailed info on the work done in QEMU will be in the second post in the series):

```
Downloads jonathanafek$ git clone git@github.com:alephsecurity/xnu-qemu-arm64.git
Cloning into 'xnu-qemu-arm64'...
remote: Enumerating objects: 377340, done.
remote: Total 377340 (delta 0), reused 0 (delta 0), pack-reused 377340
Receiving objects: 100% (377340/377340), 187.68 MiB | 5.32 MiB/s, done.
Resolving deltas: 100% (304400/304400), done.
Checking out files: 100% (6324/6324), done.
```

and compile it:

```
Downloads jonathanafek$ cd xnu-qemu-arm64
xnu-qemu-arm64 jonathanafek$ ./configure --target-list=aarch64-softmmu --disable-capstone
Install prefix      /usr/local
BIOS directory     /usr/local/share/qemu
firmware path      /usr/local/share/qemu-firmware
binary directory   /usr/local/bin
library directory  /usr/local/lib
module directory   /usr/local/lib/qemu
libexec directory  /usr/local/libexec
include directory  /usr/local/include
config directory   /usr/local/etc
local state directory /usr/local/var
Manual directory   /usr/local/share/man
ELF interp prefix  /usr/gnemul/qemu-%M
Source path        /Users/jonathanafek/Downloads/xnu-qemu-arm64
GIT binary         git
GIT submodules     ui/keycodemapdb dtc
C compiler          cc
Host C compiler    cc
C++ compiler       c++
Objective-C compiler clang
ARFLAGS            rv
CFLAGS             -O2 -g
QEMU_CFLAGS        -I/opt/local/include/pixman-1 -I$(SRC_PATH)/dtc/libfdt -D_REENTRANT -
I/opt/local/include/glib-2.0 -I/opt/local/lib/glib-2.0/include -I/opt/local/include -m64 -mcx16 -
DOS_OBJECT_USE_OBJC=0 -arch x86_64 -D_GNU_SOURCE -D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE -Wstrict-
prototypes -Wredundant-decls -Wall -Wundef -Wwrite-strings -Wmissing-prototypes -fno-strict-aliasing -
fno-common -fwrapv -Wno-error=address-of-packed-member -Wno-string-plus-int -Wno-initializer-overrides -
Wexpansion-to-defined -Wendif-labels -Wno-shift-negative-value -Wno-missing-include-dirs -Wempty-body -
Wnested-externs -Wformat-security -Wformat-y2k -Winit-self -Wignored-qualifiers -Wold-style-definition -
Wtype-limits -fstack-protector-strong -I/opt/local/include -I/opt/local/include/p11-kit-1 -
I/opt/local/include -I/opt/local/include/libpng16 -I/opt/local/include
LDFLAGS            -framework Hypervisor -m64 -framework CoreFoundation -framework IOKit -arch x86_64 -g
QEMU_LDFLAGS       -L$(BUILD_DIR)/dtc/libfdt
make               make
install            install
python             python -B
smbd              /usr/sbin/smbd
module support    no
host CPU          x86_64
host big endian   no
target list        aarch64-softmmu
gprof enabled     no
sparse enabled    no
strip binaries   yes
profiler          no
static build      no
Cocoa support     yes
SDL support       no
GTK support       no
GTK GL support    no
VTE support       no
```

```
TLS priority      NORMAL
GNUTLS support   yes
GNUTLS rnd       yes
libgcrypt        no
libgcrypt kdf    no
nettle           yes (3.4.1)
nettle kdf       yes
libtasn1         yes
curses support   yes
virgl support    no
curl support     yes
mingw32 support  no
Audio drivers    coreaudio
Block whitelist (rw)
Block whitelist (ro)
VirtFS support   no
Multipath support no
VNC support      yes
VNC SASL support yes
VNC JPEG support no
VNC PNG support  yes
xen support      no
brlapi support   no
bluez support    no
Documentation    yes
PIE              no
vde support      no
netmap support   no
Linux AIO support no
ATTR/XATTR support no
Install blobs    yes
KVM support      no
HAX support      yes
HVF support      yes
WHPX support     no
TCG support      yes
TCG debug enabled no
TCG interpreter   no
malloc trim support no
RDMA support     no
fdt support      git
membarrier       no
preadv support   no
fdatasync        no
madvise          yes
posix_madvise    yes
posix_memalign   yes
libcap-ng support no
vhost-net support no
vhost-crypto support no
vhost-scsi support no
vhost-vsock support no
vhost-user support yes
Trace backends   log
spice support    no
rbd support      no
xfsctl support   no
smartcard support no
libusb           no
usb net redir    no
OpenGL support   no
OpenGL dmabufs   no
libiscsi support no
libnfs support   no
build guest agent yes
QGA VSS support  no
QGA w32 disk info no
QGA MSI support  no
seccomp support  no
coroutine backend sigaltstack
coroutine pool    yes
debug stack usage no
mutex debugging  no
```

```

crypto afalg      no
GlusterFS support no
gcov           gcov
gcov enabled    no
TPM support     yes
libssh2 support  no
TPM passthrough no
TPM emulator    yes
QOM debugging   yes
Live block migration yes
lzo support     no
snappy support  no
bzip2 support   yes
NUMA host support no
libxml2          yes
tcmalloc support no
jemalloc support no
avx2 optimization no
replication support yes
VxHS block device no
capstone         no
docker          no

```

```

xnu-qemu-arm64 jonathanafek$ make -j16
xnu-qemu-arm64 jonathanafek$ cd -

```

And all there's left to do is execute:

```

Downloads jonathanafek$ ./xnu-qemu-arm64/aarch64-softmmu/qemu-system-aarch64 -M iPhone6splus-n66-
s8000,kernel-filename=kernelcache.release.n66.out,dtb-
filename=Firmware/all_flash/DeviceTree.n66ap.im4p.out.mod,secmon-filename=securemonitor.out,ramdisk-
filename=048-32651-104.dmg.out,tc-filename=static_tc,kern-cmd-args="debug=0x8 kextlog=0xffff cpus=1 rd=md0
serial=2" -cpu max -m 6G -serial mon:stdio
iBoot version:
corecrypto_kext_start called
FIPSPOST_KEXT [38130750] fipspost_post:156: PASSED: (6 ms) - fipspost_post_integrity
FIPSPOST_KEXT [38201250] fipspost_post:162: PASSED: (2 ms) - fipspost_post_hmac
FIPSPOST_KEXT [38233562] fipspost_post:163: PASSED: (0 ms) - fipspost_post_aes_ecb
FIPSPOST_KEXT [38275375] fipspost_post:164: PASSED: (1 ms) - fipspost_post_aes_cbc
FIPSPOST_KEXT [41967250] fipspost_post:165: PASSED: (153 ms) - fipspost_post_rsa_sig
FIPSPOST_KEXT [44373250] fipspost_post:166: PASSED: (99 ms) - fipspost_post_ecdsa
FIPSPOST_KEXT [44832437] fipspost_post:167: PASSED: (18 ms) - fipspost_post_ecdh
FIPSPOST_KEXT [44861312] fipspost_post:168: PASSED: (0 ms) - fipspost_post_drbg_ctr
FIPSPOST_KEXT [44922625] fipspost_post:169: PASSED: (2 ms) - fipspost_post_aes_ccm
FIPSPOST_KEXT [44994250] fipspost_post:171: PASSED: (2 ms) - fipspost_post_aes_gcm
FIPSPOST_KEXT [45042125] fipspost_post:172: PASSED: (1 ms) - fipspost_post_aes_xts
FIPSPOST_KEXT [45109687] fipspost_post:173: PASSED: (2 ms) - fipspost_post_tdes_cbc
FIPSPOST_KEXT [45167062] fipspost_post:174: PASSED: (1 ms) - fipspost_post_drbg_hmac
FIPSPOST_KEXT [45178250] fipspost_post:197: all tests PASSED (300 ms)
Darwin Image4 Validation Extension Version 1.0.0: Tue Oct 16 21:46:27 PDT 2018; root:AppleImage4-
1.200.18~1853/AppleImage4/RELEASE_ARM64
AppleS8000IO::start: chip-revision: A0
AppleS8000IO::start: this: <ptr>, TCC virt addr: <ptr>, TCC phys addr: 0x202240000
AUC[<ptr>]::init(<ptr>)
AUC[<ptr>]::probe(<ptr>, <ptr>)
AppleCredentialManager: init: called, instance = <ptr>.
ACMRM: init: called, ACMDRM_ENABLED=YES, ACMDRM_STATE_PUBLISHING_ENABLED=YES,
ACMDRM_KEYBAG_OBSERVING_ENABLED=YES.
ACMRM: _loadRestrictedModeForceEnable: restricted mode force-enabled = 0 .
ACMRM-A: init: called, .
ACMRM-A: _loadAnalyticsCollectionPeriod: analytics collection period = 86400 .
ACMRM: _loadStandardModeTimeout: standard mode timeout = 259200 .
ACMRM-A: notifyStandardModeTimeoutChanged: called, value = 259200 (modified = YES).
ACMRM: _loadGracePeriodTimeout: device lock timeout = 3600 .
ACMRM-A: notifyGracePeriodTimeoutChanged: called, value = 3600 (modified = YES).
AppleCredentialManager: init: returning, result = true, instance = <ptr>.
AUC[<ptr>]::start(<ptr>)
virtual bool AppleARMLightEmUp::start(IOService *): starting...
AppleKeyStore starting (BUILT: Oct 17 2018 20:34:07)
AppleSEPKeyStore::start: _sep_enabled = 1
AppleCredentialManager: start: called, instance = <ptr>.
ACMRM: _publishIOResource: AppleUSBRestrictedModeTimeout = 259200.

```

```

AppleCredentialManager: start: initializing power management, instance = <ptr>.
AppleCredentialManager: start: started, instance = <ptr>.
AppleCredentialManager: start: returning, result = true, instance = <ptr>.
AppleARMPE::getGMTTimeOfDay can not provide time of day: RTC did not show up
: apfs_module_start:1277: load: com.apple.filesystems.apfs, v748.220.3, 748.220.3, 2018/10/16
com.apple.AppleFSCo...pressionTypeZlib kmod start
IOSurfaceRoot::installMemoryRegions()
IOSurface disallowing global lookups
apfs_sysctl_register:911: done registering sysctls.
com.apple.AppleFSCo...ressionTypeZlib load succeeded
L2TP domain init
L2TP domain init complete
PPTP domain init
BSD root: md0, major 2, minor 0
apfs_vfsop_mountroot:1468: apfs: mountroot called!
apfs_vfsop_mount:1231: unable to root from devvp <ptr> (root_device): 2
apfs_vfsop_mountroot:1472: apfs: mountroot failed, error: 2
hfs: mounted PeaceB16B92.arm64UpdateRamDisk on device b(2, 0)
: : Darwin Bootstrapper Version 6.0.0: Tue Oct 16 22:26:06 PDT 2018; root:libxpc_executables-1336.220.5~209/launchd/RELEASE_ARM64
boot_args = debug=0x8 kextlog=0xffff cpus=1 rd=md0 serial=2
Thu Jan 1 00:01:05 1970 localhost com.apple.xpc.launchd[1] <Notice>: Restore environment starting.
Thu Jan 1 00:01:05 1970 localhost com.apple.xpc.launchd[1] <Notice>: Early boot complete. Continuing system boot.
Thu Jan 1 00:01:06 1970 localhost com.apple.xpc.launchd[1] (com.apple.xpc.launchd.domain.system)
<Error>: Could not read path: path = /AppleInternal/Library/LaunchDaemons, error = 2: No such file or directory
Thu Jan 1 00:01:06 1970 localhost com.apple.xpc.launchd[1] (com.apple.xpc.launchd.domain.system)
<Error>: Could not read path: path = /System/Library/NanoLaunchDaemons, error = 2: No such file or directory
Thu Jan 1 00:01:06 1970 localhost com.apple.xpc.launchd[1] (com.apple.xpc.launchd.domain.system)
<Error>: Failed to bootstrap path: path = /System/Library/NanoLaunchDaemons, error = 2: No such file or directory
bash-4.4# export
PATH=$PATH:/iosbinpack64/usr/bin:/iosbinpack64/bin:/iosbinpack64/usr/sbin:/iosbinpack64/sbin
bash-4.4# id
uid=0(root) gid=0(wheel)
groups=0(wheel),1(daemon),2(kmem),3(sys),4(tty),5(operator),8(procview),9(procmod),20(staff),29(certusers ),80(admin)
bash-4.4# pwd
/
bash-4.4# ls -la
total 18
drwxr-xr-x 17 root      wheel   748 Jun 10  2019 .
drwxr-xr-x 17 root      wheel   748 Jun 10  2019 ..
-rw-r--r--  1 root      wheel     0 Oct 20  2018 .Trashes
drwx----- 2 mobile    staff   170 Jun 10  2019 .fsevents.d
drwxr-xr-x  4 root      wheel   136 Oct 20  2018 System
drwxr-xr-x  2 root      wheel   272 Oct 20  2018 bin
dr-xr-xr-x  3 root      wheel   660 Jan  1 00:01 dev
lrwxr-xr-x  1 root      wheel   11 Oct 20  2018 etc -> private/etc
drwxr-xr-x  7 root      wheel   374 Jun 10  2019 iosbinpack64
drwxr-xr-x  2 root      wheel   68 Oct 20  2018 mnt1
drwxr-xr-x  2 root      wheel   68 Oct 20  2018 mnt2
drwxr-xr-x  2 root      wheel   68 Oct 20  2018 mnt3
drwxr-xr-x  2 root      wheel   68 Oct 20  2018 mnt4
drwxr-xr-x  2 root      wheel   68 Oct 20  2018 mnt5
drwxr-xr-x  2 root      wheel   68 Oct 20  2018 mnt6
drwxr-xr-x  2 root      wheel   68 Oct 20  2018 mnt7
drwxr-xr-x  4 root      wheel   136 Oct 20  2018 private
drwxr-xr-x  2 root      wheel   510 Oct 20  2018 sbin
drwxr-xr-x  9 root      wheel   306 Oct 20  2018 usr
lrwxr-xr-x  1 root      admin   11 Oct 20  2018 var -> private/var
bash-4.4#

```

And we have an interactive bash shell! :)

Note that the last flag (`-serial mon:stdio`) will forward all shell combinations (such as `[Ctrl]+[C]`) to the shell. To shut down QEMU, close its (empty) window.

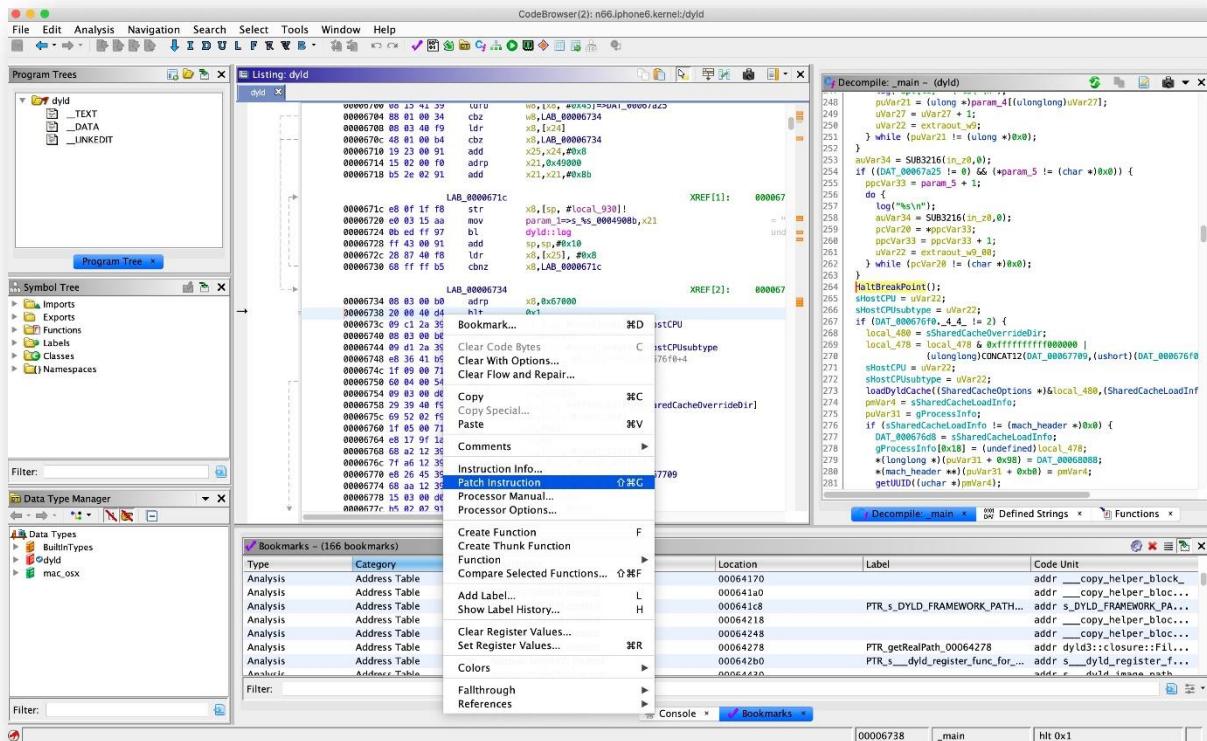
To get a kernel debugger, `-S -s` should be added to the QEMU command line, and then it's possible to execute `target remote :1234` in a gdb console which supports this architecture. More details on how to get this gdb and perform this can be found in [here](#). You can also get the relevant gdb on OSX with [mac ports](#), while adding the `multiarch` and `python27` options to the gdb port.

Future Improvements

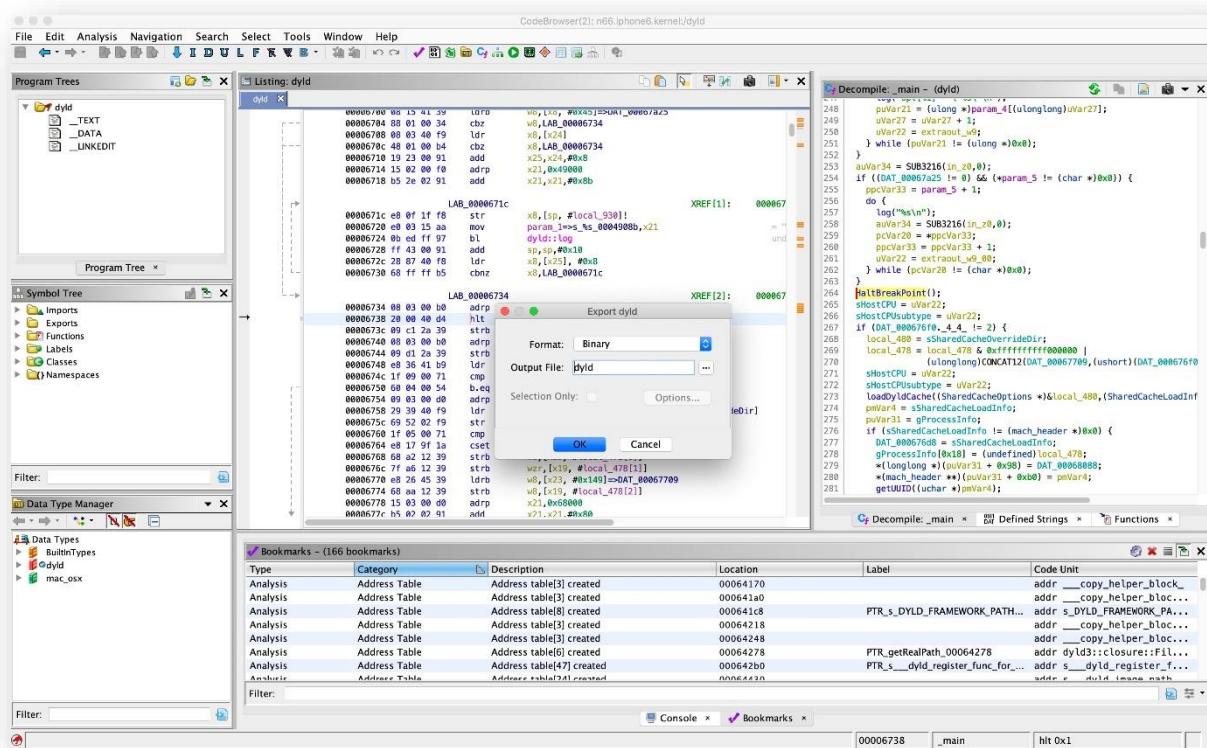
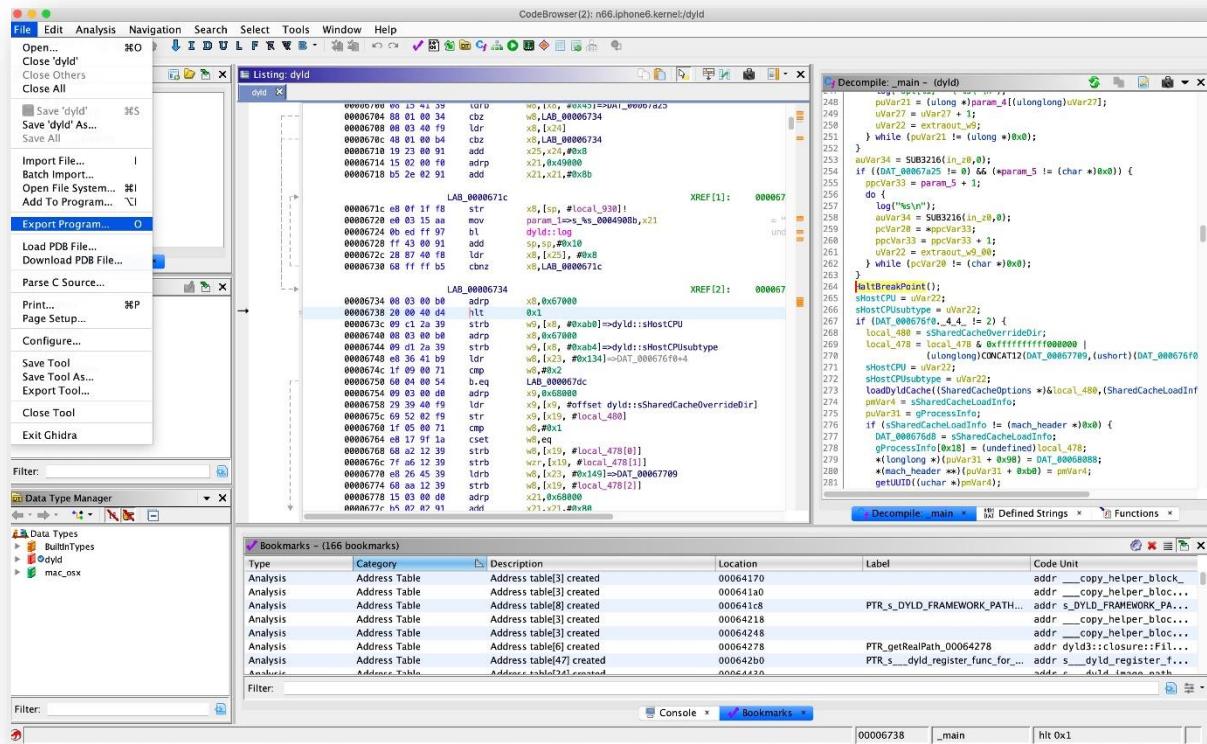
1. Make this boot much faster, without a long suspension before mounting the ram disk.
2. Add support to emulate the iOS as a USB device and communicate over usbmuxd. This will enable us to connect over SSH and therefore copy files using scp, have a more robust terminal, conduct security research for network protocols, use [gdbserver](#) to debug user mode applications and more.
3. Add support for emulated physical storage to work with a r/w mounted disk, that won't be a ram disk and will not be limited to 2GB.
4. Add support for devices such as screen, touch, wifi, BT, etc...
5. Add support for more iDevices and iOS versions.

Fun Feature

User applications get loaded at different addresses every boot because of ASLR, and can share virtual addresses with one another, so using a regular breakpoint on a static virtual address in gdb can be challenging, when debugging user mode applications. Therefore, I added another fun feature to help debug user mode applications in this kernel debugger. When QEMU encounters the HLT aarch64 instruction, it breaks in gdb just as if it was a gdb breakpoint, so all you have to do to debug user mode applications in the kernel debugger is to patch the application with an HLT instruction, using ghidra,



for example:



and then sign it using jtool with any required entitlements:

```
Downloads jonathanafek$ ./jtool/jtool --sign --ent ent.xml --inplace bin
```

After that, you need to add the new CDHash to the `tchashes` file, and recreate the static trust cache.

Following that, gdb will break when it encounters the HLT instruction in the user mode application so we can debug the application in the kernel debugger:

Running iOS in QEMU to an interactive bash shell (2): research

By [Jonathan Afek \(@JonathanAfek\)](#)

June 25, 2019

This is the second post in a 2-post series about the work I did to boot a non-patched iOS 12.1 kernel on QEMU emulating iPhone 6s plus and getting an interactive bash shell on the emulated iPhone. To see the code and the explanation on how to use it, please refer to the [first post](#). In this post, I will present some of the research that was done in order to make this happen. This research was based on the work done by [zhuowei](#) as a starting point. Based on the work done by zhuowei, I already had a way to boot an iOS kernel of a slightly different version on a different iPhone without a secure monitor, while patching the kernel at runtime to make it boot, running the launchd services that pre-existed on the ramdisk image and without interactive I/O. In this post I will present:

1. How the code was inserted in the QEMU project as a new machine type.
2. How the kernel was booted without patching the kernel either at runtime or beforehand.
3. How the secure monitor image was loaded and executed in EL3.
4. How a new static trust cache was added so self signed executables could be executed.
5. How a new launchd item was added for executing an interactive shell instead of the existing services on the ramdisk.
6. How full serial I/O was established.

The project is now available at [qemu-aleph-git](#) with the required scripts at [qemu-scripts-aleph-git](#).

QEMU code

In order to be able to later rebase the code on more recent versions of QEMU and to add support for other iDevices and iOS versions, I moved all the QEMU code changes into new modules. I now have the module `hw/arm/n66_iphone6splus.c` that is the main module for the iPhone 6s plus (n66ap) iDevice in QEMU that is responsible for:

1. Define a new machine type.
2. Define the memory layout of the UART memory mapped I/O, the loaded kernel, secure monitor, boot args, device tree, trust cache in different exception levels' memory.
3. Define the iDevice's proprietary registers (currently do nothing and just operate as general purpose registers).
4. Define the machine's capabilities and properties like having EL3 support and start execution in it at the secure monitor entry point.
5. Connect the builtin timer interrupt to FIQ.
6. Get command line parameters for defining the files for: kernel image, secure monitor image, device tree, ramdisk, static trust cache, kernel boot args.

The other main module is `hw/arm/xnu.c` and it is responsible for:

1. Loading the device tree into memory and adding the ramdisk and the static trust cache addresses to the device tree where they are actually loaded.

2. Loading the ramdisk into memory.
3. Loading the static trust cache into memory.
4. Loading the kernel image into memory.
5. Loading the secure monitor image into memory.
6. Loading and setting the kernel and secure monitor boot args.

Booting the kernel with no patches

Basing the work on [zhuowei](#) I already had the ability to boot to user mode with a different iOS version and for a different iPhone while patching the kernel at runtime using the kernel debugger. The need to patch comes from the fact that after changing the device tree and booting from a ramdisk we encounter a non-returning function waiting for an event that never comes. By placing breakpoints all over and single stepping in the kernel debugger I found that the non-returning function is `[IOSecureBSDRoot()` which can be found in the [XNU code published by Apple](#) in version `xnu-4903.221.2`:

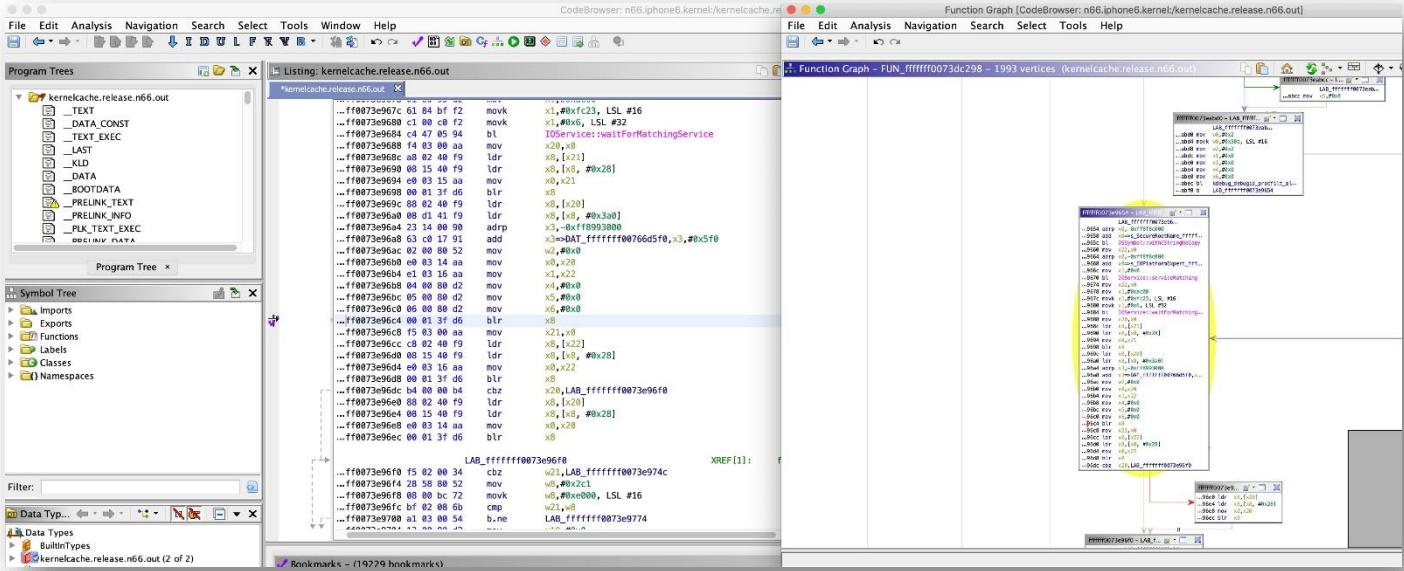
The screenshot shows the XNU kernel source code for the `IOSecureBSDRoot()` function. The left pane displays the C code, and the right pane shows the corresponding assembly code. A red vertical line highlights the assembly code, indicating where the debugger is currently executing. The assembly code is highly optimized, using various instructions like `ldr`, `add`, `mov`, and `blr`. The assembly code starts with a series of `...ff0073e96a` instructions, followed by `...ff0073e96b`, and continues with `...ff0073e96c`, `...ff0073e96d`, and `...ff0073e96e`.

```

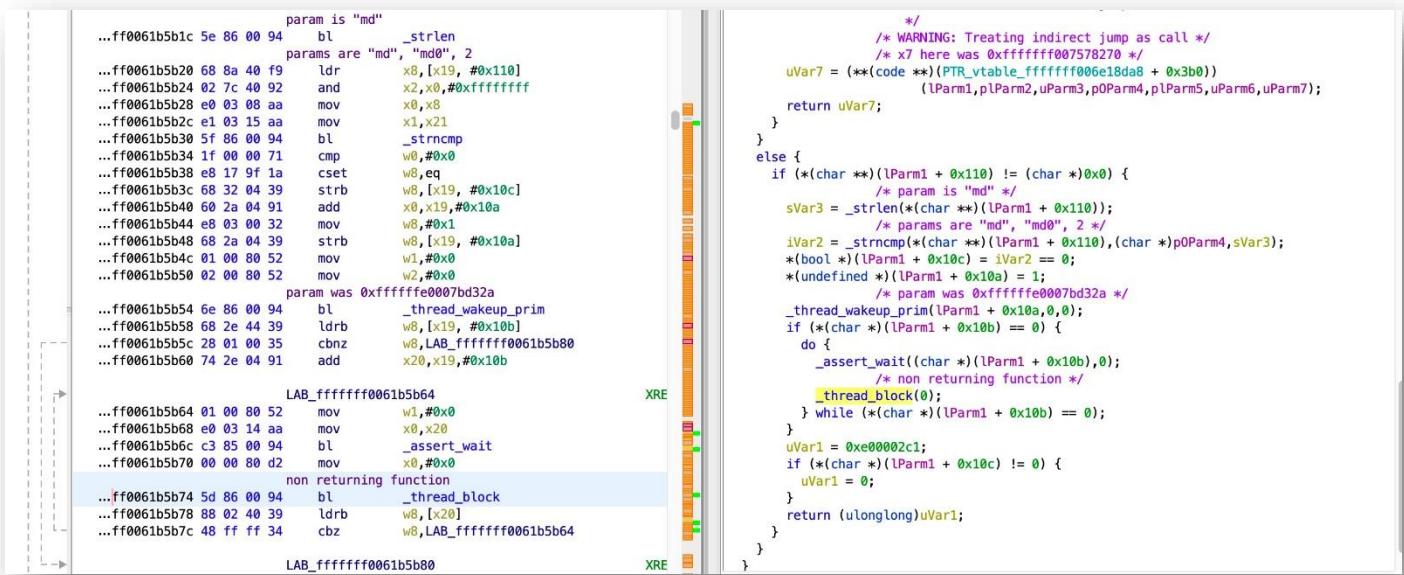
void IOSecureBSDRoot(const char * rootName)$
{
    __PLK_TEXT_EXEC
#if CONFIG_EMBEDDED$NIV DATA
    int tmpInt;$NIV DATA
    IOReturn result;$NIV DATA
    IOPlatformExpert *pe;$NIV DATA
    OSDictionary *matching;$NIV DATA
    const OSSymbol *functionName = OSSymbol::withCStringNoCopy("SecureRootName");$NIV DATA
$NIV DATA
    matching = IOService::serviceMatching("IOPlatformExpert");$NIV DATA
    assert(matching);$NIV DATA
    pe = (IOPlatformExpert *) IOService::waitForMatchingService(matching, 30ULL*kSecondScale);$NIV DATA
    matching->release();$NIV DATA
    assert(pe);$NIV DATA
    // Returns kIOReturnNotPrivileged if the root device is not secure.$NIV DATA
    // Returns kIOReturnUnsupported if "SecureRootName" is not implemented.$NIV DATA
    result = pe->callPlatformFunction(functionName, false, (void *)rootName, (void *)0, (void *)0, (void *)0);$NIV DATA
    functionName->release();$NIV DATA
    OSSafeReleaseNULL(pe);$NIV DATA
$NIV DATA
    if (result == kIOReturnNotPrivileged) {
        mdevremoveall();$NIV DATA
    } else if (result == kIOReturnSuccess) {
        // If we are booting with a secure root, and we have the right$NIV DATA
>----// boot-arg, we will want to panic on exception triage. This$NIV DATA
>----// behavior is intended as a debug aid (we can look at why an$NIV DATA
>----// exception occurred in the kernel debugger).$NIV DATA
    if (PE_parse_boot_argn("-panic_on_exception_trage", &tmpInt, sizeof(tmpInt)))$NIV DATA
        panic_on_exception_trage = 1;$NIV DATA
$NIV DATA
#endif //CONFIG_EMBEDDED$NIV DATA
}#$NIV DATA
kernelcache_release.n66.out (2 of 2)

```

and at runtime, when debugging the kernel itself:



This function doesn't return, because the call to `pe->callPlatformFunction()` doesn't return. For this function I don't have any reference code, so the kernel is disassembled:



By examining the function, we can see that the non-returning function deals a lot with specific members of the object in `x19`, and the flow changes depending on these members. I tried a few approaches to understand what these members mean, and where their values are set, but with no luck. These members do seem to be at special offsets, so after a while I tried my luck and used Ghidra to search the whole kernel for functions that use objects and their members at offsets `0x10a`, `0x10c` and `0x110` - and I got lucky! I found this function, that deals with the exact same object and these members:

```

...ff0061b57e0 08 89 40 f9    ldr      x8,[x8, #0x110]
...ff0061b57e4 e1 d6 ff 90    adr     x1,-0xffa36f000
...ff0061b57e8 21 44 17 91    add     x1,x1,#0x5d1
...ff0061b57ec e0 03 14 aa    mov     x0,x20
...ff0061b57f0 00 01 3f d6    call to IORegistryEntry::getProperty() - 0xffffffff
...ff0061b57f4 08 63 00 f0    blr      x8
...ff0061b57f8 08 0d 46 f9    adr     x8,[x8, #0xc18]>>->OSData::metaClass
...ff0061b57fc 01 01 40 f9    ldr     x1=>OSData::gMetaClass,[x8]>>->OSData
...ff0061b5800 d2 85 00 94    bl      safeMetaCast
...ff0061b5804 c0 00 00 b4    cbz    x0,LAB_foooo0061b581c
...ff0061b5808 08 00 40 f9    ldr     x8,[x0]
...ff0061b580c 08 69 40 f9    ldr     x8,[x8, #0xd0]
...ff0061b5810 00 01 3f d6    blr      x8
...ff0061b5814 60 8a 00 f9    ptr     x0,[x19, #0x110]
...ff0061b5818 04 00 00 14    b      LAB_foooo0061b5828

        LAB_foooo0061b581c
...ff0061b581c 75 32 04 39    strb   w21,[x19, #0x10c]
...ff0061b5820 28 20 80 52    mov    w8,#0x101
...ff0061b5824 68 16 02 79    strh   w8,[x19, #0x10a]

        LAB_foooo0061b5828
...ff0061b5828 88 02 40 f9    ldr     x8,[x20]
...ff0061b582c 08 89 40 f9    ldr     x8,[x8, #0x110]
...ff0061b5830 e1 d6 ff 90    adr     x1,-0xffa36f000
...ff0061b5834 21 90 17 91    add     x1,x1,#0x5e4
...ff0061b5838 e0 03 14 aa    mov     x0,x20
...ff0061b583c 00 01 3f d6    blr      x8
...ff0061b5840 60 00 00 b4    cbz    x0,LAB_foooo0061b581c

lVar1 = _IOLockAlloc();
plParm1[0x1f] = lVar1;
lVar1 = withCapacity(OSDictionary *)@x1,extraout_w1;
plParm1[0x20] = lVar1;
/* call to IOPlatformExpert::setBootROMType() - 0xfffffffff007578394 */
(**(code **)(*plParm1 + 0x538))(plParm1,1);
/* call to IORegistryEntry::getProperty() - 0xfffffffff00752d2a8 */
p0Var2 = (OSMetaClassBase *)(**(code **)(*plParm2 + 0x110))(plParm2,"secure-root-prefix")
plVar3 = (longlong *)safeMetaCast(p0Var2,(OSMetaClass **)PTR_metaClass_foooo0061b581c
if (plVar3 == (longlong *)0x0) {
    *(undefined *)((longlong)plParm1 + 0x10c) = 1;
    *(undefined *)((longlong)plParm1 + 0x10a) = 0x101;
}
else {
    lVar1 = (**(code **)(*plVar3 + 0xd0))();
    plParm1[0x22] = lVar1;
}
lVar1 = (**(code **)(*plParm2 + 0x110))(plParm2,&UNK_foooo005c915e4);
if (!lVar1 != 0) {
    *(undefined *)((longlong)plParm1 + 0x21) = 1;
}
lVar1 = (**(code **)(*plParm2 + 0x110))(plParm2,"no-suspend");
if (!lVar1 != 0) {
    *(undefined *)((longlong)plParm1 + 0x109) = 1;
}
uVar4 = (**(code **)(PTR_vtable_foooo0061b581c + 0x2b8))(plParm1,plParm2);
if ((int)uVar4 != 0) {
    FUN_foooo0061ae668(plParm2,plParm1);
    (**(code **)(*plParm1 + 0x298))(plParm1,0);
}

```

In this function, it is easy to see that when the prop `secure-root-prefix` is not in the device tree, the member at offset 0x110 is kept untouched with the value 0, and then the original function I looked at (`[pe->callPlatformFunction()`) just returns, and voila - there no need to patch the kernel.

Loading the secure monitor image

Where we left off with the work done by [zhuowei](#), I was able to boot an iPhone X image to user mode. This image boots straight into EL1 and has no secure monitor. I decided to work with another image for iPhone 6s plus, as Apple left off a lot of symbols in there and I thought it would make the research simpler. It turns out that [KPP](#) (Kernel Patch Protection) based devices with no [KTRR](#) (Kernel Text Readonly Region) have a secure monitor image that needs to be loaded with its own boot args, and executed in EL3. This part of the project was about finding the secure monitor image embedded in the kernel file, loading it, understanding the boot args structure, and loading the image and configuring QEMU to start executing the entry point in EL3. After completing these steps, things still didn't work. The reason seemed to be that the secure monitor image tries to parse the kernel mach-o header at the kernel base (read from the kernel boot args), and I didn't have a kernel image at that base address, so it failed with a data-abort. This all happens in this function:

```

void FUN_ffff004100005174(ulonglong **ppuParm1, longlong lParm2)
{
    undefined8 auVar1 [16];
    undefined8 auVar2 [16];
    longlong lVar3;
    int memcmp_res;
    int memcmp_res1;
    int iVar4;
    ulonglong **local_68;
    ulonglong *vmaaddr_of_segment;
    undefined8 *puVar5;
    ulonglong *size_of_segment;
    ulonglong uVar6;
    ulonglong *vmaaddr_seg;
    ulonglong *puVar7;
    ulonglong *vmsize_seg;
    ulonglong end_of_seg_offset_from_kern;
    ulonglong *puVar8;
    ulonglong **ppuVar9;
    ulonglong **iter_ptr_to_seg_data;
    ulonglong uVar10;
    longlong *pIVar11;
    longlong lVar12;
    ulonglong uVar13;
    ulonglong uVar14;
    int *load_command_ptr;
    uint current_load_command_count;
    undefined in_z8 [32];
    longlong base_of_kernel_ptr;
    int *name_of_segment_ptr;
    ulonglong **ptr_to_seg_data;
    ulonglong **ptr_to_seg_data+8;
}

```

I believe this function is responsible for the KPP functionality because it keeps a map of kernel sections based on the permissions they should have, but this assumption still needs to be verified.

As can be seen in the code from [zhuowei](#), the `virt_base` arg was pointing to the lowest segment of the loaded kernel:

```

static uint64_t arm_load_macho(struct arm_boot_info *info, uint64_t *pentry, AddressSpace *as)
{
    hwaddr kernel_load_offset = 0x00000000;
    hwaddr mem_base = info->loader_start;

    uint8_t *data = NULL;
    gsize len;
    bool ret = false;
    uint8_t* rom_buf = NULL;
    if (!g_file_get_contents(info->kernel_filename, (char**) &data, &len, NULL)) {
        goto out;
    }
    struct mach_header_64* mh = (struct mach_header_64*)data;
    struct load_command* cmd = (struct load_command*)(data + sizeof(struct mach_header_64));
    // iterate through all the segments once to find highest and lowest addresses
    uint64_t pc = 0;
    uint64_t low_addr_temp;
    uint64_t high_addr_temp;
    macho_highest_lowest(mh, &low_addr_temp, &high_addr_temp);
    uint64_t rom_buf_size = high_addr_temp - low_addr_temp;
    rom_buf = g_malloc0(rom_buf_size);
    for (unsigned int index = 0; index < mh->ncmds; index++) {
        switch (cmd->cmd) {
            case LC_SEGMENT_64: {
                struct segment_command_64* segCmd = (struct segment_command_64*)cmd;
                memcpy(rom_buf + (segCmd->vmaddr - low_addr_temp), data + segCmd->fileoff, segCmd->filesize);
                break;
            }
            case LC_UNIXTHREAD: {
                // grab just the entry point PC
                uint64_t* ptrPc = (uint64_t*)((char*)cmd + 0x110); // for arm64 only.
                pc = VAtopA(*ptrPc);
                break;
            }
        }
    }
}

```

```

        cmd = (struct load_command*)((char*)cmd + cmd->cmdszie);
    }
    hwaddr rom_base = VAtoPA(low_addr_temp);
    rom_add_blob_fixed_as("macho", rom_buf, rom_buf_size, rom_base, as);
    ret = true;

    uint64_t load_extra_offset = high_addr_temp;
    uint64_t ramdisk_address = load_extra_offset;
    gsize ramdisk_size = 0;

    // load ramdisk if exists
    if (info->initrd_filename) {
        uint8_t* ramdisk_data = NULL;
        if (g_file_get_contents(info->initrd_filename, (char**) &ramdisk_data, &ramdisk_size, NULL)) {
            info->initrd_filename = NULL;
            rom_add_blob_fixed_as("xnu_ramdisk", ramdisk_data, ramdisk_size, VAtoPA(ramdisk_address),
as);
            load_extra_offset = (load_extra_offset + ramdisk_size + 0xfffffull) & ~0xfffffull;
            g_free(ramdisk_data);
        } else {
            fprintf(stderr, "ramdisk failed?!\n");
            abort();
        }
    }

    uint64_t dtb_address = load_extra_offset;
    gsize dtb_size = 0;
    // load device tree
    if (info->dtb_filename) {
        uint8_t* dtb_data = NULL;
        if (g_file_get_contents(info->dtb_filename, (char**) &dtb_data, &dtb_size, NULL)) {
            info->dtb_filename = NULL;
            if (ramdisk_size != 0) {
                macho_add_ramdisk_to_dtb(dtb_data, dtb_size, VAtoPA(ramdisk_address), ramdisk_size);
            }
            rom_add_blob_fixed_as("xnu_dtb", dtb_data, dtb_size, VAtoPA(dtb_address), as);
            load_extra_offset = (load_extra_offset + dtb_size + 0xfffffull) & ~0xfffffull;
            g_free(dtb_data);
        } else {
            fprintf(stderr, "dtb failed?!\n");
            abort();
        }
    }

    // fixup boot args
    // note: device tree and args must follow kernel and be included in the kernel data size.
    // macho_setup_bootargs takes care of adding the size for the args
    // osfmk/arm64/arm_vm_init.c:arm_vm_prot_init
    uint64_t bootargs_addr = VAtoPA(load_extra_offset);
    uint64_t phys_base = (mem_base + kernel_load_offset);
    uint64_t virt_base = low_addr_temp & ~0xfffffffffull;
    macho_setup_bootargs(info, as, bootargs_addr, virt_base, phys_base, VAtoPA(load_extra_offset),
dtb_address, dtb_size);

    // write bootloader
    uint32_t fixupcontext[FIXUP_MAX];
    fixupcontext[FIXUP_ARGPTR] = bootargs_addr;
    fixupcontext[FIXUP_ENTRYPOINT] = pc;
    write_bootloader("bootloader", info->loader_start,
                    bootloader_aarch64, fixupcontext, as);
    *pentry = info->loader_start;

out:
    if (data) {
        g_free(data);
    }
    if (rom_buf) {
        g_free(rom_buf);
    }
    return ret? high_addr_temp - low_addr_temp : -1;
}

```

This segment, in our case, was mapped below the address of the loaded mach-o header. This means that the `virt_base` does not point to the kernel mach-o header, and therefore doesn't work with the secure monitor code as presented above. One way I tried solving this was by setting the `virt_base` to the address of the mach-o header, but this made some kernel drivers code load below `virt_base`, which messed up a lot of stuff, like the following function:

```
vm_offset_t
ml_static_vtop(vm_offset_t va)
{
    for (size_t i = 0; (i < PTOV_TABLE_SIZE) && (ptov_table[i].len != 0); i++) {
        if ((va >= ptov_table[i].va) && (va < (ptov_table[i].va + ptov_table[i].len)))
            return (va - ptov_table[i].va + ptov_table[i].pa);
    }
    if (((vm_address_t)(va) - gVirtBase) >= gPhysSize)
        panic("ml_static_vtop(): illegal VA: %p\n", (void*)va);
    return ((vm_address_t)(va) - gVirtBase + gPhysBase);
}
```

Another approach I tried was to skip the execution of the secure monitor, and start straight from the kernel entry point in EL1. This worked until I hit the first SMC instruction. It might have been possible to solve this by patching the kernel at points where SMC is used, but I didn't want to go this way, as I opted for no patches at all (if possible), and you never know where not having some secure monitor functionality might hit you again. What eventually worked was setting the `virt_base` to a lower address below the lowest loaded segment, and just have another copy of the whole raw kernelcache file at this place (in addition to the copy loaded segment by segment where the code is actually executed from). This solution satisfied all the conditions of having the `virt_base` below all the virtual addresses actually used in the kernel, having it point to the kernel mach-o header, and having the kernel loaded at its preferred address segment by segment, where it is actually executed from.

Trust Cache

In this section, I will present the work that was done to load non-Apple, self-signed executables. iOS systems normally will only execute trusted executables that are either in a trust cache, or signed by Apple or an installed profile. More background on the subject can be found [here](#), as well as other writeups on the web and in books. In general, there are 3 types of trust caches:

1. A trust cache hardcoded in the kernelcache.
2. A trust cache that can be loaded at runtime from a file.
3. A trust cache in memory pointed to from the device tree.

I decided to go after the 3rd one. The following function contains the top level logic for checking whether an executable has a code signature that is approved for execution, based on trust cahces or otherwise:

```

...ff0061e4ef8 e1 ef 02 91 add x1,sp,#0xb0b
...ff0061e4efc e2 eb 02 91 add x2,sp,#0xb0b
...ff0061e4ef00 09 f8 ff 97 bl FUN_fffffff0061e2f24
...ff0061e4f00 68 02 40 b9 ldr w0,[x19]
...ff0061e4f00 08 01 00 2a orr w0,w0,w0
...ff0061e4f00 68 02 00 b9 str w0,[x19]
...ff0061e4f10 e8 ef 42 39 ldrb w0,[sp,#0xb0b]
...ff0061e4f14 01 00 71 cmp w0,w0
...ff0061e4f18 08 00 88 52 mov w0,#0x4000
...ff0061e4f1c 08 00 a7 72 movk w0,#0x4000, LSL #16
...ff0061e4f20 9c 03 88 1a csel w28,w28,w0,eq

    LAB_fffffff0061e4f24 XREF[1]: ffffff
...ff0061e4f24 e0 03 18 aa mov x0,x24
...ff0061e4f28 63 05 00 94 bl _csblob_get_cdhash
...ff0061e4f2c f7 03 00 aa mov x23,x0
...ff0061e4f30 77 03 00 b4 cbz x23,LAB_fffffff0061e4f9c
...ff0061e4f34 e0 03 17 aa mov x0,x23
...ff0061e4f38 05 01 ff 97 bl FUN_fffffff0061e134c
...ff0061e4f40 60 02 34 cbz w0,LAB_fffffff0061e5028
...ff0061e4f44 68 02 40 b9 ldr w0,[x19]
...ff0061e4f48 02 01 1c 2a orr w0,w0,w28
...ff0061e4f4c 62 02 00 b9 str w0,[x19]
...ff0061e4f4d c3 d5 ff b8 adrp x3,-0xfa363000
...ff0061e4f50 63 30 17 91 add x3=>s_in_kernel_fffffff005c9d5cc,x3,#0x5cc
...ff0061e4f54 e0 03 16 aa mov x0,x22
...ff0061e4f58 e1 03 18 aa mov x1,x24
...ff0061e4f5c e4 03 14 aa mov x4,x20
...ff0061e4f60 e5 03 15 aa mov x5,x21
...ff0061e4f64 0b 04 00 94 bl FUN_fffffff0061e5f90
...ff0061e4f68 17 00 00 d2 mov x23,#0x0
...ff0061e4f6c 40 02 00 36 tbz w0,#0x0,LAB_fffffff0061e4fb4
...ff0061e4f70 00 00 00 14 b LAB_fffffff0061e5170

    LAB_fffffff0061e4f74 XREF[1]: ffffff
...ff0061e4f74 00 00 00 94 bl FUN_fffffff0061e5f90
...ff0061e4f78 17 00 00 d2 mov x23,#0x0
...ff0061e4f7c 40 02 00 36 tbz w0,#0x0,LAB_fffffff0061e4fb4
...ff0061e4f70 00 00 00 14 b LAB_fffffff0061e5170

    LAB_fffffff0061e4f74 XREF[1]: ffffff
...ff0061e4f74 00 00 00 94 bl FUN_fffffff0061e5f90
...ff0061e4f78 17 00 00 d2 mov x23,#0x0
...ff0061e4f7c 40 02 00 36 tbz w0,#0x0,LAB_fffffff0061e4fb4
...ff0061e4f70 00 00 00 14 b LAB_fffffff0061e5170

    LAB_fffffff0061e4f74 XREF[1]: ffffff
...ff0061e4f74 00 00 00 94 bl FUN_fffffff0061e5f90
...ff0061e4f78 17 00 00 d2 mov x23,#0x0
...ff0061e4f7c 40 02 00 36 tbz w0,#0x0,LAB_fffffff0061e4fb4
...ff0061e4f70 00 00 00 14 b LAB_fffffff0061e5170

```

If we drill down deeper, we finally get to this function, that checks the static trust cache:

```

...ff0071b5c9c e0 07 9f 1a cset w0,ne
...ff0071b5ca0 04 00 00 14 b LAB_fffffff0071b5cb0

    LAB_fffffff0071b5ca4 XREF[2]: ffffff
...ff0071b5ca4 9f fe 9f c8 stlr xzr,[x20]=>DAT_fffffff00764bb0
...ff0071b5ca8 85 ic 00 94 bl FUN_fffffff0071bcebc
...ff0071b5cac e0 03 00 32 mov w0,#0x1

    LAB_fffffff0071b5cb0 XREF[1]: ffffff
...ff0071b5cb0 fd 7b 42 a9 ldp x20=>local_10,x30,[sp,#0x20]
...ff0071b5cb4 f4 4f 41 a9 ldp x20,x19,[sp,#local_20]
...ff0071b5cb8 f6 57 c3 a9 ldp x22,x21,[sp],#0x30
...ff0071b5cbc c0 03 5f d6 ret

    ***** FUNCTION *****
    ***** _cdecl __pmap_lookup_in_static_trust_cache(byt...
ulonglong __cdecl __pmap_lookup_in_static_trust_cache(byt...
    ulonglong <RETURN>
    byte * pbParm1_input_hash
    ulonglong uVar5_hash_table_part_size XREF[1]:
    byte * pbVar5_pointer_to_current_part_of_hash...
    ulonglong uVar5_index_hash_table XREF[1]:
    undefined8 Stack[-0x10]:8 local_10
    undefined8 Stack[-0x20]:8 local_20 XREF[2]:
    undefined8 Stack[-0x30]:8 local_30 XREF[3]:
    _pmap_lookup_in_static_trust_cache XREF[1]:
        Entry P _pmap_l _pmap_l ffffff
...ff0071b5cc0 ff c3 00 d1 sub sp,sp,#0x30
...ff0071b5cc4 f4 f0 01 a9 stp x20,x19,[sp,#local_20]

    LAB_fffffff00707de28 XREF[1]: ffffff
...ff0070efad8 28 7e b2 orr x9,x2r,#-0x4
...ff0070efab0 09 a5 00 f9 str x9,[x8,#0x148]
...ff0070efab0 00 da 9f 58 adr x0=>s_mac_policy_initmach_fffffff00702f5fe,-0x...
...ff0070efac0 1f 20 03 d5 nop
...ff0070efac4 35 08 00 94 bl FUN_fffffff0073eb98
...ff0070efac8 a8 28 00 98 adrp x8,-0xf8f89f0d000
...ff0070efacc 08 b9 43 99 ldr x8,[x8,#0x770]=>DAT_fffffff007603770
...ff0070efad0 48 00 00 b4 cbz x8,LAB_fffffff0070efad8
...ff0070efad4 00 01 3f d6 blr x8

    LAB_fffffff00707de18 XREF[1]: ffffff
...ff0070efad8 28 2c 00 b8 adrp x8,-0xff898c000
...ff0070efadc e9 03 00 32 mov w0,#0x1
...ff0070efae0 09 21 00 39 strb w0,[x8,#0x348]=>DAT_fffffff007674348
...ff0070efae4 88 e0 03 18 adr x0=>s_map_cs_no_legacy_tc_fffffff0070378d4,-0...
...ff0070efaec 61 19 c7 10 adr x1=>DAT_fffffff00707de18,-0xff8f821e8
...ff0070efaf0 1f 20 03 d5 nop
...ff0070efaf4 e2 03 32 mov w2,#0x4
...ff0070efaf8 03 00 02 52 mov w2,#0x0
...ff0070efafc ca 12 94 bl PE_parse_boot_argn
...ff0070efab4 73 02 c7 58 ldr x10,DAT_fffffff00707db70_static_trustcache_size
...ff0070efab8 03 00 04 cbz x10,LAB_fffffff0070efc38
...ff0070efb02 05 02 07 58 ldr x21,DAT_fffffff00707db68_segEXTRADATA_static_t...
...ff0070efb14 79 fc ff 07 str x21,[x25,#0xe20]=>DAT_fffffff00707de20
...ff0070efb1c e8 03 15 aa mov x8,x21
...ff0070efb20 7f 0e 00 f1 cmp x10,#0x3
...ff0070efb24 c8 00 00 b1 hli LAB_fffffff0070efb3c
...ff0070efb28 f3 03 00 f9 str x10,[sp]=>local_1b0
...ff0070efb2c e0 ed 03 38 adr x0=>_short_serialized_trust_cache_re_ffff...
...ff0070efb30 0f 20 03 d5 nop
...ff0070efb34 60 04 ff 97 ldi _panic
...ff0070efb34 60 04 ff 97 ldi _panic

    LAB_fffffff0070efb28 XREF[1]: ffffff
...ff0070efb28 20 00 00 00 ldi LAB_fffffff0070efb28
...ff0070efb30 20 00 00 00 ldi LAB_fffffff0070efb30
...ff0070efb34 20 00 00 00 ldi LAB_fffffff0070efb34
...ff0070efb38 20 00 00 00 ldi LAB_fffffff0070efb38
...ff0070efb42 20 00 00 00 ldi LAB_fffffff0070efb42
...ff0070efb46 20 00 00 00 ldi LAB_fffffff0070efb46
...ff0070efb50 20 00 00 00 ldi LAB_fffffff0070efb50
...ff0070efb54 20 00 00 00 ldi LAB_fffffff0070efb54
...ff0070efb58 20 00 00 00 ldi LAB_fffffff0070efb58
...ff0070efb62 20 00 00 00 ldi LAB_fffffff0070efb62
...ff0070efb66 20 00 00 00 ldi LAB_fffffff0070efb66
...ff0070efb70 20 00 00 00 ldi LAB_fffffff0070efb70
...ff0070efb74 20 00 00 00 ldi LAB_fffffff0070efb74
...ff0070efb78 20 00 00 00 ldi LAB_fffffff0070efb78
...ff0070efb82 20 00 00 00 ldi LAB_fffffff0070efb82
...ff0070efb86 20 00 00 00 ldi LAB_fffffff0070efb86
...ff0070efb90 20 00 00 00 ldi LAB_fffffff0070efb90
...ff0070efb94 20 00 00 00 ldi LAB_fffffff0070efb94
...ff0070efb98 20 00 00 00 ldi LAB_fffffff0070efb98
...ff0070efb9c 20 00 00 00 ldi LAB_fffffff0070efb9c
...ff0070efb9e 20 00 00 00 ldi LAB_fffffff0070efb9e
...ff0070efb9f 20 00 00 00 ldi LAB_fffffff0070efb9f
...ff0070efba0 20 00 00 00 ldi LAB_fffffff0070efba0
...ff0070efba4 20 00 00 00 ldi LAB_fffffff0070efba4
...ff0070efba8 20 00 00 00 ldi LAB_fffffff0070efba8
...ff0070efbae 20 00 00 00 ldi LAB_fffffff0070efbae
...ff0070efbb2 20 00 00 00 ldi LAB_fffffff0070efbb2
...ff0070efbb6 20 00 00 00 ldi LAB_fffffff0070efbb6
...ff0070efbb8 20 00 00 00 ldi LAB_fffffff0070efbb8
...ff0070efbbc 20 00 00 00 ldi LAB_fffffff0070efbbc
...ff0070efbc0 20 00 00 00 ldi LAB_fffffff0070efbc0
...ff0070efbc4 20 00 00 00 ldi LAB_fffffff0070efbc4
...ff0070efbc8 20 00 00 00 ldi LAB_fffffff0070efbc8
...ff0070efbcf 20 00 00 00 ldi LAB_fffffff0070efbcf
...ff0070efbd0 20 00 00 00 ldi LAB_fffffff0070efbd0
...ff0070efbd4 20 00 00 00 ldi LAB_fffffff0070efbd4
...ff0070efbd8 20 00 00 00 ldi LAB_fffffff0070efbd8
...ff0070efbdc 20 00 00 00 ldi LAB_fffffff0070efbdc
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0070efbea
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe0 20 00 00 00 ldi LAB_fffffff0070efbe0
...ff0070efbe4 20 00 00 00 ldi LAB_fffffff0070efbe4
...ff0070efbe8 20 00 00 00 ldi LAB_fffffff0070efbe8
...ff0070efbea 20 00 00 00 ldi LAB_fffffff0
```

The function above parses the raw trust cache format. It is left as an exercise to the reader to follow the code and the error messages, to conclude that the trust cache format is:

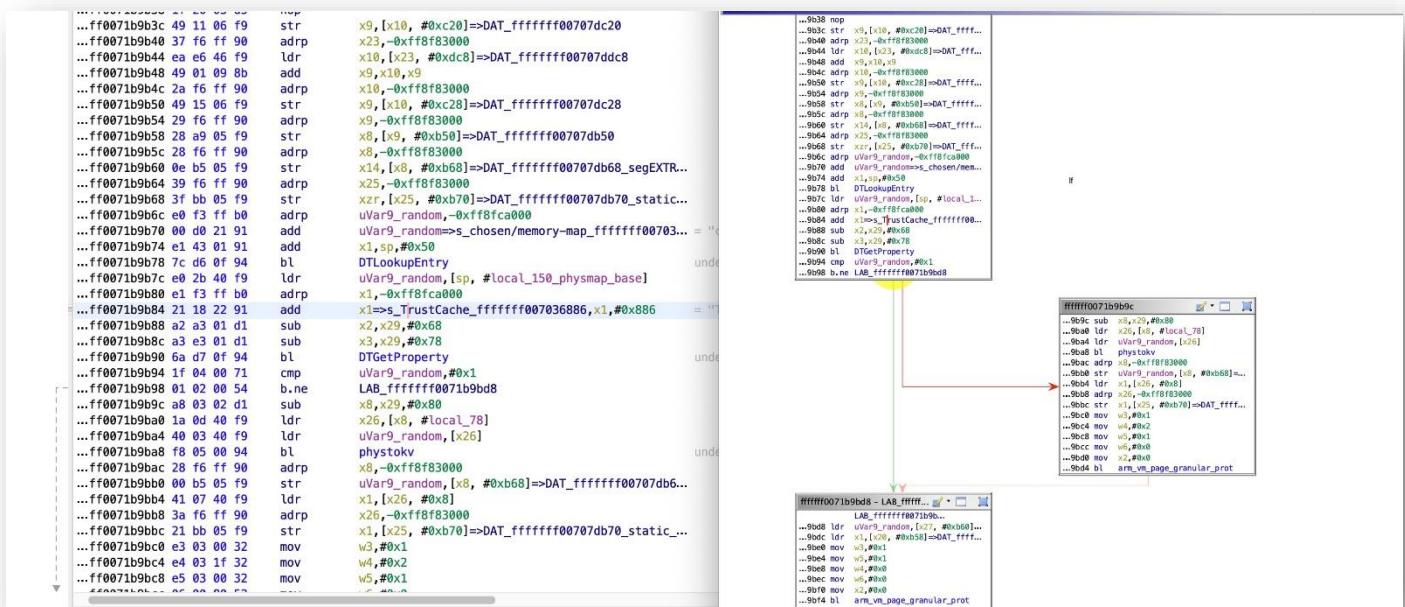
```
struct cdhash {
    uint8_t hash[20]; //first 20 bytes of the cdhash
    uint8_t hash_type; //left as 0
    uint8_t hash_flags; //left as 0
};

struct static_trust_cache_entry {
    uint64_t trust_cache_version; //should be 1
    uint64_t unknown1; //left as 0
    uint64_t unknown2; //left as 0
    uint64_t unknown3; //left as 0
    uint64_t unknown4; //left as 0
    uint64_t number_of_cdhashes;
    struct cdhash[];
};

struct static_trust_cache_buffer {
    uint64_t number_of_trust_caches_in_buffer;
    uint64_t offsets_to_trust_caches_from_beginning_of_buffer[];
    struct static_trust_cache_entry entries[];
};
```

And it seems that even though the structure supports multiple trust caches in the buffer, the code actually limits the size to 1.

Following XREFS from this function leads us to the following code:



This data is, therefore, read from the device tree.

Now, all that's left to do is load the trust cache into memory, and point to it from the device tree. I have to decide where to place it in memory. There is a kernel boot arg top of kernel data which points to the address after the kernel, ramdisk, device tree and the boot args. The first location I tried was near this top of kernel data address (before and after it). This didn't go so well because of the following code which is the code ~matching the above assembly:

```

void
arm_vm_prot_init(boot_args * args)
{
    segLOWESTTEXT = UINT64_MAX;
    if (segSizePRELINKTEXT && (segPRELINKTEXTB < segLOWESTTEXT)) segLOWESTTEXT = segPRELINKTEXTB;
    assert(segSizeTEXT);
    if (segTEXTB < segLOWESTTEXT) segLOWESTTEXT = segTEXTB;
    assert(segLOWESTTEXT < UINT64_MAX);

    segEXTRADATA = segLOWESTTEXT;
    segSizeEXTRADATA = 0;

    DTEentry memory_map;
    MemoryMapFileInfo *trustCacheRange;
    unsigned int trustCacheRangeSize;
    int err;

    err = DTLookupEntry(NULL, "chosen/memory-map", &memory_map);
    assert(err == kSuccess);

    err = DTGetProperty(memory_map, "TrustCache", (void**)&trustCacheRange, &trustCacheRangeSize);
    if (err == kSuccess) {
        assert(trustCacheRangeSize == sizeof(MemoryMapFileInfo));

        segEXTRADATA = phystokv(trustCacheRange->paddr);
        segSizeEXTRADATA = trustCacheRange->length;

        arm_vm_page_granular_RNX(segEXTRADATA, segSizeEXTRADATA, FALSE);
    }

    /* Map coalesced kext TEXT segment RWNX for now */
    arm_vm_page_granular_RWNX(segPRELINKTEXTB, segSizePRELINKTEXT, FALSE); // Refined in
OSKext::readPrelinkedExtensions

    /* Map coalesced kext DATA_CONST segment RWNX (could be empty) */
    arm_vm_page_granular_RWNX(segPLKDATACONSTB, segSizePLKDATACONST, FALSE); // Refined in
OSKext::readPrelinkedExtensions

    /* Map coalesced kext TEXT_EXEC segment RWX (could be empty) */
    arm_vm_page_granular_ROX(segPLKTEXTEXECB, segSizePLKTEXTEXEC, FALSE); // Refined in
OSKext::readPrelinkedExtensions

    /* if new segments not present, set space between PRELINK_TEXT and xnu TEXT to RWNX
     * otherwise we no longer expect any space between the coalesced kext read only segments and xnu
     * nosegments
     */
    if (!segSizePLKDATACONST && !segSizePLKTEXTEXEC) {
        if (segSizePRELINKTEXT)
            arm_vm_page_granular_RWNX(segPRELINKTEXTB + segSizePRELINKTEXT, segTEXTB -
(segPRELINKTEXTB + segSizePRELINKTEXT), FALSE);
    } else {
        /*
         * If we have the new segments, we should still protect the gap between kext
         * read-only pages and kernel read-only pages, in the event that this gap
         * exists.
         */
        if ((segPLKDATACONSTB + segSizePLKDATACONST) < segTEXTB) {
            arm_vm_page_granular_RWNX(segPLKDATACONSTB + segSizePLKDATACONST, segTEXTB -
(segPLKDATACONSTB + segSizePLKDATACONST), FALSE);
        }
    }

    /*
     * Protection on kernel text is loose here to allow shenanigans early on. These
     * protections are tightened in arm_vm_prot_finalize(). This is necessary because
     * we currently patch LowResetVectorBase in cpu.c.
     *
     * TEXT segment contains mach headers and other non-executable data. This will become RONX
     * later.
     */
    arm_vm_page_granular_RNX(segTEXTB, segSizeTEXT, FALSE);
}

```

```

/* Can DATACONST start out and stay RNX?
 * NO, stuff in this segment gets modified during startup (viz.
mac_policy_init()/mac_policy_list)
 * Make RNX in prot_finalize
 */
arm_vm_page_granular_RWNX(segDATACONSTB, segSizeDATACONST, FALSE);

/* TEXTEXEC contains read only executable code: becomes ROX in prot_finalize */
arm_vm_page_granular_RWX(segTEXTEXECB, segSizeTEXTEXEC, FALSE);

/* DATA segment will remain RWNX */
arm_vm_page_granular_RWNX(segDATAB, segSizeDATA, FALSE);

arm_vm_page_granular_RWNX(segBOOTDATAB, segSizeBOOTDATA, TRUE);
arm_vm_page_granular_RNX((vm_offset_t)&intstack_low_guard, PAGE_MAX_SIZE, TRUE);
arm_vm_page_granular_RNX((vm_offset_t)&intstack_high_guard, PAGE_MAX_SIZE, TRUE);
arm_vm_page_granular_RNX((vm_offset_t)&excepstack_high_guard, PAGE_MAX_SIZE, TRUE);

arm_vm_page_granular_ROX(segKLDB, segSizeKLD, FALSE);
arm_vm_page_granular_RWNX(segLINKB, segSizeLINK, FALSE);
arm_vm_page_granular_RWNX(segPLKLINKEDITB, segSizePLKLINKEDIT, FALSE); // Coalesced kext
LINKEDIT segment
arm_vm_page_granular_ROX(segLASTB, segSizeLAST, FALSE); // __LAST may be empty, but we cannot
assume this

arm_vm_page_granular_RWNX(segPRELINKDATAB, segSizePRELINKDATA, FALSE); // Prelink __DATA for
kexts (RW data)

if (segSizePLKLLVMCOV > 0)
    arm_vm_page_granular_RWNX(segPLKLLVMCOVB, segSizePLKLLVMCOV, FALSE); // LLVM code
coverage data

arm_vm_page_granular_RWNX(segPRELINKINFOB, segSizePRELINKINFO, FALSE); /* PreLinkInfoDictionary
*/
arm_vm_page_granular_RNX(phystokv(args->topOfKernelData), BOOTSTRAP_TABLE_SIZE, FALSE); // Boot
page tables; they should not be mutable.
}

```

Here we can see that when we have a static trust cache, `segEXTRADATA` is set to the trust cache buffer, instead of `segLOWESTTEXT`.

In the following 2 functions we can see that if the data between

`gVirtBase` and `segEXTRADATA` holds anything meaningful, terrible things happen:

```

static void
arm_vm_physmap_init(boot_args *args, vm_map_address_t physmap_base, vm_map_address_t dynamic_memory_begin
__unused)
{
    ptov_table_entry temp_ptov_table[PTOV_TABLE_SIZE];
    bzero(temp_ptov_table, sizeof(temp_ptov_table));

    // Will be handed back to VM layer through ml_static_mfree() in arm_vm_prot_finalize()
    arm_vm_physmap_slide(temp_ptov_table, physmap_base, gVirtBase, segEXTRADATA - gVirtBase,
AP_RWNA, FALSE);

    arm_vm_page_granular_RWNX(end_kern, phystokv(args->topOfKernelData) - end_kern, FALSE); /*

Device Tree, RAM Disk (if present), bootArgs */

    arm_vm_physmap_slide(temp_ptov_table, physmap_base, (args->topOfKernelData +
BOOTSTRAP_TABLE_SIZE - gPhysBase + gVirtBase),
                           real_avail_end - (args->topOfKernelData + BOOTSTRAP_TABLE_SIZE),
AP_RWNA, FALSE); // rest of physmem

    assert((temp_ptov_table[ptov_index - 1].va + temp_ptov_table[ptov_index - 1].len) <=
dynamic_memory_begin);

    // Sort in descending order of segment length. LUT traversal is linear, so largest (most likely
used)
    // segments should be placed earliest in the table to optimize lookup performance.
}

```

```

        qsort(temp_ptov_table, PTOV_TABLE_SIZE, sizeof(temp_ptov_table[0]), cmp_ptov_entries);

        memcpy(ptov_table, temp_ptov_table, sizeof(ptov_table));
    }

void
arm_vm_prot_finalize(boot_args * args __unused)
{
    /*
     * At this point, we are far enough along in the boot process that it will be
     * safe to free up all of the memory preceding the kernel. It may in fact
     * be safe to do this earlier.
     *
     * This keeps the memory in the V-to-P mapping, but advertises it to the VM
     * as usable.
     */

    /*
     * if old style PRELINK segment exists, free memory before it, and after it before XNU text
     * otherwise we're dealing with a new style kernel cache, so we should just free the
     * memory before PRELINK_TEXT segment, since the rest of the KEXT read only data segments
     * should be immediately followed by XNU's TEXT segment
     */
}

ml_static_mfree(phystokv(gPhysBase), segEXTRADATA - gVirtBase);

/*
 * KTRR support means we will be mucking with these pages and trying to
 * protect them; we cannot free the pages to the VM if we do this.
 */
if (!segSizePLKDATACONST && !segSizePLKTEXTEXEC && segSizePRELINKTEXT) {
    /* If new segments not present, PRELINK_TEXT is not dynamically sized, free DRAM
between it and xnu TEXT */
    ml_static_mfree(segPRELINKTEXTB + segSizePRELINKTEXT, segTEXTB - (segPRELINKTEXTB +
segSizePRELINKTEXT));
}

/*
 * LowResetVectorBase patching should be done by now, so tighten executable
 * protections.
 */
arm_vm_page_granular_ROX(segTEXTEXECB, segSizeTEXTEXEC, FALSE);

/* tighten permissions on kext read only data and code */
if (segSizePLKDATACONST && segSizePLKTEXTEXEC) {
    arm_vm_page_granular_RNX(segPRELINKTEXTB, segSizePRELINKTEXT, FALSE);
    arm_vm_page_granular_ROX(segPLKTEXTEXECB, segSizePLKTEXTEXEC, FALSE);
    arm_vm_page_granular_RNX(segPLKDATACONSTB, segSizePLKDATACONST, FALSE);
}

cpu_stack_alloc(&BootCpuData);
arm64_replace_bootstack(&BootCpuData);
ml_static_mfree(phystokv(segBOOTDATAB - gVirtBase + gPhysBase), segSizeBOOTDATA);

#ifndef __ARM_KERNEL_PROTECT__
    arm_vm_populate_kernel_el0_mappings();
#endif /* __ARM_KERNEL_PROTECT__ */

#ifndef defined(KERNEL_INTEGRITY_KTRR)
    /*
     * __LAST,__pinst should no longer be executable.
     */
    arm_vm_page_granular_RNX(segLASTB, segSizeLAST, FALSE);

    /*
     * Must wait until all other region permissions are set before locking down DATA_CONST
     * as the kernel static page tables live in DATA_CONST on KTRR enabled systems
     * and will become immutable.
     */
#endif

```

```

    arm_vm_page_granular_RNX(segDATACONSTB, segSizeDATACONST, FALSE);

#ifndef __ARM_L1_PTW__
    FlushPoC_Dcache();
#endif
    __builtin_arm_dsb(DSB_ISH);
    flush_mmu_tlb();
}

```

Alright, so now, based on the above observation, I decided to place the trust cache buffer right after the raw kernel file I placed at `virt_base`. This, of course, still didn't work. Following the code that sets the page table, I found where this memory location gets unloaded from the table, and finally understood that a few pages after the end of the raw kernel file get unloaded from memory at some point. Therefore, I placed it a few MBs above that address, and it finally worked (partially).

Looking at the code at:

```

...ff0071b5c9c e0 07 9f 1a    cset    w0,ne
...ff0071b5ca0 04 00 00 14    b       LAB_fffffff0071b5cb0

LAB_fffffff0071b5ca4
...ff0071b5ca4 9f fe 9f c8    stlr    [x20]>=DAT_fffffff00764bb80
...ff0071b5ca8 85 1c 00 94    bl      FUN_fffffff0071bcbe
...ff0071b5cac e0 03 00 32    mov     w0,#0x1

LAB_fffffff0071b5cb0
...ff0071b5cb0 fd 7b 42 a9    ldp    x20<local_10,x30,[sp, #0x20]
...ff0071b5cb4 f4 4f 41 a9    ldp    x20,x19,[sp, #local_20]
...ff0071b5cb8 f6 57 c3 a8    ldp    x22,x21,[sp], #0x30
...ff0071b5cbc c0 03 5f d6    ret

***** FUNCTION *****
ulonglong __cdecl ulonglong _pmap_lookup_in_static_trust_cache(byt...
byte * x0:8 <RETURN>
byte * x0:8 pbParm1_input_hash
byte * x8:8 uVar5_hash_table_part_size
byte * x9:8 pbVar5_pointer_to_current_part_of_hash...
ulonglong x11:8 uVar5_index_hash_table
undefined8 Stack[-0x10]:8 local_10
undefined8 Stack[-0x20]:8 local_20
undefined8 Stack[-0x30]:8 local_30
_pmap_lookup_in_static_trust_cache
XREF[3]: Entry P _pmap_l _pmap_l ffffff

...ff0071b5cc0 ff c3 00 d1    sub    sp,sp,#0x30
...ff0071b5cc4 f4 4f 01 a9    stp    x20,x19,[sp, #local_20]

2 ulonglong _pmap_lookup_in_static_trust_cache(byte *pbParm1_input_hash)
3
4
5 longlong lVar1;
6 byte *pVar2;
7 byte bVar3;
8 ushort uVar4;
9 ulonglong uVar5_hash_table_part_size;
10 byte *pbVar5_pointer_to_current_part_of_hash_table;
11 ulonglong uVar5;
12 ulonglong uVar6;
13 uint uVar7;
14 uint uVar8;
15
16
17 if (DAT_fffffff00707de28 != 0) {
18
19     if ((uint *)DAT_fffffff00707de28 + 0x14) != 0) {
20         pbVar5_pointer_to_current_part_of_hash_table = (byte *)DAT_fffffff00707de28 + 0x18;
21     do {
22         uVar5_index_hash_table = uVar5_hash_table_part_size >> 1;
23         uVar7 = (uint)pbParm1_input_hash;
24         bVar3 = pbVar5_pointer_to_current_part_of_hash_table[uVar5_index_hash_table * 0x16];
25         uVar8 = (uint)bVar3;
26         if (pbParm1_input_hash == bVar3) {
27             uVar1 = (uint)pbParm1_input_hash[1];
28             uVar8 = (uint)pbVar5_pointer_to_current_part_of_hash_table
29                         [uVar5_index_hash_table * 0x16 + 1];
30             if (pbParm1_input_hash[1] ==
31                 pbVar5_pointer_to_current_part_of_hash_table[uVar5_index_hash_table * 0x16 + 1]) {
32                 uVar7 = (uint)pbParm1_input_hash[2];
33                 uVar8 = (uint)pbVar5_pointer_to_current_part_of_hash_table
34                         [uVar5_index_hash_table * 0x16 + 2];
35             if (pbParm1_input_hash[2] ==
36                 pbVar5_pointer_to_current_part_of_hash_table[uVar5_index_hash_table * 0x16 + 2]) {
37                 lVar1 = (ulonglong)pbParm1_input_hash[3];
38             }
39         }
40     }
41 }
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
59
60
61
62
63
64
65
66
67
68
69
69
70
71
72
73
74
75
76
77
78
79
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
109
110
111
112
113
114
115
116
117
118
119
119
120
121
122
123
124
125
126
127
128
129
129
130
131
132
133
134
135
136
137
138
139
139
140
141
142
143
144
145
146
147
148
149
149
150
151
152
153
154
155
156
157
158
159
159
160
161
162
163
164
165
166
167
167
168
169
169
170
171
172
173
174
175
176
177
177
178
179
179
180
181
182
183
184
185
186
186
187
188
188
189
189
190
191
192
193
194
195
196
196
197
198
198
199
199
200
201
202
203
204
205
205
206
207
207
208
208
209
209
210
211
212
213
213
214
214
215
215
216
216
217
217
218
218
219
219
220
220
221
221
222
222
223
223
224
224
225
225
226
226
227
227
228
228
229
229
230
230
231
231
232
232
233
233
234
234
235
235
236
236
237
237
238
238
239
239
240
240
241
241
242
242
243
243
244
244
245
245
246
246
247
247
248
248
249
249
250
250
251
251
252
252
253
253
254
254
255
255
256
256
257
257
258
258
259
259
260
260
261
261
262
262
263
263
264
264
265
265
266
266
267
267
268
268
269
269
270
270
271
271
272
272
273
273
274
274
275
275
276
276
277
277
278
278
279
279
280
280
281
281
282
282
283
283
284
284
285
285
286
286
287
287
288
288
289
289
290
290
291
291
292
292
293
293
294
294
295
295
296
296
297
297
298
298
299
299
300
300
301
301
302
302
303
303
304
304
305
305
306
306
307
307
308
308
309
309
310
310
311
311
312
312
313
313
314
314
315
315
316
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1
```

```

        <string>/iosbinpack64/bin/bash</string>
    </array>
    <key>RunAtLoad</key>
    <true/>
    <key>StandardErrorPath</key>
    <string>/dev/console</string>
    <key>StandardInPath</key>
    <string>/dev/console</string>
    <key>StandardOutPath</key>
    <string>/dev/console</string>
    <key>Umask</key>
    <integer>0</integer>
    <key>UserName</key>
    <string>root</string>
</dict>
</plist>
```

This made launchd try and execute bash, but still no go. It seems that the ramdisk comes without the dynamic loader cache on it. To solve this, I copied the dyld cache from the full disk image to the ramdisk (after resizing the ramdisk so it would have enough space for it). Alas, it still didn't work and it seemed the problem was still the same missing libraries, even with the dyld cache in place. To debug this, I needed to better understand where the failure happens. I found out that loading the cache happens inside [dyld](#) in:

```

static void mapSharedCache()
{
    uint64_t cacheBaseAddress = 0;
    // quick check if a cache is already mapped into shared region
    if ( _shared_region_check_np(&cacheBaseAddress) == 0 ) {
        sSharedCache = (dyld_cache_header*)cacheBaseAddress;
        // if we don't understand the currently mapped shared cache, then ignore
#ifndef __x86_64__
        const char* magic = (sHaswell ? ARCH_CACHE_MAGIC_H : ARCH_CACHE_MAGIC);
#else
        const char* magic = ARCH_CACHE_MAGIC;
#endif
        if ( strcmp(sSharedCache->magic, magic) != 0 ) {
            sSharedCache = NULL;
            if ( gLinkContext.verboseMapping ) {
                dyld::log("dyld: existing shared cached in memory is not
compatible\n");
                return;
            }
        }
        // check if cache file is slidable
        const dyld_cache_header* header = sSharedCache;
        if ( (header->mappingOffset >= 0x48) && (header->slideInfoSize != 0) ) {
            // solve for slide by comparing loaded address to address of first region
            const uint8_t* loadedAddress = (uint8_t*)(long)(mappings[0].address);
            const dyld_cache_mapping_info* const mappings =
(dyld_cache_mapping_info*)(loadedAddress+header->mappingOffset);
            const uint8_t* preferredLoadAddress = (uint8_t*)(long)(mappings[0].address);
            sSharedCacheSlide = loadedAddress - preferredLoadAddress;
            dyld::gProcessInfo->sharedCacheSlide = sSharedCacheSlide;
            //dyld::log("sSharedCacheSlide=0x%08lX, loadedAddress=%p,
preferredLoadAddress=%p\n", sSharedCacheSlide, loadedAddress, preferredLoadAddress);
        }
        // if cache has a uuid, copy it
        if ( header->mappingOffset >= 0x68 ) {
            memcpy(dyld::gProcessInfo->sharedCacheUUID, header->uuid, 16);
        }
        // verbose logging
        if ( gLinkContext.verboseMapping ) {
            dyld::log("dyld: re-using existing shared cache mapping\n");
        }
    }
    else {
#ifndef __i386__ || __x86_64__

```

```

// <rdar://problem/5925940> Safe Boot should disable dyld shared cache
// if we are in safe-boot mode and the cache was not made during this boot cycle,
// delete the cache file
uint32_t safeBootValue = 0;
size_t safeBootValueSize = sizeof(safeBootValue);
if ( (sysctlbyname("kern.safeboot", &safeBootValue, &safeBootValueSize, NULL, 0) == 0)
&& (safeBootValue != 0) ) {
    // user booted machine in safe-boot mode
    struct stat dyldCacheStatInfo;
    // Don't use custom DYLD_SHARED_CACHE_DIR if provided, use standard path
    if ( my_stat(MACOSX_DYLD_SHARED_CACHE_DIR DYLD_SHARED_CACHE_BASE_NAME
ARCH_NAME, &dyldCacheStatInfo) == 0 ) {
        struct timeval bootTimeValue;
        size_t bootTimeValueSize = sizeof(bootTimeValue);
        if ( (sysctlbyname("kern.boottime", &bootTimeValue,
&bootTimeValueSize, NULL, 0) == 0) && (bootTimeValue.tv_sec != 0) ) {
            // if the cache file was created before this boot, then
            // throw it away and let it rebuild itself
            if ( dyldCacheStatInfo.st_mtime < bootTimeValue.tv_sec ) {
                ::unlink(MACOSX_DYLD_SHARED_CACHE_DIR
DYLD_SHARED_CACHE_BASE_NAME ARCH_NAME);
                gLinkContext.sharedRegionMode =
                    return;
            }
        }
    }
}
#endif

// map in shared cache to shared region
int fd = openSharedCacheFile();
if ( fd != -1 ) {
    uint8_t firstPages[8192];
    if ( ::read(fd, firstPages, 8192) == 8192 ) {
        dyld_cache_header* header = (dyld_cache_header*)firstPages;
#ifdef __x86_64__
        const char* magic = (sHaswell ? ARCH_CACHE_MAGIC_H :
ARCH_CACHE_MAGIC);
#else
        const char* magic = ARCH_CACHE_MAGIC;
#endif
        if ( strcmp(header->magic, magic) == 0 ) {
            const dyld_cache_mapping_info* const fileMappingsStart =
(dyld_cache_mapping_info*)&firstPages[header->mappingOffset];
            const dyld_cache_mapping_info* const fileMappingsEnd =
shared_file_mapping_np mappings[header-
>mappingCount] = header->mappingCount;
            int codeSignatureMappingIndex = -1;
            int readWriteMappingIndex = -1;
            int readOnlyMappingIndex = -1;
            // validate that the cache file has not been truncated
            bool goodCache = false;
            struct stat stat_buf;
            if ( fstat(fd, &stat_buf) == 0 ) {
                goodCache = true;
                int i=0;
                for (const dyld_cache_mapping_info* p =
fileMappingsStart; p < fileMappingsEnd; ++p, ++i) {
                    mappings[i].sfm_address =
p->address;
                    mappings[i].sfm_size =
p->size;
                    p->fileOffset;
                    mappings[i].sfm_file_offset = p-
>fileOffset;
                    mappings[i].sfm_max_prot = p->maxProt;
                    mappings[i].sfm_init_prot = p-
>initProt;
                    update_dyld_shared_cache tool could make a cache file
otherwise ok.
                    // rdar://problem/5694507 old
                    // that is not page aligned, but
                }
            }
        }
    }
}

```

```

if ( p->fileOffset+p->size >
    dyld::log("dyld: shared cached
file is corrupt: %s" DYLD_SHARED_CACHE_BASE_NAME ARCH_NAME "\n", sSharedCacheDir);
    goodCache = false;
}
if ( (mappings[i].sfm_init_prot &
    (VM_PROT_READ|VM_PROT_WRITE)) == (VM_PROT_READ|VM_PROT_WRITE) ) {
    readWriteMappingIndex = i;
}
if ( mappings[i].sfm_init_prot ==
    VM_PROT_READ ) {
    readOnlyMappingIndex = i;
}
// if shared cache is code signed, add a mapping
uint64_t signatureSize = header-
// zero size in header means signature runs to
if ( signatureSize == 0 )
    signatureSize = stat_buf.st_size -
header->codeSignatureOffset;
if ( signatureSize != 0 ) {
int linkeditMapping = mappingCount-1;
    codeSignatureMappingIndex =
mappingCount++;

mappings[codeSignatureMappingIndex].sfm_address =
mappings[linkeditMapping].sfm_address + mappings[linkeditMapping].sfm_size;
#if __arm__ || __arm64__
    mappings[codeSignatureMappingIndex].sfm_size
        = (signatureSize+16383) & (-16384);
#else
    mappings[codeSignatureMappingIndex].sfm_size
        = (signatureSize+4095) & (-4096);
#endif

mappings[codeSignatureMappingIndex].sfm_file_offset
    = header->codeSignatureOffset;
mappings[codeSignatureMappingIndex].sfm_max_prot
    = VM_PROT_READ;
mappings[codeSignatureMappingIndex].sfm_init_prot
    = VM_PROT_READ;
}

#if __MAC_OS_X_VERSION_MIN_REQUIRED
// sanity check that /usr/lib/libSystem.B.dylib stat() info
matches cache
if ( header->imagesCount * sizeof(dyld_cache_image_info) +
header->imagesOffset < 8192 ) {
    bool foundLibSystem = false;
    if ( my_stat("/usr/lib/libSystem.B.dylib",
&stat_buf) == 0 ) {
(dyld_cache_image_info*)&firstPages[header->imagesOffset];
imagesEnd = &images[header->imagesCount];
images; p < imagesEnd; ++p) {
stat_buf.st_mtime) && ((ino_t)p->inode == stat_buf.st_ino) ) {
const dyld_cache_image_info* images =
const dyld_cache_image_info* const
for (const dyld_cache_image_info* p =
if ( ((time_t)p->modTime ==
foundLibSystem = true;
break;
}
}

if ( !sSharedCacheIgnoreInodeAndTimeStamp &&
!foundLibSystem ) {
dyld::log("dyld: shared cached file was
built against a different libSystem.dylib, ignoring cache.\n"

```



```

cacheSlide;
sSharedCacheSlide=0x%08lX\n", sSharedCache, sSharedCacheSlide);
dyld::gProcessInfo->sharedCacheSlide =
//dyld::log("sSharedCache=%p
// if cache has a uuid, copy it
if ( header->mappingOffset >= 0x68 ) {
    memcpy(dyld::gProcessInfo-
>sharedCacheUUID, header->uuid, 16);
}

}
else {
    throw "dyld shared cache could not be
mapped";
#endif

file could not be mapped\n");
}

}
else {
    if ( gLinkContext.verboseMapping )
        dyld::log("dyld: shared cached file is
invalid\n");
}

}
else {
    if ( gLinkContext.verboseMapping )
        dyld::log("dyld: shared cached file cannot be read\n");
    close(fd);
}

}
else {
    if ( gLinkContext.verboseMapping )
        dyld::log("dyld: shared cached file cannot be opened\n");
}

}

// remember if dyld loaded at same address as when cache built
if ( sSharedCache != NULL ) {
    gLinkContext.dyldLoadedAtSameAddressNeededBySharedCache = ((uintptr_t)(sSharedCache-
>dyldBaseAddress) == (uintptr_t)&_mh_dylinker_header);
}

// tell gdb where the shared cache is
if ( sSharedCache != NULL ) {
    const dyld_cache_mapping_info* const start =
(dyld_cache_mapping_info*)((uint8_t*)sSharedCache + sSharedCache->mappingOffset);
    dyld_shared_cache_ranges.sharedRegionsCount = sSharedCache->mappingCount;
    // only room to tell gdb about first four regions
    if ( dyld_shared_cache_ranges.sharedRegionsCount > 4 )
        dyld_shared_cache_ranges.sharedRegionsCount = 4;
    const dyld_cache_mapping_info* const end =
&start[dyld_shared_cache_ranges.sharedRegionsCount];
    int index = 0;
    for (const dyld_cache_mapping_info* p = start; p < end; ++p, ++index ) {
        dyld_shared_cache_ranges.ranges[index].start = p->address+sSharedCacheSlide;
        dyld_shared_cache_ranges.ranges[index].length = p->size;
        if ( gLinkContext.verboseMapping ) {
            dyld::log("      0x%08lX->0x%08lX %s%s%s init=%x, max=%x\n",
p->address+sSharedCacheSlide, p-
>address+sSharedCacheSlide+p->size-1,
                ((p->initProt & VM_PROT_READ) ? "read " : ""),
                ((p->initProt & VM_PROT_WRITE) ? "write " : ""),
                ((p->initProt & VM_PROT_EXECUTE) ? "execute " : ""),
                p-
>initProt, p->maxProt);
        }
    }
}

// If a non-writable and executable region is found in the R/W shared region,
// then this is __IMPORT segments
// This is an old cache. Make writable. dyld no longer supports turn W on
and off as it binds

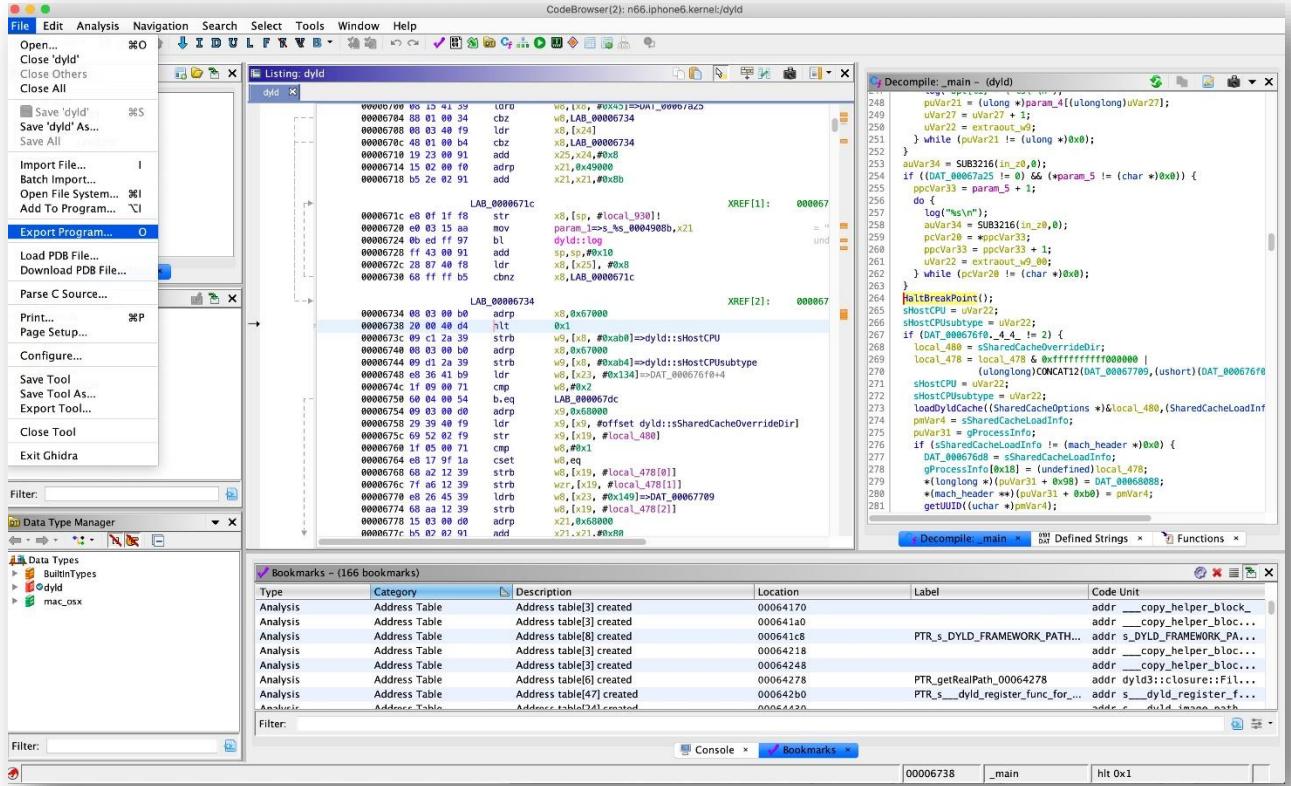
```

```

        if ( (p->initProt == (VM_PROT_READ|VM_PROT_EXECUTE)) && ((p->address &
0xF0000000) == 0xA0000000) ) {
            if ( p->size != 0 ) {
                vm_prot_t prot = VM_PROT_EXECUTE | PROT_READ |
VM_PROT_WRITE;
                vm_protect(mach_task_self(), p->address, p->size, false,
prot);
                if ( gLinkContext.verboseMapping ) {
                    dyld::log("%18s at 0x%08llx->0x%08llx altered
permissions to %c%c%c\n", "", p->address,
p->address+p->size-1,
(prot & PROT_READ) ? 'r' : '.', (prot &
PROT_WRITE) ? 'w' : '.', (prot & PROT_EXEC) ? 'x' : '.');
                }
            }
        }
#endif
}
if ( gLinkContext.verboseMapping ) {
    // list the code blob
    dyld_cache_header* header = (dyld_cache_header*)sSharedCache;
    uint64_t signatureSize = header->codeSignatureSize;
    // zero size in header means signature runs to end-of-file
    if ( signatureSize == 0 ) {
        struct stat stat_buf;
        // FIXME: need size of cache file actually used
        if ( my_stat(IPHONE_DYLD_SHARED_CACHE_DIR
DYLD_SHARED_CACHE_BASE_NAME ARCH_NAME, &stat_buf) == 0 )
            signatureSize = stat_buf.st_size - header-
>codeSignatureOffset;
    }
    if ( signatureSize != 0 ) {
        const dyld_cache_mapping_info* const last =
&start[dyld_shared_cache_ranges.sharedRegionsCount-1];
        uint64_t codeBlobStart = last->address + last->size;
        dyld::log("      0x%08llx->0x%08llx (code signature)\n",
codeBlobStart, codeBlobStart+signatureSize);
    }
}
#endif
// check for file that enables dyld shared cache dylibs to be overridden
struct stat enableStatBuf;
// check file size to determine if correct file is in place.
// See <rdar://problem/13591370> Need a way to disable roots without removing
/S/L/C/com.apple.dyld/enable...
    sDylibsOverrideCache = ( (my_stat(IPHONE_DYLD_SHARED_CACHE_DIR "enable-dylibs-to-
override-cache", &enableStatBuf) == 0)
&&
(enableStatBuf.st_size < ENABLE_DYLIBS_TO_OVERRIDE_CACHE_SIZE) );
#endif
}
}

```

By using the fun feature from the [previous post](#), which lets us debug user mode apps in the gdb kernel debugger, I was able to debug, by stepping through this function, and seeing what fails. To do this, I patched dyld with an `HLT` instruction, which our modified QEMU treats as a breakpoint. I then re-signed the executable with `jtool`, and added the new signature to the static trust cache:

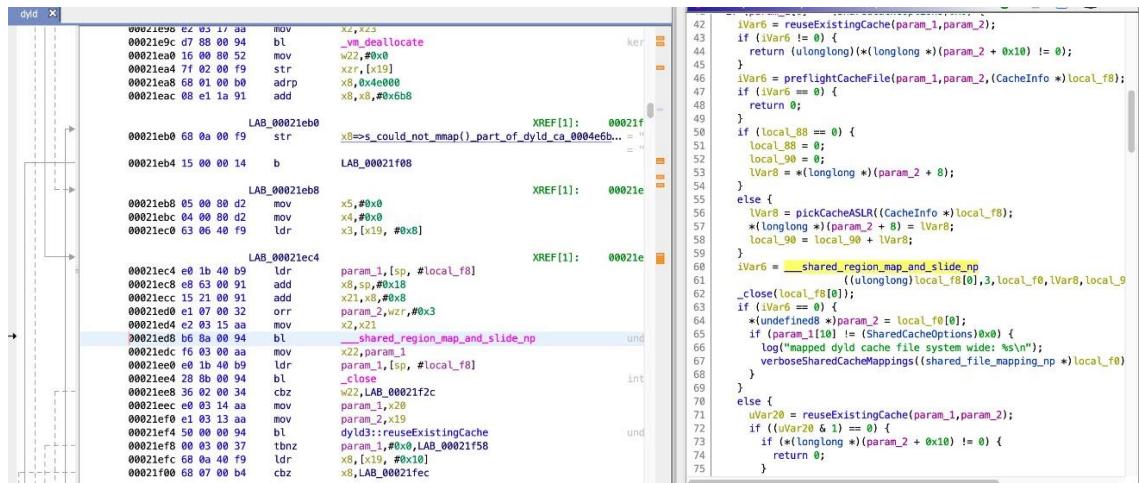


This showed me that actually the failure is in:

```

if (_shared_region_map_and_slide_np(fd, mappingCount, mappings, codeSignatureMappingIndex,
cacheSlide, slideInfo, slideInfoSize) == 0) {
    // successfully mapped cache into shared region
    sSharedCache = (dyld_cache_header*)mappings[0].sfm_address;
    sSharedCacheSlide = cacheSlide;
    dyld::gProcessInfo->sharedCacheSlide = cacheSlide;
    //dyld::log("sSharedCache=%p sSharedCacheSlide=0x%08lx\n", sSharedCache, sSharedCacheSlide);
    // if cache has a uuid, copy it
    if ( header->mappingOffset >= 0x68 ) {
        memcpy(dyld::gProcessInfo->sharedCacheUUID, header->uuid, 16);
    }
}
else {
    if __IPHONE_OS_VERSION_MIN_REQUIRED
        throw "dyld shared cache could not be mapped";
#endif
    if ( gLinkContext.verboseMapping )
        dyld::log("dyld: shared cached file could not be mapped\n");
}

```



Good thing I am running a kernel debugger, so I can step right into the kernel system call and see where it fails. And another good thing is that I have some version of the [code](#):

```
int
shared_region_map_and_slide_np(
    struct proc *p,
    struct shared_region_map_and_slide_np_args *uap,
    __unused int *retvalp)
{
    struct shared_file_mapping_np *mappings;
    unsigned int mappings_count = uap->count;
    kern_return_t kr = KERN_SUCCESS;
    uint32_t slide = uap->slide;

#define SFM_MAX_STACK 8
    struct shared_file_mapping_np stack_mappings[SFM_MAX_STACK];

    /* Is the process chrooted?? */
    if (p->p_fd->fd_rdir != NULL) {
        kr = EINVAL;
        goto done;
    }

    if ((kr = vm_shared_region_sliding_valid(slide)) != KERN_SUCCESS) {
        if (kr == KERN_INVALID_ARGUMENT) {
            /*
             * This will happen if we request sliding again
             * with the same slide value that was used earlier
             * for the very first sliding.
             */
            kr = KERN_SUCCESS;
        }
        goto done;
    }

    if (mappings_count == 0) {
        SHARED_REGION_TRACE_INFO(
            ("shared_region: %p [%d(%s)] map(): "
             "no mappings\n",
             (void *)VM_KERNEL_ADDRPERM(current_thread()),
             p->p_pid, p->p_comm));
        kr = 0; /* no mappings: we're done ! */
        goto done;
    } else if (mappings_count <= SFM_MAX_STACK) {
        mappings = &stack_mappings[0];
    } else {
        SHARED_REGION_TRACE_ERROR(
            ("shared_region: %p [%d(%s)] map(): "
             "too many mappings (%d)\n",
             (void *)VM_KERNEL_ADDRPERM(current_thread()),
             p->p_pid, p->p_comm,
             mappings_count));
        kr = KERN_FAILURE;
        goto done;
    }

    if ( (kr = shared_region_copyin_mappings(p, uap->mappings, uap->count, mappings))) {
        goto done;
    }

    kr = _shared_region_map_and_slide(p, uap->fd, mappings_count, mappings,
                                      slide,
                                      uap->slide_start, uap->slide_size);
    if (kr != KERN_SUCCESS) {
        return kr;
    }

done:
    return kr;
}
```

By stepping through the code in the debugger, I found that the call to `_shared_region_map_and_slide()` is what actually fails:

```
/*
 * shared_region_map_np()
 *
 * This system call is intended for dyld.
 *
 * dyld uses this to map a shared cache file into a shared region.
 * This is usually done only the first time a shared cache is needed.
 * Subsequent processes will just use the populated shared region without
 * requiring any further setup.
 */
int
_shared_region_map_and_slide(
    struct proc                                *p,
    int                                         fd,
    uint32_t                                     mappings_count,
    struct shared_file_mapping_np               *mappings,
    uint32_t                                     slide,
    user_addr_t                                  slide_start,
    user_addr_t                                  slide_size)
{
    int                                         error;
    kern_return_t                               kr;
    struct fileproc                            *fp;
    struct vnode                                *vp, *root_vp, *scdir_vp;
    struct vnode_attr                           va;
    off_t                                       fs;
    memory_object_size_t                      file_size;
#if CONFIG_MACF
    vm_prot_t                                  maxprot = VM_PROT_ALL;
#endif
    memory_object_control_t                    file_control;
    struct vm_shared_region                   *shared_region;
    uint32_t                                    i;

    SHARED_REGION_TRACE_DEBUG(
        ("shared_region: %p [%d(%s)] -> map\n",
         (void *)VM_KERNEL_ADDRPERM(current_thread()),
         p->p_pid, p->p_comm));

    shared_region = NULL;
    fp = NULL;
    vp = NULL;
    scdir_vp = NULL;

    /* get file structure from file descriptor */
    error = fp_lookup(p, fd, &fp, 0);
    if (error) {
        SHARED_REGION_TRACE_ERROR(
            ("shared_region: %p [%d(%s)] map: "
             "fd=%d lookup failed (error=%d)\n",
             (void *)VM_KERNEL_ADDRPERM(current_thread()),
             p->p_pid, p->p_comm, fd, error));
        goto done;
    }

    /* make sure we're attempting to map a vnode */
    if (FILEGLOB_DTYPE(fp->f_fglob) != DTYPE_VNODE) {
        SHARED_REGION_TRACE_ERROR(
            ("shared_region: %p [%d(%s)] map: "
             "fd=%d not a vnode (type=%d)\n",
             (void *)VM_KERNEL_ADDRPERM(current_thread()),
             p->p_pid, p->p_comm,
             fd, FILEGLOB_DTYPE(fp->f_fglob)));
        error = EINVAL;
        goto done;
    }

    /* we need at least read permission on the file */
```

```

if (! (fp->f_fglob->fg_flag & FREAD)) {
    SHARED_REGION_TRACE_ERROR(
        ("shared_region: %p [%d(%s)] map: "
         "fd=%d not readable\n",
         (void *)VM_KERNEL_ADDRPERM(current_thread()),
         p->p_pid, p->p_comm, fd));
    error = EPERM;
    goto done;
}

/* get vnode from file structure */
error = vnode_getwithref((vnode_t) fp->f_fglob->fg_data);
if (error) {
    SHARED_REGION_TRACE_ERROR(
        ("shared_region: %p [%d(%s)] map: "
         "fd=%d getwithref failed (error=%d)\n",
         (void *)VM_KERNEL_ADDRPERM(current_thread()),
         p->p_pid, p->p_comm, fd, error));
    goto done;
}
vp = (struct vnode *) fp->f_fglob->fg_data;

/* make sure the vnode is a regular file */
if (vp->v_type != VREG) {
    SHARED_REGION_TRACE_ERROR(
        ("shared_region: %p [%d(%s)] map(%p:'%s'): "
         "not a file (type=%d)\n",
         (void *)VM_KERNEL_ADDRPERM(current_thread()),
         p->p_pid, p->p_comm,
         (void *)VM_KERNEL_ADDRPERM(vp),
         vp->v_name, vp->v_type));
    error = EINVAL;
    goto done;
}

#if CONFIG_MACF
/* pass in 0 for the offset argument because AMFI does not need the offset
   of the shared cache */
error = mac_file_check_mmap(vfs_context_ucred(vfs_context_current()),
                            fp->f_fglob, VM_PROT_ALL, MAP_FILE, 0, &maxprot);
if (error) {
    goto done;
}
#endif /* MAC */

/* make sure vnode is on the process's root volume */
root_vp = p->p_fd->fd_rdir;
if (root_vp == NULL) {
    root_vp = rootvnode;
} else {
    /*
     * Chroot-ed processes can't use the shared_region.
     */
    error = EINVAL;
    goto done;
}

if (vp->v_mount != root_vp->v_mount) {
    SHARED_REGION_TRACE_ERROR(
        ("shared_region: %p [%d(%s)] map(%p:'%s'): "
         "not on process's root volume\n",
         (void *)VM_KERNEL_ADDRPERM(current_thread()),
         p->p_pid, p->p_comm,
         (void *)VM_KERNEL_ADDRPERM(vp), vp->v_name));
    error = EPERM;
    goto done;
}

/* make sure vnode is owned by "root" */
VATTR_INIT(&va);
VATTR_WANTED(&va, va_uid);
error = vnode_getattr(vp, &va, vfs_context_current());
if (error) {

```

```

SHARED_REGION_TRACE_ERROR(
    ("shared_region: %p [%d(%s)] map(%p:'%s'): "
     "vnode_getattr(%p) failed (error=%d)\n",
     (void *)VM_KERNEL_ADDRPERM(current_thread()),
     p->p_pid, p->p_comm,
     (void *)VM_KERNEL_ADDRPERM(vp), vp->v_name,
     (void *)VM_KERNEL_ADDRPERM(vp), error));
    goto done;
}
if (va.va_uid != 0) {
    SHARED_REGION_TRACE_ERROR(
        ("shared_region: %p [%d(%s)] map(%p:'%s'): "
         "owned by uid=%d instead of 0\n",
         (void *)VM_KERNEL_ADDRPERM(current_thread()),
         p->p_pid, p->p_comm,
         (void *)VM_KERNEL_ADDRPERM(vp),
         vp->v_name, va.va_uid));
    error = EPERM;
    goto done;
}

if (scdir_enforce) {
    /* get vnode for scdir_path */
    error = vnode_lookup(scdir_path, 0, &scdir_vp, vfs_context_current());
    if (error) {
        SHARED_REGION_TRACE_ERROR(
            ("shared_region: %p [%d(%s)] map(%p:'%s'): "
             "vnode_lookup(%s) failed (error=%d)\n",
             (void *)VM_KERNEL_ADDRPERM(current_thread()),
             p->p_pid, p->p_comm,
             (void *)VM_KERNEL_ADDRPERM(vp), vp->v_name,
             scdir_path, error));
        goto done;
    }

    /* ensure parent is scdir_vp */
    if (vnode_parent(vp) != scdir_vp) {
        SHARED_REGION_TRACE_ERROR(
            ("shared_region: %p [%d(%s)] map(%p:'%s'): "
             "shared cache file not in %s\n",
             (void *)VM_KERNEL_ADDRPERM(current_thread()),
             p->p_pid, p->p_comm,
             (void *)VM_KERNEL_ADDRPERM(vp),
             vp->v_name, scdir_path));
        error = EPERM;
        goto done;
    }
}

/* get vnode size */
error = vnode_size(vp, &fs, vfs_context_current());
if (error) {
    SHARED_REGION_TRACE_ERROR(
        ("shared_region: %p [%d(%s)] map(%p:'%s'): "
         "vnode_size(%p) failed (error=%d)\n",
         (void *)VM_KERNEL_ADDRPERM(current_thread()),
         p->p_pid, p->p_comm,
         (void *)VM_KERNEL_ADDRPERM(vp), vp->v_name,
         (void *)VM_KERNEL_ADDRPERM(vp), error));
    goto done;
}
file_size = fs;

/* get the file's memory object handle */
file_control = ubc_getobject(vp, UBC_HOLDOBJECT);
if (file_control == MEMORY_OBJECT_CONTROL_NULL) {
    SHARED_REGION_TRACE_ERROR(
        ("shared_region: %p [%d(%s)] map(%p:'%s'): "
         "no memory object\n",
         (void *)VM_KERNEL_ADDRPERM(current_thread()),
         p->p_pid, p->p_comm,
         (void *)VM_KERNEL_ADDRPERM(vp), vp->v_name));
    error = EINVAL;
}

```

```

        goto done;
    }

/* check that the mappings are properly covered by code signatures */
if (!cs_system_enforcement()) {
    /* code signing is not enforced: no need to check */
} else for (i = 0; i < mappings_count; i++) {
    if (mappings[i].sfm_init_prot & VM_PROT_ZF) {
        /* zero-filled mapping: not backed by the file */
        continue;
    }
    if (ubc_cs_is_range_codesigned(vp,
                                    mappings[i].sfm_file_offset,
                                    mappings[i].sfm_size)) {
        /* this mapping is fully covered by code signatures */
        continue;
    }
}

SHARED_REGION_TRACE_ERROR(
    ("shared_region: %p [%d(%s)] map(%p:'%s'): "
     "mapping #%"PRIu32"/%"PRIu32" [0x%"PRIx64":0x%"PRIx64":0x%"PRIx64":0x%"PRIx64":0x%"PRIx64"] "
     "is not code-signed\n",
     (void *)VM_KERNEL_ADDRPERM(current_thread()),
     p->p_pid, p->p_comm,
     (void *)VM_KERNEL_ADDRPERM(vp), vp->v_name,
     i, mappings_count,
     mappings[i].sfm_address,
     mappings[i].sfm_size,
     mappings[i].sfm_file_offset,
     mappings[i].sfm_max_prot,
     mappings[i].sfm_init_prot));
error = EINVAL;
goto done;
}

/* get the process's shared region (setup in vm_map_exec()) */
shared_region = vm_shared_region_trim_and_get(current_task());
if (shared_region == NULL) {
    SHARED_REGION_TRACE_ERROR(
        ("shared_region: %p [%d(%s)] map(%p:'%s'): "
         "no shared region\n",
         (void *)VM_KERNEL_ADDRPERM(current_thread()),
         p->p_pid, p->p_comm,
         (void *)VM_KERNEL_ADDRPERM(vp), vp->v_name));
    error = EINVAL;
    goto done;
}

/* map the file into that shared region's submap */
kr = vm_shared_region_map_file(shared_region,
                               mappings_count,
                               mappings,
                               file_control,
                               file_size,
                               (void *) p->p_fd->fd_rdir,
                               slide,
                               slide_start,
                               slide_size);

if (kr != KERN_SUCCESS) {
    SHARED_REGION_TRACE_ERROR(
        ("shared_region: %p [%d(%s)] map(%p:'%s'): "
         "vm_shared_region_map_file() failed kr=0x%"PRIx32"\n",
         (void *)VM_KERNEL_ADDRPERM(current_thread()),
         p->p_pid, p->p_comm,
         (void *)VM_KERNEL_ADDRPERM(vp), vp->v_name, kr));
    switch (kr) {
    case KERN_INVALID_ADDRESS:
        error = EFAULT;
        break;
    case KERN_PROTECTION_FAILURE:
        error = EPERM;
        break;
    case KERN_NO_SPACE:
        error = ENOMEM;
    }
}

```

```

        break;
    case KERN_FAILURE:
    case KERN_INVALID_ARGUMENT:
    default:
        error = EINVAL;
        break;
    }
    goto done;
}

error = 0;

vnode_lock_spin(vp);

vp->v_flag |= VSHARED_DYLD;

vnode_unlock(vp);

/* update the vnode's access time */
if (! (vnode_vfsvisflags(vp) & MNT_NOATIME)) {
    VATTR_INIT(&va);
    nanotime(&va.va_access_time);
    VATTR_SET_ACTIVE(&va, va_access_time);
    vnode_setattr(vp, &va, vfs_context_current());
}

if (p->p_flag & P_NOSHLIB) {
    /* signal that this process is now using split libraries */
    OSBitAndAtomic(~((uint32_t)P_NOSHLIB), &p->p_flag);
}

done:

if (vp != NULL) {
    /*
     * release the vnode...
     * ubc_map() still holds it for us in the non-error case
     */
    (void)vnode_put(vp);
    vp = NULL;
}
if (fp != NULL) {
    /* release the file descriptor */
    fp_drop(p, fd, fp, 0);
    fp = NULL;
}
if (scdir_vp != NULL) {
    (void)vnode_put(scdir_vp);
    scdir_vp = NULL;
}

if (shared_region != NULL) {
    vm_shared_region_deallocate(shared_region);
}

SHARED_REGION_TRACE_DEBUG(
    ("shared_region: %p [%d(%s)] <- map\n",
     (void *)VM_KERNEL_ADDRPERM(current_thread()),
     p->p_pid, p->p_comm));

return error;
}

```

And by stepping through this function in the kernel debugger, I found that the failure is in this part:

```

/* make sure vnode is owned by "root" */
VATTR_INIT(&va);
VATTR_WANTED(&va, va_uid);
error = vnode_getattr(vp, &va, vfs_context_current());
if (error) {
    SHARED_REGION_TRACE_ERROR(

```

```

        ("shared_region: %p [%d(%s)] map(%p:'%s'): "
         "vnode_getattr(%p) failed (error=%d)\n",
         (void *)VM_KERNEL_ADDRPERM(current_thread()),
         p->p_pid, p->p_comm,
         (void *)VM_KERNEL_ADDRPERM(vp), vp->v_name,
         (void *)VM_KERNEL_ADDRPERM(vp), error));
    goto done;
}
if (va.va_uid != 0) {
    SHARED_REGION_TRACE_ERROR(
        ("shared_region: %p [%d(%s)] map(%p:'%s'): "
         "owned by uid=%d instead of 0\n",
         (void *)VM_KERNEL_ADDRPERM(current_thread()),
         p->p_pid, p->p_comm,
         (void *)VM_KERNEL_ADDRPERM(vp),
         vp->v_name, va.va_uid));
    error = EPERM;
    goto done;
}

```

And the problem is that the cache file isn't owned by root, so I found a way to mount the ramdisk on OSX with file permissions enabled, and `chown`ed the file. This made bash work! Now we only have stdout, but no input support.

UART interactive I/O

Now, all that's left is enabling UART input. After going over and reversing some of the serial I/O handling code I found this place, which decides whether to enable the UART input (which is off by default):



This code reads a global value, and checks bit #1. If it is on, UART input is enabled. By examining this global, we can see that it is set here:

```

...ff0071bad54 85 d9 0f 94 bl    _serial_init
...ff0071bad58 a8 1f 00 f0 adrp x8,-0xffff8af4f000
...ff0071bad5c 08 a1 04 91 add x8,x8,#0x128
...ff0071bad60 09 b0 f2 10 adr x9,-0xffff8e5fc0a
...ff0071bad64 1f 20 03 d5 nop
...ff0071bad68 1f 00 00 71 cmp uVar9_random,#0x0
...ff0071bad6c 28 01 88 9a csel x8,x9,x8,eq
...ff0071bad70 49 22 00 b0 adrp x9,-0xffff8f89fd000
...ff0071bad74 28 b1 03 f9 str x8=>_serial_putc,[x9, #0x760]>=_PE_kputr
...ff0071bad78 01 f6 ff f0 adrp x1,-0xffff8f83000
...ff0071bad7c 21 80 1d 91 add x1=>DAT_ffffff00707d760,x1,#0x760
...ff0071bad80 3f 00 00 b9 str wzr,[x1]=>DAT_ffffff00707d760
...ff0071bad84 e0 f3 ff 90 adrp uVar9_random,-0xffff8fc0a000
...ff0071bad88 00 e0 20 91 add uVar9_random=>s_serial_ffffff00703683t
...ff0071bad8c e2 03 1e 32 mov w2,#0x4
...ff0071bad90 03 00 80 52 mov w3,#0x0
...ff0071bad94 24 d3 0f 94 bl PE_parse_boot_argn
...ff0071bad98 40 02 00 34 cbz uVar9_random,LAB_ffffff0071bade0
...ff0071bad9c 13 f6 ff f0 adrp x19,-0xffff8fb3000
...ff0071badad 68 62 47 b9 ldr w8,[x19, #0x760]=>DAT_ffffff00707d760
...ff0071bad4 09 01 1e 12 and w9,w8,#0x4
...ff0071bad8 09 53 00 b9 str w9,[sp, #local_150_physmap_base]
...ff0071badac 68 01 10 37 tbnz w8,#0x2,LAB_ffffff0071badd8
...ff0071badbe e0 f3 ff 90 adrp uVar9_random,-0xffff8fc0a000
...ff0071badb4 00 fc 20 91 add uVar9_random=>s_drain_uart_sync_ffffff
...ff0071badb8 e2 03 1e 32 mov w2,#0x4
...ff0071badbc e1 43 01 91 add x1,sp,#0x50
...ff0071badcc 03 00 80 52 mov w3,#0x0
...ff0071badc4 18 d3 0f 94 bl PE_parse_boot_argn
...ff0071badc8 c8 00 00 34 cbz uVar9_random,LAB_ffffff0071bade0
...ff0071badcc e8 53 40 b9 ldr w8,[sp, #local_150_physmap_base]
...ff0071baddd 88 00 00 34 cbz w8,LAB_ffffff0071bade0

```

/* WARNING: Subroutine does not return */
 iVar9 = PE_parse_boot_argn("debug",&local_150_physmap_base,4,0);
 DAT_ffffff0076742c8 = 0x11;
 DAT_ffffff0076742c0 = 0;
 DataMemoryBarrier(2,3);
 iVar9 = PE_parse_boot_argn("debug",&local_150_physmap_base,4,0);
 iVar9 = _serial_init();
 _PE_kputc = FUN_ffffff0071a0360;
 if ((iVar9 != 0) && (((byte)local_150_physmap_base >> 3 & 1) != 0)) {
 DAT_ffffff007095cf0 = 0;
 }
 iVar9 = _serial_init();
 _PE_kputc = _serial_putc;
 if ((iVar9 != 0) && (((byte)local_150_physmap_base >> 3 & 1) != 0)) {
 iVar9 = PE_parse_boot_argn("serial",&DAT_ffffff00707d760,4,0);
 if ((iVar9 != 0) && (((longlong *)((ulonglong)local_150_physmap_base & 0xffffffff00000000 |
 ((ulonglong)DAT_ffffff00707d760 & 0xffffffff00000004),
 (DAT_ffffff00707d760 >> 2 & 1) != 0 || ((iVar9 = PE_parse_boot_argn("drain_uart_sync",&local_150_physmap_base,4,0), iVar9 != 0)))))) {
 DAT_ffffff00707d760 = DAT_ffffff00707d760 | 4;
 }
 if (DAT_ffffff007607910 == 0) {
 local_150_physmap_base = (longlong *)((ulonglong)local_150_physmap_base & 0xfffffff00000000);
 iVar9 = PE_parse_boot_argn("validation_disables",&local_150_physmap_base,4,0);
 iVar9 = local_150_physmap_base._0_4_ = DAT_ffffff007607910;
 if (iVar9 != 0) {
 local_150_physmap_base = (longlong *)((ulonglong)local_150_physmap_base | 1);
 }
 }
 }
}

The value is taken from the `serial` boot arg. So finally, by setting the `serial` boot arg to `2` (bit #1 on) I got an interactive bash shell!

Conclusion

This project was a lot of fun and the team plans to keep working on it and expand the features displayed out in the previous post. This post shows some of the details of some of the interesting parts of the work, but as I worked very sparsely on this project for the past few months, and documentation during the research lacked some details, not all the details are here. With that in mind, all the functionality is in the code which does include comments in non-trivial places. I got into iOS internals only through this project, so some parts can possibly be improved, and I already received some improvement suggestions. If you have any comments/ideas/suggestions for this project, please contact me.