1,000 bugs in your pocket

__gsch & q3k & nyan_satan

0x41con 2025

What is this even about?!

whoarewe

__gsch

- Newbie iOS vulnerability researcher at Paradigm Shift
- Corporate world escapee
- Rust and fuzzing enthusiast

nyan_satan

- Software developer & reverse engineer (sorta)
- Love to tinker with low-level iOS stuff
  - You might have heard about me in context of running DOOM on Lightning to HDMI dongle & other whacky experiments
- Committed a lot of DMCA crimes against Apple Inc.

q3k

- Absent physically, but present in spirit
- HW/SW hacker, mostly RE of obscure things

Agenda

- Introduction
  - Motivation
  - Hardware
  - Software
- Exploits
  - Userspace & ROM
- Demos
- Miscellaneous

Motivation

Because we *can*!

(they're cute!)

Hardware

Timeline

iPod (1st generation)
PP5002

2001

iPod (2nd generation)
PP5002

2002

iPod (3rd generation)
PP5002

2003

iPod (4th gen/Photo)
PP5002

2004

iPod mini (1st generation)
PP5020

iPod mini (2nd generation)
PP5020

iPod video
PP5021C

2005

iPod nano (1st generation)
PP5021C

iPod shuffle (1st generation)
STMP 3550

iPod shuffle (2nd generation)
S5L8441

2006

iPod nano (2nd generation)
S5L8701

iPod classic
S5L8702

2007

iPod nano (3rd generation)
S5L8702

Timeline (cont.)

iPod shuffle (2nd generation)
S5L8441

iPod shuffle (3rd generation)
S5L8442

iPod classic
S5L8702

iPod shuffle (3rd generation)
S5L8443

2006 2007 2008 2009 2010 2012

iPod nano (2nd generation)
S5L8701

iPod nano (4th generation)
S5L8720

iPod nano (6th generation)
S5L8723

iPod nano (7th generation)
S5L8740

iPod nano (3rd generation)
S5L8702

iPod nano (5th generation)
S5L8730

Scope

- We will only focus on the iPods from 2006 onwards in this talk
- This is when they switched from PortalPlayer SoCs to Samsung
- Shortly after iPod nano 2 & iPod shuffle 2 release in September 2006, Apple released yet another device with a Samsung SoC…

Scope (cont.)

- Non-iOS iPod hacking story partially crosses paths with the story of the 1st-gen iOS devices
- All because of the similarities in hardware and (a bit of) software
  - iPod nano 4 & iPod touch 2 even used the same SoC - S5L8720
    - Less DRAM on the nano though - 32 MiB vs. 128 MiB
    - Also the iPod touch switched to iBoot-based bootrom aka SecureROM

Timeline (SoCs)

iPhone
iPod touch

iPhone / iPod touch
S5L8900

iPod touch 2
S5L8720

Apple Lightning video adapters
S5L8747

iPod nano
iPod classic

nano 2
S5L8701

nano 3
S5L8702

nano 4
S5L8720

nano 5
S5L8730

nano 6
S5L8723

nano 7
S5L8740

2006 | 2007 | 2008 | 2009 | 2010 | 2012

classic
S5L8702

iPod shuffle

shuffle 2
S5L8441

shuffle 3
S5L8442

shuffle 4
S5L8443

Scope (cont.)

- Starting with iPhone 3GS (2009, S5L8920), the SoCs started becoming more and more Apple's than Samsung's
  - iPhone 5 (2012, S5L8950) was the first with Apple's own core design - Swift
  - Few years later Samsung stopped being involved even in their manufacturing
- The remaining non-iOS iPod models stayed more or less the same as the old Samsung chips, however

Specs

- ARM SoC, single-core
  - ARM9 on the older models
  - ARM11 on the newer models
  - ARM Cortex-A5 on the latest iPod shuffle & nano
- 32 or 64 MiB of DRAM
- Often the same (or similar) peripherals compared to iOS-world SoCs from that era
- NAND storage for data, older devices also have NOR for config/ bootloader

Software

ROM BOOT

- Samsung's in-house BootROM
- Loads Image1 from NOR/NAND or USB DFU into SRAM
  - A simple container that provides metadata for optional signing and encryption
    - But not personalization
  - Known as `8900` image format in iOS world
- Biggest focus of the community in the past few years

EFI bootloaders

- Loaded from NOR/NAND or DFU (WTF)
  - “What’s The Firmware” and not what you thought
- EFI firmware volume with runtime
  - Split up into drivers
- Can chainload into blobs or load further EFI payloads (e.g. Diags)
- Loads Image1 from NOR/NAND or DFU into DRAM
- Payloads:
  - RetailOS (usually NAND)
  - Disk Mode (NAND or DFU)
  - Diags (usually NAND)

RetailOS/OSOS

- User-facing interface
- Barely an OS: based on RTXC runtime, no privsep, single blob linked together
- On iPods with Click Wheel can load signed/encrypted games (‘eApp’)
- An absolute pain to RE: a massive blob of C++
- Only some RTTI symbols/debug, barely any logging

Exploitation

Notes exploit

- Untethered RetailOS bug
  - Buffer overflow in HTML parser
    - Used for notes uploaded via iTunes
- Discovered by cmwslw/Linux4Nano project in 2009
- Vulnerable: iPod classic & nano 2 - 4
- Patched in nano 5 after public release
- Offsets needed bruteforcing: crowdsourced

Pwnage 2.0

- Untethered ROM BOOT vulnerability in X.509 parsing code
  - A very stupid stack buffer overflow upon copying certificate signature
- Discovered by iPhone Dev Team, then ported
- Vulnerable:
  - iPod classic & nano 2 - 4
  - iPhone & iPhone 3G
  - iPod touch
- Patched in iPod nano 5
  - SecureROM (iOS) never had it in the first place

wInd3x

- Tethered ROM BOOT vulnerability in USB stack
- Discovered by q3k in late 2021
- Vulnerable:
  - iPod classic & nano 3
  - iPod nano 4
  - iPod nano 5 - had no public exploits before
- Found internally and patched in iPod nano 6

wInd3x (the bug)

- Upon handling certain kinds of USB control requests, `wIndex` field is directly used to load a function pointer from an array
  - `wIndex` is one of USB control request properties that we fully control
- Needless to say, there is no range check

wInd3x (the exploit)

- The array is populated into a global state structure
  - ROM BOOT puts all kinds of stuff into there
- Shortly after the array, there is a counter that signifies amount of bytes left to upload
  - In USB DFU terms, *upload* is getting data from target to host
- By initiating an upload and then timing it out on the host side, we can have any value between 0x0 and 0x600 in the counter

wInd3x (the exploit, cont.)

- Since it's good ole ARMv6, 0x0 has exception vectors and in this case the entire ROM
- Our gadget of interest is `blx r0`, which is located in that range
- The function will be called with a pointer to raw USB packet data as an argument
  - `r0` is the first argument as per ABI
  - So basically we'll execute the packet header

wInd3x (the exploit, cont.)

- We can make an ARM instruction and USB packet polyglot
  - Packet data - `0x20 0xfe 0xff 0xea 0x03 0x00 0x00 0x00`

```
bmRequestType bRequest wValue wIndex wLength
```

```
$ rasm2 -a arm -b 32 -o 0x2202e300 -D "20feffea03000000"
0x2202e300   4                 20feffea  b 0x2202db88
0x2202e304   4                 03000000  andeq r0, r0, r3
```

- The branch then goes into temporary data buffer which we obviously fully control

Image Swap Bug

- Discovered by Alyssa Rosenzweig over a decade ago
- You can flash the device with modified resource payloads, but it'll fail signature checks on boot
- 2nd-stage bootloader checks resources if booting RetailOS. On failure, boots Disk Mode for recovery
- Trick: swap Disk Mode with RetailOS on NAND, force Disk Mode
  - Now you have RetailOS without resource signature checking!
- Custom images, fonts… and font parsing bugs?

ipod_sun

- Untethered RetailOS exploit chain
- Written by CUB3D around Dec 2023
- Vulnerable: all Samsung SoC iPods (except shuffles)
  - The implementation only supports nano 6 & 7 (2015) as of today
    - Hack - you can force 2012 iPod nano 7 to run 2015's firmware
- Ported from comex's CVE-2010-1797 FreeType bug used in JailbreakMe 2.0/Star
  - Uses the image swap bug to inject the malformed font

S5Late

- Tethered ROM BOOT vulnerability in DFU code
- Discovered by __gsch in late 2024
- Vulnerable: all ROM BOOT based devices
- On DFU_DNLOAD, multiple missing or erroneous checks on the amount of data received lead to an overflow of the temporary USB DFU buffer

S5Late (the bug)

```
1  void __fastcall dfu_boot_handle_class(usb_device_request *req)
2  {
3    wLength = req->wLength;
4    switch ( req->bRequest )
5    {
6      case DFU_DNLOAD:
7        bState = gState->dfu_status.bState;
8        if ( bState != dfuIDLE || bState != dfuDNLOAD_IDLE)
9          goto ERROR_STALLEDPKT;
10       // [1] wLength checked against temporary buffer size
11       if ( gState->dfu_buf_size < wLength )
12         goto ERROR_NOTDONE;
13       if ( !req->wValue )
14       {
15         gState->dfu_transfered_bytes = 0;
16         gState->dfu_buf_offset = 0;
17         gState->usb_upload_complete = 0;
18         gState->crc = -1;
19       }
20       if ( wLength )
21       {
22         // [2] write wLength bytes to temporary buffer plus offset
23         usb_ep0_out_recv_sync_buf(&gState->dfu_buf[gState->dfu_buf_offset], wLength);
24         gState->dfu_buf_offset += wLength;
25         gState->dfu_status.bStatus = errOK;
26         gState->dfu_status.bState = dfuDNLOAD_SYNC;
27         usb_ep0_in_transmit_buf(0, 0);
28       }
29       else
30       {
31         gState->usb_upload_complete = 1;
32         gState->dfu_status.bStatus = errOK;
33         gState->dfu_status.bState = dfuMANIFEST_SYNC;
34         usb_ep0_in_transmit_buf(0, 0);
35       }
36       return;
```

```
37     case DFU_GETSTATUS:
38       if ( wLength != 6 )
39         goto STALL_ENDPOINT;
40       bState = gState->dfu_status.bState;
41       if ( bState != dfuDNLOAD_SYNC )
42       {
43         goto DFU_MANIFEST_SYNC;
44       }
45       dfu_buf_offset = gState->dfu_buf_offset;
46       // [3] If temporary buffer not yet full continue DFU_DNLOAD
47       if ( dfu_buf_offset != gState->dfu_buf_size && !gState->usb_upload_complete )
48       {
49         gState->dfu_status.bwPollTimeout[0] = 0;
50         gState->dfu_status.bwPollTimeout[1] = 0;
51         gState->dfu_status.bwPollTimeout[2] = 0;
52         gState->dfu_status.bState = dfuDNLOAD_IDLE;
53         usb_ep0_in_transmit_buf(&gState->dfu_status, 6u);
54         return;
55       }
56       if ( gState->dfu_buf_offset )
57         gState->dfu_status.bwPollTimeout[0] = 0xA;
58       else
59         gState->dfu_status.bwPollTimeout[0] = 0;
60       gState->dfu_status.bwPollTimeout[1] = 0;
61       gState->dfu_status.bwPollTimeout[2] = 0;
62       gState->dfu_status.bState = dfuDNBUSY;
63       usb_ep0_in_transmit_buf(&gState->dfu_status, 6u);
64       // [4] Copy temporary buffer to image buffer and check size
65       if ( gState->dfu_on_download_chunk(gState) )
66       {
67         gState->dfu_status.bStatus = errADDRESS;
68         gState->dfu_status.bState = dfuIDLE;
69         usb_ep0_in_transmit_buf(&gState->dfu_status, 6u);
70         return;
71       }
72       gState->dfu_transfered_bytes += gState->dfu_buf_offset;
```

S5Late (the exploit)

- We can overwrite until the end of SRAM
- In the very end of SRAM we have 4 uint32's:

  `0xBA581043 0x00000018 0x2202BA3C 0x2202D900`

- `0x2202D900` is a pointer to heap
- `0x2202BA3C` is a pointer to the global state structure we mentioned before
- `0x00000018` is set before jumping to the next stage
  - Likely to communicate boot mode

S5Late (the exploit, cont.)

- `0xBA581043` is something really weird
  - No software ever accesses it
    - Or we failed to find references
  - Few bits change between tethered and untethered boot
    - `0xBA581067` vs. `0xBA581043`
  - Putting zeroes/garbage there makes everything fall apart
  - We are really unsure what it is

S5Late (the exploit, cont.)

- We can dump the global state struct and the pointers from the end of SRAM using **ipod_sun** (or others) since it's not cleared
- We flip a few fields in the dumped state and switch to it by overwriting the pointer at the bottom of SRAM
  - Various enums/flags
  - `on_upload()` callback

Diags

- Diags has strings which look like a command line REPL
- When booted untethered (via button combo): no serial shell, only boring yellow menu on display

Diags (cont.)

- But if you boot it over WTF… (no exploit needed!)

```
Apple 'Snowfox' diagnostics shell
menu returned Load Error error
:-) help
EDK Boot Loader commands (help command for more info):
 abort: ; Exit EBL
 accel: Test accelermeter in fct &fatp station
 accessory:     Command to change power settings as well as identifying accessories connected to the unit
 accuart:       Command to test accessory uart
 audioinit:     initialize the audio subsystem: run before all audio commands
 bl:     Change Backlight Level
 blockdevice:   ; Show information about boot devices
 boardid:       ; Get the board ID
 bootcfg:       ; Get the boot configuration
 btncount:      button counter test
 button0:       Test buttons without menu key of the unit
 buttonlist:    Test buttons on the unit
 buttons:       ; reports which Vol+/-/play buttons are pressed down
 buttont:       [--alt] Test buttons on the unit
 charge:        set USB current limit
 chgcur:        Realtime display of battery current and voltage
 chipid:        ; Show soc chip ID
 clcd:  [--id][--iqc]   --id: LCD panel id, vendor id
        --iqc:choice,print the result in LCD for iqc test
```

Only confirmed to properly work on iPod nano 7 though

Diags (cont.)

- WTF has EFI serial driver, the normal bootloader doesn’t
- The shell has all kinds of debug commands, including arbitrary memory RW
- Is this even a bug?!

What Went Wrong

- BootROM complexity is how you get unpatchable bugs
  - Pwnage 2.0, wInd3x, S5Late
  - This particular codebase is *really* bad
- Image1 has no ‘intent’ signature field: confusion from mixing and matching payloads
  - Image Swap Bug, the Diags trick
- Using FreeType:
  - ipod_sun

Demo!

Demo went here

Linux/Rockbox ports

- Technically now all doable (as shown in the demo)
- Reminder: there are no SoC docs and REing RetailOS is a pain
- NAND driver is holding us back: FTL layer fully implemented in software, custom ISA for NAND command sequencing
- Linux Proof Of Concept on iPod nano 5 & 7
- Early Rockbox bootloader port for iPod nano 3 (including some FTL code)

iPod shuffle

What are you?

- iPod shuffle is the dumbest of all iPods
  - No screen, only buttons and a LED
- Firmwares, although tiny (~1 MiB), look similar to the ones for bigger iPods
  - Payloads are encrypted…
  - …but containers' metadata shows that they use the same Image1 containers for bootloaders, including the DFU one
  - Certificate data yields similar SoC names - S5L8442 & etc.

What are you? (cont.)

- My first thought was to check if iPod shuffle 3 ROM is vulnerable to wInd3x or even Pwnage 2.0
  - The device was released in early 2009
  - Should be similar to iPod nano 5 which is vulnerable to wInd3x and was released late in the same year
  - WTF Image1 is even version 1.0 with 0x800 padding that matches 2007 devices
  - So it was decided to try Pwnage 2.0 first

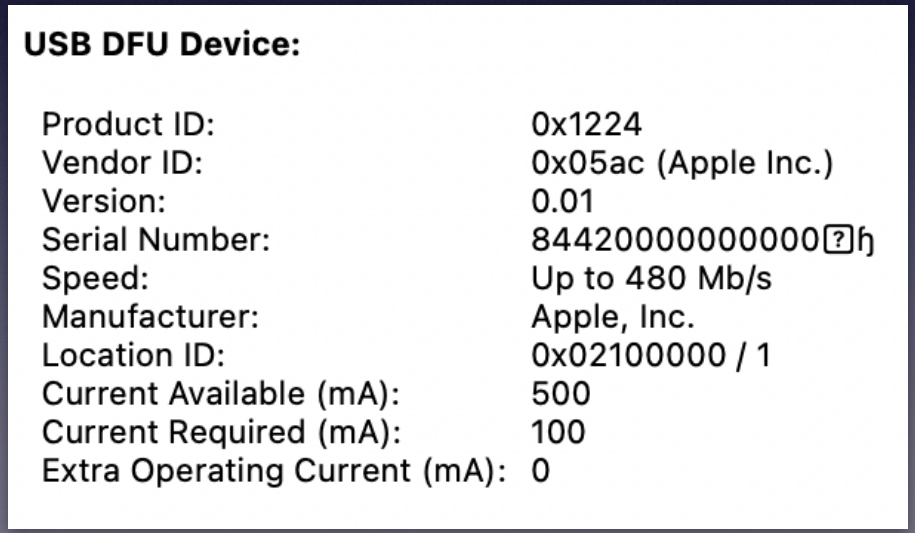
iPod shuffle 3

iPod shuffle 3 DFU

- This device is even dumber than the rest of the shuffles
  - No buttons at all - only a 3-position switch
  - Playback is meant to be controlled via headphones
- So the question is - *how do you even put it into DFU mode?*

iPod shuffle 3 DFU (cont.)

- Turns out it's enough to flash firmware with a broken bootloader image
  - The bootrom is then unable to load next stage and enters DFU



S5L8442 Pwnage 2.0

- Pwnage 2.0 was proven to be there
  - Crafted certificate made the ROM completely hang
- The original S5L8900 (1st-gen iOS devices) exploit didn’t use any shellcode
  - LR was set to the epilogue of the caller function
  - R4 was set to 1 and then moved to R0 (return value)
  - Thus an image was considered trusted
  - The ROM would continue loading the image normally

S5L8442 Pwnage 2.0 (cont.)

- The problem is that I had neither the ROM dump nor a decrypted 2nd-stage bootloader
- I had an iPhone 3G (S5L8900) and its' ROM dump, however
- Given that the codebase & memory layout are very similar, I was able to write a payload that could find certain offsets dynamically
- I used it to patch USB string descriptors and restart DFU
- This way I could extract a few bytes of data at a time
    - Dumping the ROM took ~6-7 hours

S5L8442 Pwnage 2.0 Post-exp

- Instead of jumping to the next stage bootloader, I took the “pwned” DFU approach
  - Executing custom code in the context of ROM and sending my own commands over USB
- Hijacking `on_upload` callback was proven to be a good way
  - It doesn’t do anything originally
- At that point I could dump any memory, as well as drive SoC’s AES engine and thus decrypt the firmware

S5L8442 Firmware

- It doesn’t seem to use EFI for anything
- Logging is done via semihosting
  - External debugger traps SVCs and extracts log string
- Apparently there is no DRAM - only 512 KiB of SRAM
- No firmware ever accesses ARM CP15 registers
  - No MMU at all?
  - No MIDR (Main ID Register) either then
  - The hell is this core?

iPod shuffle 4

iPod shuffle 4 DFU

- We got the buttons back!
  - But there's still no DFU combination it seems
  - Luckily the broken firmware trick still works
- USB DFU serial number string reports iPod nano 6 chip ID - 8723

USB DFU Device:

Product ID:	0x1233
Vendor ID:	0x05ac (Apple Inc.)
Version:	0.01
Serial Number:	87230000000001
Speed:	Up to 480 Mb/s
Manufacturer:	Apple, Inc.
Location ID:	0x02100000 / 1
Current Available (mA):	500
Current Required (mA):	100
Extra Operating Current (mA):	0

iPod shuffle 4 exploitation

- Released the same day as iPod nano 6
- The ROM seems to be very close to S5L8723 as well
  - No Pwnage 2.0
  - No wInd3x
- S5Late should work then
  - My prior port for iPod nano 6 works almost 1:1

iPod shuffle 4 CPU

- It is just Cortex-A5 according to MIDR
  - Very weird given that iPod nano 6 is still ARM1176
  - My shuffle 4 is in 2012 color
  - Was there also a hardware revision?

iPod shuffle 2

iPod shuffle 2 DFU

- The very first Samsung SoC iPods - nano 2 & shuffle 2 do not enter the true DFU by a button combination
  - The broken firmware trick doesn’t seem to work
  - MLB test points to the rescue!

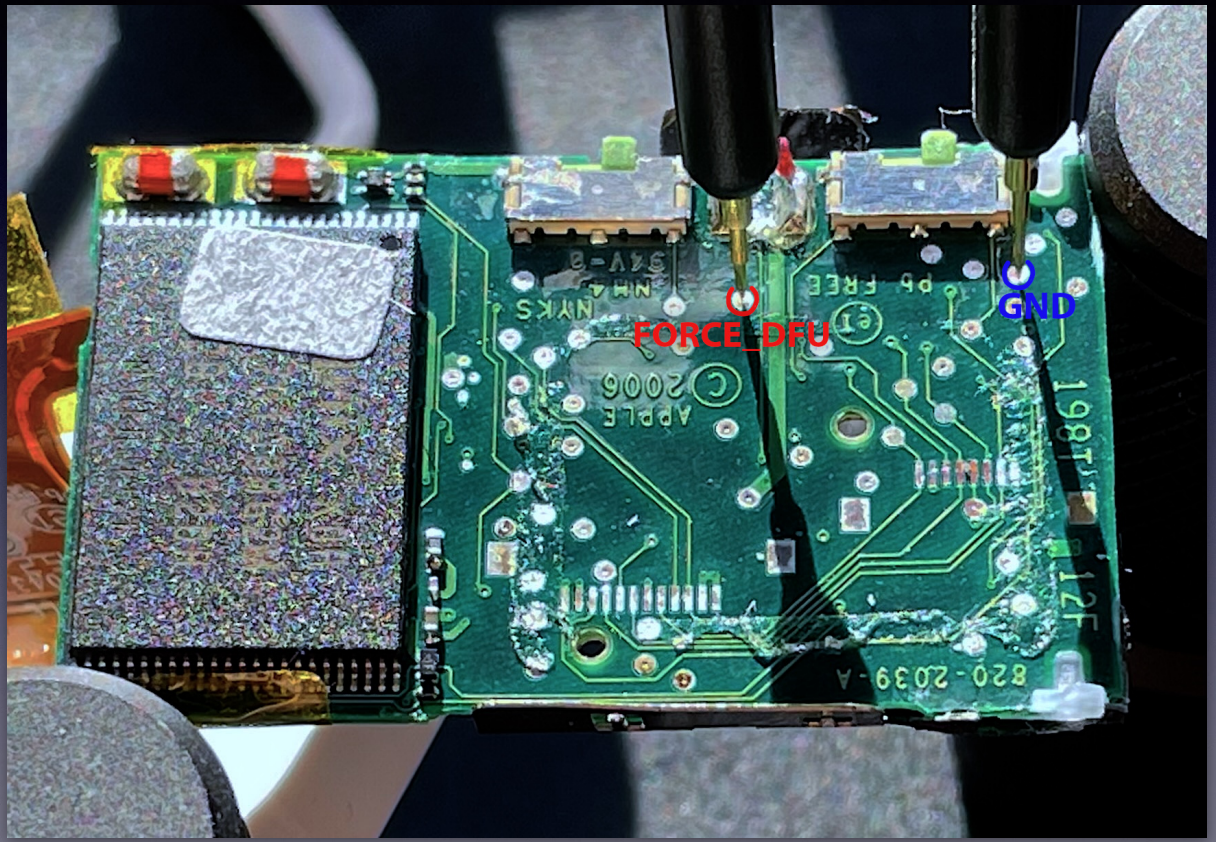
USB DFU Device:

Product ID:	0x1220
Vendor ID:	0x05ac (Apple Inc.)
Version:	0.01
Serial Number:	87010000000001
Speed:	Up to 480 Mb/s
Manufacturer:	Apple Computer, Inc.
Location ID:	0x01110000 / 3
Current Available (mA):	500
Current Required (mA):	100
Extra Operating Current (mA):	0

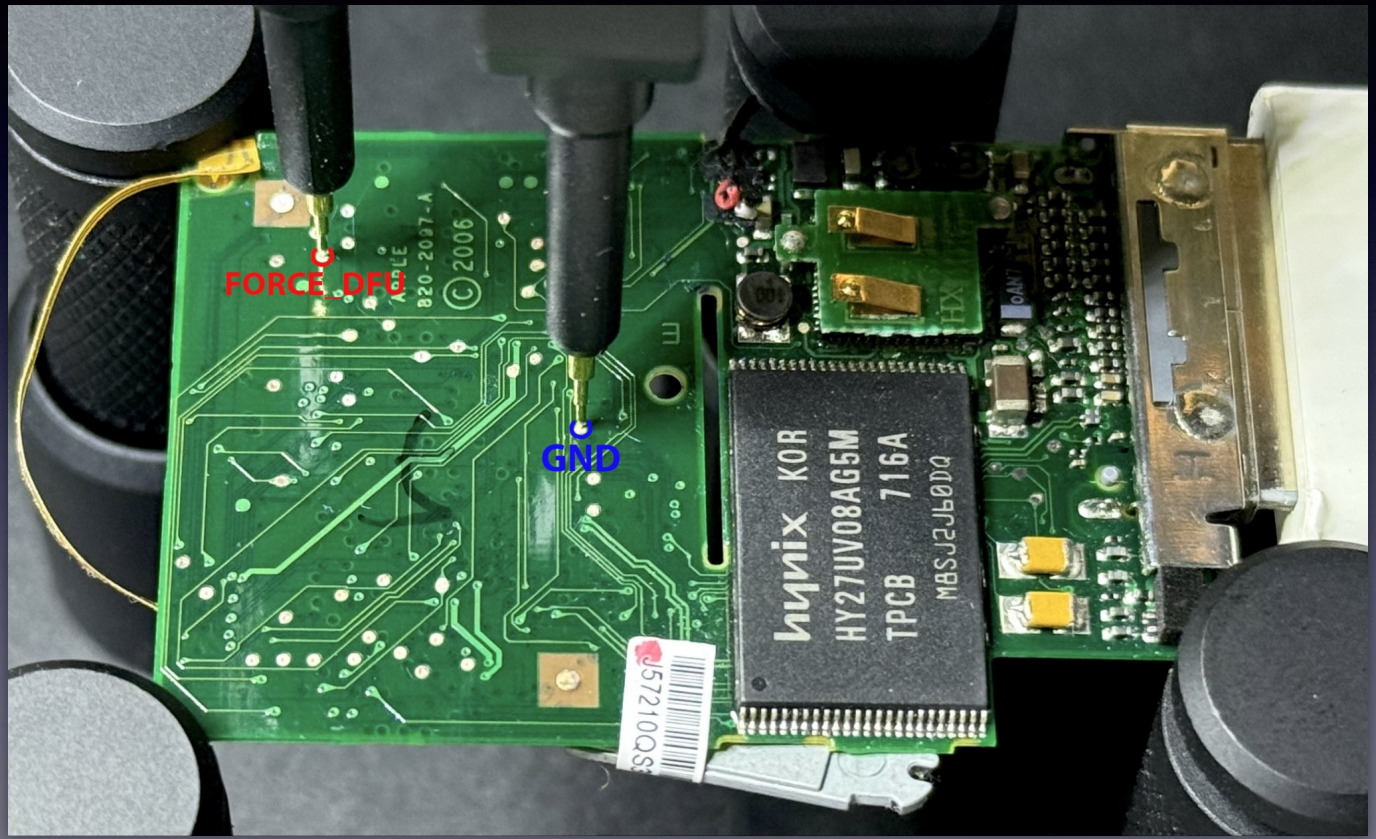
USB DFU Device:

Product ID:	0x1221
Vendor ID:	0x05ac (Apple Inc.)
Version:	0.01
Serial Number:	8441010000000001
Speed:	Up to 480 Mb/sec
Manufacturer:	Apple Computer, Inc.
Location ID:	0x26200000 / 2
Current Available (mA):	500
Current Required (mA):	100

iPod shuffle 2 FORCE_DFU



iPod nano 2 FORCE_DFU



Yet another ROM bug

One ancient file you can find online - `libipoddfu.py` - has the following piece of code:

```
119    def upload(self, data, exploit = 0):
120      if exploit == 1 and self.generation == 2 and self.type == 1:
121        data = f.read().ljust(0x200f0, "\0") \
122             + "\xb8\x48\x02\x22\0\0\0\0\0\0\0\0\0\0\0\0\0\0" \
123             + "\0\0\0\x22\0\0\0\x22\0\0\0\x22\0\0\0\x22" \
124             + "\0\0\0\x22\0\0\0\x22\0\0\0\x22\0\0\0\x22" \
125             + "\0\0\0\x22\0\0\0\x22\0\0\0\x22\0\0\0\x22" \
126             + "\0\0\0\x22\0\0\0\x22\0\0\0\x22\0\0\0\x22" \
127             + "\0\0\0\x22\0\0\0\x22\0\0\0\x22\0\0\0\x22" \
128             + "\0\0\0\x22\0\0\0\x22\0\0\0\x22\0\0\0\x22" \
129             + "\0\0\0\x22\0\0\0\x22\0\0\0\x22\0\0\0\x22" \
130             + "\0\0\0\x22\0\0\0\x22\0\0\0\x22\0\0\0\x22"
```

Yet another ROM bug (cont.)

- This is beginning of USB DFU upload function
- The algorithm is the following:
  - If exploit is requested and the target is iPod nano 2 in ROM DFU:
    - Advance user supplied data to 0x200F0
    - Append a bit more of specific data
    - Continue as usual

Yet another ROM bug (cont.)

- `on_download` callback of iPod nano 2 doesn't do range checks *at all*
  - Unlike S5Late, where we attack the temporary DFU buffer, this allows overflowing the destination
  - No tricks required - you just continue to send data as per USB DFU spec and eventually go out of load area's bounds
    - Immediately after there are DATA, BSS and eventually stacks
    - This implementation just sets the entirety of exception handlers table to 0x22000000 - beginning of the load area
    - Payload is meant to be provided by end user

Yet another ROM bug (cont.)

- The bug doesn’t seem to have a name
- It’s unknown who found it either
  - The file has copyright of TheSeven (2010)
- There’s no information online, apart from the file itself and its’ older version
  - Or I just couldn’t find it

iPod shuffle 2

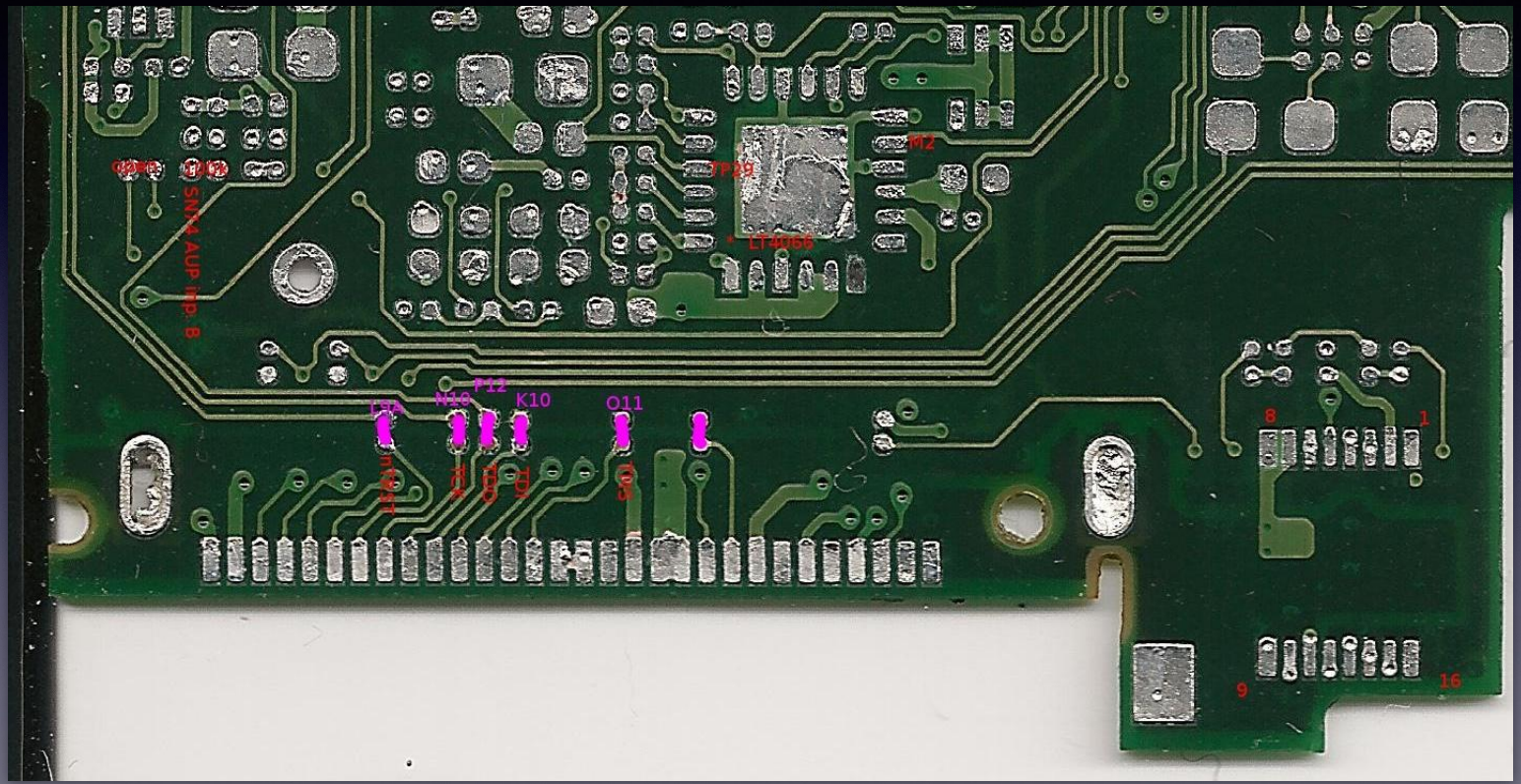
- The very *first* Samsung SoC iPod (along with nano 2) remains unhacked to this day
  - Which makes it the *last* to remain in such status
- The ROM is most likely vulnerable to all of the bugs described before
  - Except Pwnage 2.0 - because there's no X.509 stack at all
- So it's just a matter of dedicating some time

Miscellaneous

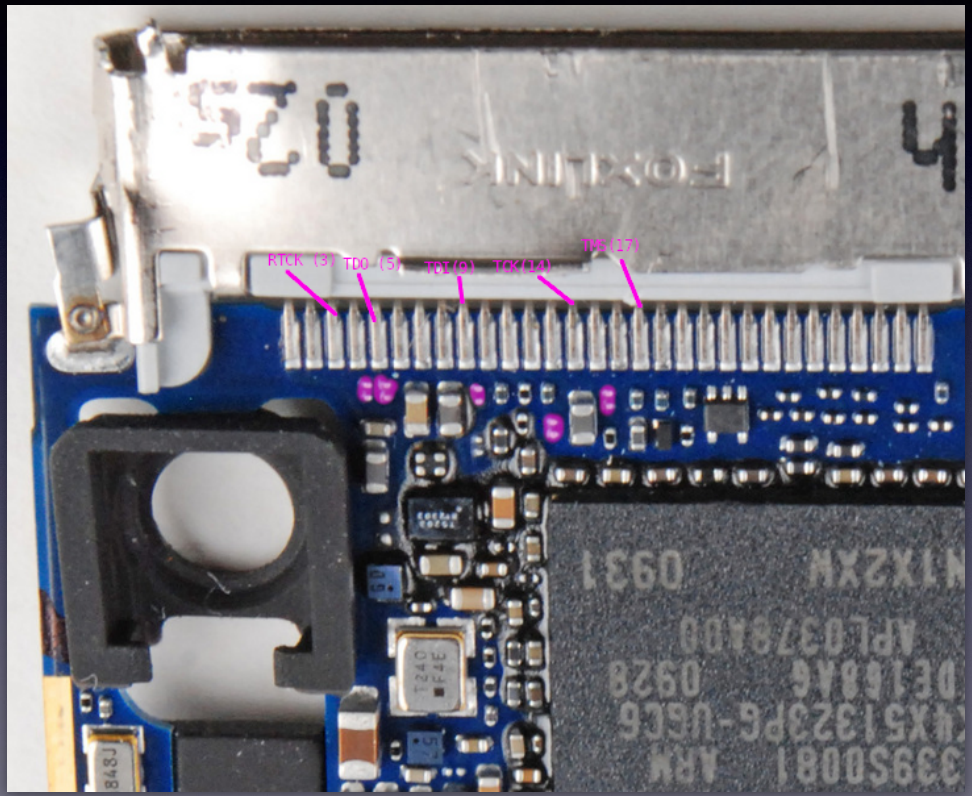
JTAG - expectations

- Good news! S5L87xx SoCs have JTAG!
- Dock connectors (30-pin) miss a few resistors though
  - Nothing some 0201 0-ohm resistors or bits of wire can’t solve

iPod nano 2 JTAG



iPod nano 5 JTAG



JTAG - reality

- CPU core can be controlled, but memory buses seems to get disconnected on JTAG chain scan
- “SecureJTAG” by Samsung: used on some other SoCs, maybe same here?
- Already encountered by Linux4Nano - they leaked code/data fragments by reading core-internal caches

JTAG on Lightning devices

- Lightning is only known to carry SWD (2-wire) and not the normal JTAG (5-wire)
- Production iPod nano 7 is not debuggable via leaked AppleInternal hardware & software
  - Prototype versions of Haywire (Lightning video adapters) are, however
  - Haywire uses seemingly similar SoC to the nano 7's - S5L8747 vs. S5L8740 on the nano

S5L8747 SWD

- iOS-world SoCs can be demoted (since S5L8940, 2011) - long story short, you can unlock SWD if you have a special signature from Apple or code execution in the bootrom context
  - S5L8747 is from 2012, but still lacks this feature
  - I guess it’s safe to assume iPod nano 7 lacks it as well
- However, S5L8747 iBoot has a very weird pattern when reading CHIPID (demotion-related registers are there)

S5L8747 SWD (cont.)

```
1  int chipid_get_chip_id()
2  {
3    while ( (MEMORY[0x3D100000] & 1) == 0 )
4      ;
5    return HIWORD(MEMORY[0x3D100008]);
6  }
```

S5L8747 SWD (cont.)

- Before actually reading it waits for some flag to appear first
- Could it mean the actual CHIPID data is read from somewhere?
  - Could it be hijacked in such case?
- Leaked S5L8700 datasheet (not to be confused with S5L8701) says CHIPID is going to be available for reading after at most 90 cycles after reset
  - S5L8720 (iPod nano 4 & touch 2) iBoot doesn’t seem to wait for anything though

Links

- **Linux4Nano report** (Notes exploit & a lot more useful info) - files.freemyipod.org/reports/Linux4NanoReport.pdf
- **Pwnage 2.0** details - freemyipod.org/wiki/Pwnage_2.0
- **Pwnage 2.0** details - theapplewiki.com/wiki/25C3_presentation_%22Hacking_the_iPhone%22
- **wInd3x** write-up - q3k.org/wInd3x.html
- **wInd3x** implementation - github.com/freemyipod/wInd3x
  - Also **S5Late** now
- **S5Late** for iPod nano 7, WebUSB implementation - nugget.zone

Links (cont.)

- **ipod_sun** - github.com/CUB3D/ipod_sun
- **S5Late** old implementation - github.com/m-gsch/S5Late
- **S5Late** iPod nano 6 & shuffle 4 port - github.com/NyanSatan/S5Late-8723
- **Pwnage 2.0** iPod shuffle 3 implementation - github.com/NyanSatan/S5L8442Pwnage2
- **`libipoddfu.py`** - github.com/jeanthom/ibugger/blob/master/libipoddfu.py

Links (cont.)

- **freemyipod** wiki - freemyipod.org
  - A lot of information on iPod hardware & software
    - A bit outdated in some places though
- **ROM BOOT** dumps - securerom.fun

Questions?