

# Targeting the iOS Kernel

Stefan Esser <[stefan.esser@sektioneins.de](mailto:stefan.esser@sektioneins.de)>

*SyScan Singapore 2011  
April 28th-29th*

# Who am I?

## Stefan Esser

- from Cologne/Germany
- Information Security since 1998
- PHP Core Developer since 2001
- Suhosin / Hardened-PHP 2004
- Month of PHP Bugs 2007 / Month of PHP Security 2010
- ASLR for jailbroken iPhones 2010 / untethered jailbreak for iOS 4.3.1/2
- Head of Research & Development at SektionEins GmbH

# Motivation

- iPhone security heavily relies on kernel level protections
  - code signing / sandboxing
  - NX / ASLR
- public iPhone exploit payloads are very limited in what they can do
- security researchers have relied on the jailbreakers to provide kernel pwnage
- this session is an introduction to finding bugs in the iOS kernel

# Agenda

- Introduction
- How to get the iOS kernelcache
- Analysing the content of the kernelcache
- Trying to get some kernel symbols
- Using the kernelcache to determine attack surface
- Learning how to use the iOS kernel debugger
  
- Exploitation is not covered in this session - contact me to discuss this topic

# Part I

## Introduction

# Finding Vulnerabilities in the iOS Kernel (I)

- For OS X Apple provides
  - the source code for the latest OS X version (XNU)
  - the source code of some OS X kernel extensions
  - symbols for the binary kernel and some extension (in DebugKit)
- For iOS Apple provides neither

# Finding Vulnerabilities in the iOS Kernel (II)

- because iOS is also XNU based the public source is partly useable
- however the kernel of OS X and iOS are very out of sync
- kernel vulnerabilities that are only interesting for iOS are not fixed in OS X
- auditing XNU will reveal a bunch of vulnerabilities already fixed in iOS
- interesting parts like the ASLR are not yet in any public XNU release

# Finding Vulnerabilities in the iOS Kernel (III)

- source code of kernel extensions is less likely to be desync
- however only a small subset of kernel extensions have source code available
- finding vulnerabilities in iOS kernel extension requires binary analysis



# Interesting Kernel Bugs - OS X

## OS X Kernel

- user-land dereference bugs are not exploitable
- privilege escalation to root usually highest goal
- memory corruptions or code exec in kernel nice but usually not required
- kernel exploits only triggerable as root are not interesting

# Interesting Kernel Bugs - iOS

## iOS Kernel

- user-land dereference bugs are partially exploitable
- privilege escalation to root just a starting point
- memory corruptions or code exec in kernel always required
- kernel exploits only triggerable as root are interesting

# Part II

## The iOS Kernelcache

# Getting the iOS Kernelcache (I)

- iOS kernel is stored within a 6mb file
- stored in /System/Library/Caches/com.apple.kernelcaches/kernelcache
- easier to extract from a firmware image

```
$ ls -la iPhone3,1_4.3.2_8H7_Restore/
total 1362456
drwxr-xr-x  11 sesser  staff      374 18 Apr 22:05 .
drwxr-xr-x  24 sesser  staff      816 18 Apr 22:02 ..
-rw-r--r--@  1 sesser  staff 630358016  5 Apr 04:58 038-1025-007.dmg
-rw-r--r--@  1 sesser  staff  25004228  5 Apr 03:47 038-1031-007.dmg
-rw-r--r--@  1 sesser  staff 23709892  5 Apr 04:14 038-1035-007.dmg
-rw-r--r--@  1 sesser  staff   22691  5 Apr 05:02 BuildManifest.plist
drwxr-xr-x   5 sesser  staff    170  5 Apr 03:15 Firmware
-rw-r--r--@  1 sesser  staff   2076  5 Apr 04:58 Restore.plist
-rw-r--r--@  1 sesser  staff 6179844  5 Apr 02:30 kernelcache.release.k48
-rw-r--r--@  1 sesser  staff 6086404  5 Apr 02:30 kernelcache.release.n81
-rw-r--r--@  1 sesser  staff 6204036  5 Apr 02:30 kernelcache.release.n90
```

# Getting the iOS Kernelcache (II)

- kernelcache is a packed and encrypted IMG3 file
- can be decrypted and unpacked with xpwntool
- decryption IV + KEY can only be generated with exploited devices
- but can be found on the internet or inside redsn0w

```
00000000 33 67 6d 49 84 aa 5e 00 70 aa 5e 00 38 a2 5e 00 |3gmI..^.p.^.8.^.|
00000010 6c 6e 72 6b 45 50 59 54 20 00 00 00 04 00 00 00 |lnrkEPYT .....|
00000020 6c 6e 72 6b 00 00 00 00 00 00 00 00 00 00 00 00 |lnrk.....|
00000030 00 00 00 00 41 54 41 44 2c a1 5e 00 16 a1 5e 00 |...ATAD,.^...^.|
00000040 04 59 a3 f2 af f3 29 69 38 f4 2f bb dd 7f 41 ae |.Y....)i8./...A.|
00000050 13 49 fa 56 4a cd bd 46 09 2c 77 6f 03 8c cc eb |.I.VJ..F.,wo...|
00000060 95 29 39 c2 2f 68 4f 18 5a c3 7d 5b 9c 12 8c ac |.)9./h0.Z.}[...|
00000070 8c f9 75 76 db a5 85 70 8d 90 7a ed 7b 94 b2 b3 |..uv...p..z.{...|
00000080 7b dc 95 5f de aa e6 0d 0b ad d6 94 ba dd 7e fe |{.._.....~.|
00000090 a8 aa e9 44 da b2 62 41 3a df dd 5e 24 f3 8a 76 |...D..bA:..^$.v|
000000a0 f2 3b 12 3f ab 7f da 60 d3 db ad 92 5c f3 90 ef |.;.?...`....\...|
```

# Getting the iOS Kernelcache (III)

- decrypting and unpacking reveals an ARMv7 MACH-O binary
- all MACH-O tools will work out of the box with the kernelcache
- this includes IDA but also otool and MachOView

```
00000000  ce fa ed fe 0c 00 00 00 09 00 00 00 02 00 00 00 |.....|
00000010  0b 00 00 00 d8 07 00 00 01 00 00 00 01 00 00 00 |.....|
00000020  d0 01 00 00 5f 5f 54 45 58 54 00 00 00 00 00 00 |....__TEXT.....|
00000030  00 00 00 00 00 10 00 80 00 d0 27 00 00 00 00 00 |.....'.....|
00000040  00 d0 27 00 05 00 00 00 05 00 00 00 06 00 00 00 |..'.....|
00000050  00 00 00 00 5f 5f 74 65 78 74 00 00 00 00 00 00 |....__text.....|
00000060  00 00 00 00 5f 5f 54 45 58 54 00 00 00 00 00 00 |....__TEXT.....|
00000070  00 00 00 00 00 20 00 80 dc 00 21 00 00 10 00 00 |.....!.....|
00000080  0c 00 00 00 00 00 00 00 00 00 00 00 00 04 00 80 |.....|
00000090  00 00 00 00 00 00 00 00 5f 5f 63 73 74 72 69 6e |.....__cstrinl|
000000a0  67 00 00 00 00 00 00 00 5f 5f 54 45 58 54 00 00 |lg.....__TEXT..|
```

# Kernelcache is just a Mach-O Binary

kernelcache.iPhone3,1\_4.3.2\_8H7.decrypted

RAW RVA

▼ Mach-O Image (???)

- Mach Header
  - Load Commands
  - Section (\_\_TEXT,\_\_text)
  - Section (\_\_TEXT,\_\_cstring)
  - Section (\_\_TEXT,\_\_const)
  - Section (\_\_TEXT,\_\_initcode)
  - Section (\_\_TEXT,\_\_constructor)
  - Section (\_\_TEXT,\_\_destructor)
  - Section (\_\_DATA,\_\_data)
  - Section (\_\_DATA,\_\_sysctl\_set)
  - Section (\_\_KLD,\_\_text)
  - Section (\_\_KLD,\_\_constructor)
  - Section (\_\_KLD,\_\_destructor)
  - Section (\_\_KLD,\_\_cstring)
  - Section (\_\_KLD,\_\_data)
  - Section (\_\_PRELINK\_STATE,\_\_kernel)
  - Section (\_\_PRELINK\_STATE,\_\_kexts)
  - Symbol Table
    - String Table
    - Section (\_\_PRELINK\_TEXT,\_\_text)
    - Section (\_\_PRELINK\_INFO,\_\_info)

Offset	Data	Description	Value
00000000	FEEDFACE	Magic Number	MH_MAGIC
00000004	0000000C	CPU Type	???
00000008	00000009	CPU SubType	???
0000000C	00000002	File Type	MH_EXECUTE
00000010	0000000B	Number of Load Commands	11
00000014	000007D8	Size of Load Commands	2008
00000018	00000001	Flags	MH_NOUNDEFS

# Part III

## Analysing the Kernelcache



# iOS Kernelcache vs. IDA

- IDA can load the iOS kernelcache as an ARMv7 Mach-O binary
- however the autoanalysis will fail completely
- large parts not analysed
- code recognized as data and vice versa
- functions not marked as functions
- **IDA clearly needs help**



# Helping IDA - Pointerlists

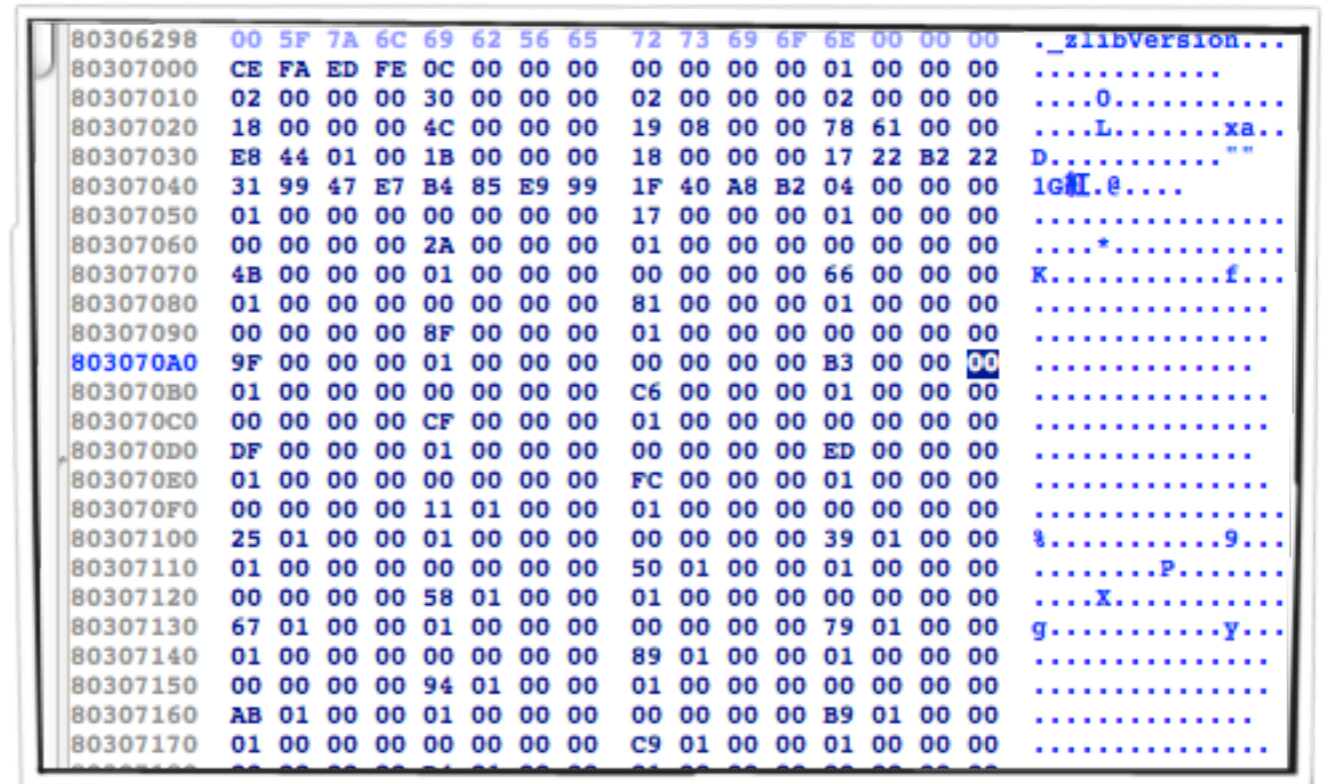
```
__constructor:8037C774 ; Segment type: Pure data
__constructor:8037C774      AREA __constructor, DATA, ALI
__constructor:8037C774      ; ORG 0x8037C774
__constructor:8037C774      DCD loc_80363644+1
__constructor:8037C778      DCD loc_80363B00+1
__constructor:8037C77C      DCD loc_803641BC+1
__constructor:8037C780      DCD loc_80364510+1
__constructor:8037C784      DCD loc_803651D0+1
__constructor:8037C788      DCD loc_80365E80+1
__constructor:8037C78C      DCD loc_80366C50+1
__constructor:8037C790      DCD loc_80367054+1
__constructor:8037C794      DCD loc_80367678+1
__constructor:8037C798      DCD loc_80367F30+1
__constructor:8037C79C      DCD loc_80368B40+1
__constructor:8037C7A0      DCD loc_8036C250+1
__constructor:8037C7A4      DCD loc_8036C90C+1
__constructor:8037C7A8      DCD loc_8036D084+1
__constructor:8037C7AC      DCD loc_8036DF5C+1
__constructor:8037C7B0      DCD loc_8036E328+1
__constructor:8037C7B4      DCD loc_8036E9E8+1
__constructor:8037C7B8      DCD loc_8036FB38+1
__constructor:8037C7BC      DCD loc_80370354+1
__constructor:8037C7C0      DCD loc_80370480+1
__constructor:8037C7C4      DCD loc_80370EA4+1
__constructor:8037C7C4 ; __constructor ends
__constructor:8037C7C4
```

- **pointerlists**

- `__constructor` and `__destructor` contain pointers to code
- `__sysctl_set` is a pointerlist to `sysctl_oid` structs
- second `__data` section contains only pointers
- can be changed with an IDAPython script easily

# Helping IDA - Kernel Extensions

- `__PRELINK_TEXT` seems to contain Mach-O files
- these files are loaded KEXT
- more than 130 of them
- IDA cannot handle this by default
- need a IDAPython script that finds all KEXT and adds their segments

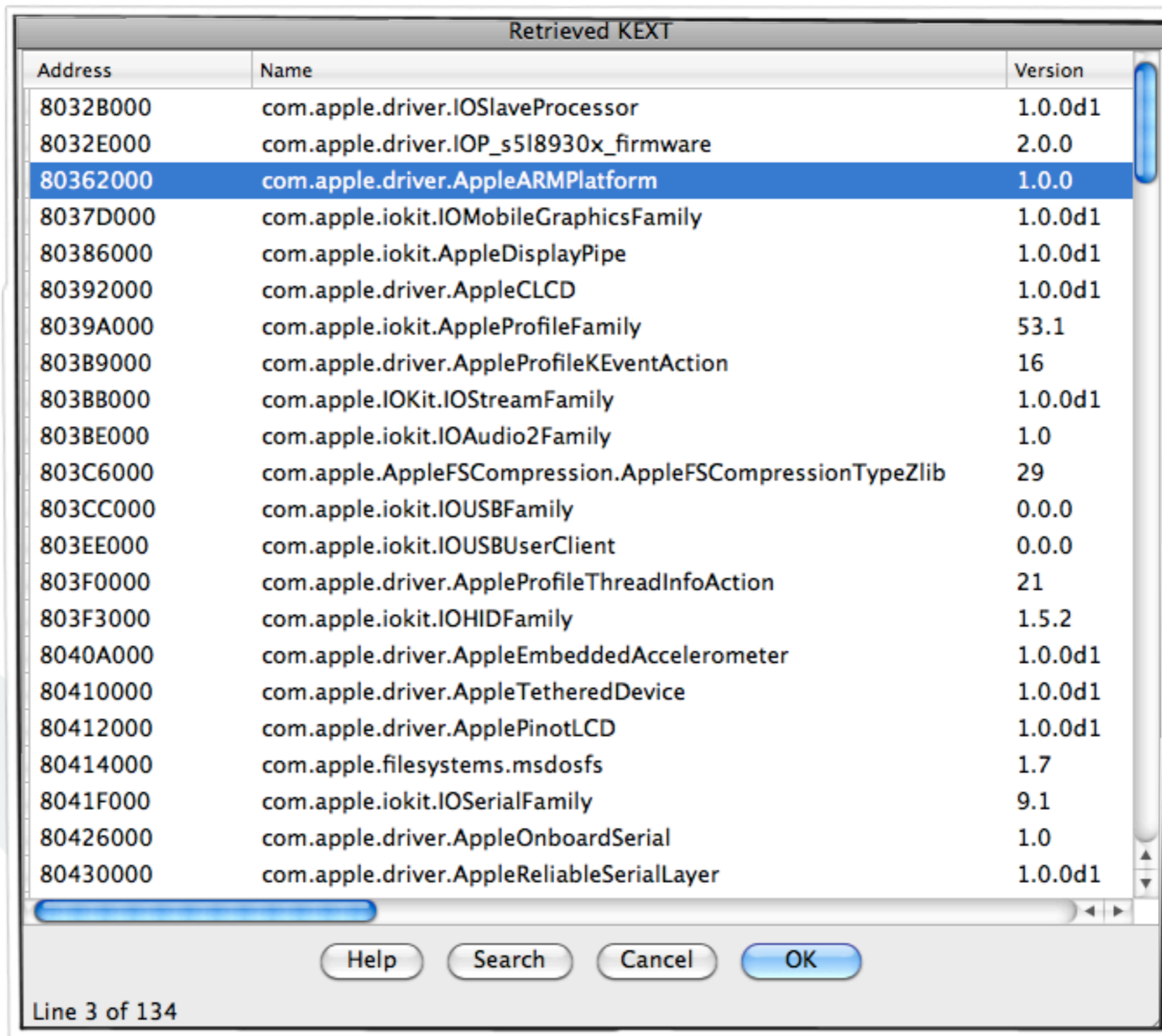


```
80306298 00 5F 7A 6C 69 62 56 65 72 73 69 6F 6E 00 00 00 00 ._zlibVersion...
80307000 CE FA ED FE 0C 00 00 00 00 00 00 00 01 00 00 00 .....
80307010 02 00 00 00 30 00 00 00 02 00 00 00 02 00 00 00 .....0.....
80307020 18 00 00 00 4C 00 00 00 19 08 00 00 78 61 00 00 ....L.....xa..
80307030 E8 44 01 00 1B 00 00 00 18 00 00 00 17 22 B2 22 D....."
80307040 31 99 47 E7 B4 85 E9 99 1F 40 A8 B2 04 00 00 00 00 1gM.e....
80307050 01 00 00 00 00 00 00 00 17 00 00 00 01 00 00 00 .....
80307060 00 00 00 00 2A 00 00 00 01 00 00 00 00 00 00 00 .....*.....
80307070 4B 00 00 00 01 00 00 00 00 00 00 00 66 00 00 00 K.....f...
80307080 01 00 00 00 00 00 00 00 81 00 00 00 01 00 00 00 .....
80307090 00 00 00 00 8F 00 00 00 01 00 00 00 00 00 00 00 .....
803070A0 9F 00 00 00 01 00 00 00 00 00 00 00 B3 00 00 00 .....
803070B0 01 00 00 00 00 00 00 00 C6 00 00 00 01 00 00 00 .....
803070C0 00 00 00 00 CF 00 00 00 01 00 00 00 00 00 00 00 .....
803070D0 DF 00 00 00 01 00 00 00 00 00 00 00 ED 00 00 00 .....
803070E0 01 00 00 00 00 00 00 00 FC 00 00 00 01 00 00 00 .....
803070F0 00 00 00 00 11 01 00 00 01 00 00 00 00 00 00 00 .....
80307100 25 01 00 00 01 00 00 00 00 00 00 00 39 01 00 00 3.....9...
80307110 01 00 00 00 00 00 00 00 50 01 00 00 01 00 00 00 .....P.....
80307120 00 00 00 00 58 01 00 00 01 00 00 00 00 00 00 00 .....X.....
80307130 67 01 00 00 01 00 00 00 00 00 00 00 79 01 00 00 g.....y...
80307140 01 00 00 00 00 00 00 00 89 01 00 00 01 00 00 00 .....
80307150 00 00 00 00 94 01 00 00 01 00 00 00 00 00 00 00 .....
80307160 AB 01 00 00 01 00 00 00 00 00 00 00 B9 01 00 00 .....
80307170 01 00 00 00 00 00 00 00 C9 01 00 00 01 00 00 00 .....
```

# Helping IDA - findAndMarkKEXT.py

- IDAPython script that
  - scans the `__PRELINK_TEXT` segment for Mach-O files
  - adds new segments for each KEXT section
  - marks code segments as THUMB code
  - handles `__destructor` and `__constructor`
  - adds `kmod_info` to sqlite database
  - shows a list of KEXT

# Helping IDA - findAndMarkKEXT.py



Address	Name	Version
8032B000	com.apple.driver.IOSlaveProcessor	1.0.0d1
8032E000	com.apple.driver.IOP_s5l8930x_firmware	2.0.0
80362000	com.apple.driver.AppleARMPlatform	1.0.0
8037D000	com.apple.iokit.IOMobileGraphicsFamily	1.0.0d1
80386000	com.apple.iokit.AppleDisplayPipe	1.0.0d1
80392000	com.apple.driver.AppleCLCD	1.0.0d1
8039A000	com.apple.iokit.AppleProfileFamily	53.1
803B9000	com.apple.driver.AppleProfileKeyEventAction	16
803BB000	com.apple.IOKit.IOStreamFamily	1.0.0d1
803BE000	com.apple.iokit.IOAudio2Family	1.0
803C6000	com.apple.AppleFSCompression.AppleFSCompressionTypeZlib	29
803CC000	com.apple.iokit.IOUSBFamily	0.0.0
803EE000	com.apple.iokit.IOUSBUserClient	0.0.0
803F0000	com.apple.driver.AppleProfileThreadInfoAction	21
803F3000	com.apple.iokit.IOHIDFamily	1.5.2
8040A000	com.apple.driver.AppleEmbeddedAccelerometer	1.0.0d1
80410000	com.apple.driver.AppleTetheredDevice	1.0.0d1
80412000	com.apple.driver.ApplePinotLCD	1.0.0d1
80414000	com.apple.filesystems.msdosfs	1.7
8041F000	com.apple.iokit.IOSerialFamily	9.1
80426000	com.apple.driver.AppleOnboardSerial	1.0
80430000	com.apple.driver.AppleReliableSerialLayer	1.0.0d1

Line 3 of 134

# Functions and Code

- after performing previous fixups IDA is already a lot better



- however a lot of functions are not recognized
- script that scans for code outside of functions and creates functions
- many cases still require manual work

# IOKit Driver Classes (I)

- IOKit drivers are implemented in a subset of C++
- classes and their method tables can be found in kernelcache
- main kernel IOKit classes even come with symbols

```
8026A2A8 ; `vtable for'IOService
8026A2A8 __ZTV9IOService DCB 0 ; DATA XREF: IOResources::getWorkLoop(void)+C|o
8026A2A8 ; __text:off_801D1AE0|o ...
8026A2A9 DCB 0
8026A2AA DCB 0
8026A2AB DCB 0
8026A2AC DCB 0
8026A2AD DCB 0
8026A2AE DCB 0
8026A2AF DCB 0
8026A2B0 off_8026A2B0 DCD sub_801D6F10+1 ; DATA XREF: IOService::IOService(void)+E|o
8026A2B0 ; __text:off_801D6A14|o ...
8026A2B4 DCD __ZN9IOServiceD0Ev+1
8026A2B8 DCD __ZNK8OSObject7releaseEi+1
8026A2BC DCD __ZNK8OSObject14getRetainCountEv+1
8026A2C0 DCD __ZNK8OSObject6retainEv+1
8026A2C4 DCD __ZNK8OSObject7releaseEv+1
8026A2C8 DCD __ZNK8OSObject9serializeEP11OSSerialize+1
8026A2CC DCD __ZNK9IOService12getMetaClassEv+1
8026A2D0 DCD __ZNK15OSMetaClassBase9isEqualToEPKS_+1
8026A2D4 DCD __ZNK8OSObject12taggedRetainEPKv+1
8026A2D8 DCD __ZNK8OSObject13taggedReleaseEPKv+1
8026A2DC DCD __ZNK8OSObject13taggedReleaseEPKvi+1
8026A2E0 DCD __ZN15OSMetaClassBase25_RESERVEDOSMetaClassBase3Ev+1
8026A2E4 DCD __ZN15OSMetaClassBase25_RESERVEDOSMetaClassBase4Ev+1
8026A2E8 DCD __ZN15OSMetaClassBase25_RESERVEDOSMetaClassBase5Ev+1
8026A2EC DCD __ZN15OSMetaClassBase25_RESERVEDOSMetaClassBase6Ev+1
8026A2F0 DCD __ZN15OSMetaClassBase25_RESERVEDOSMetaClassBase7Ev+1
8026A2F4 DCD __ZN8OSObject4initEv+1
8026A2F8 off_8026A2F8 DCD __ZN9IOService4freeEv+1
8026A2F8 ; DATA XREF: IOMapper::free(void)+18|o
8026A2F8 ; IOUserClient::free(void)+1E|o ...
8026A2FC DCD __ZNK15IORegistryEntry12copyPropertyEPKcPK15IORegistryPlanem+1
```

# IOKit Driver Classes (II) - MetaClass

- most iOS IOKit classes come without symbols
- however IOKit defines for almost all classes a so called MetaClass
- MetaClass contains runtime information about the original object
- constructors of MetaClass'es leak name and parent objects

```
801D5A00 ; IOService::MetaClass::MetaClass(void)
801D5A00         EXPORT __ZN9IOService9MetaClassC1Ev
801D5A00 __ZN9IOService9MetaClassC1Ev         ; CODE XREF: sub_801D5A28+1E|p
801D5A00         PUSH        {R4,R7,LR}
801D5A02         ADD         R7, SP, #4
801D5A04         MOVS        R3, #0x50 ; 'P'
801D5A06         LDR         R1, =aIoservice ; "IOService"
801D5A08         LDR         R2, =__ZN15IORegistryEntry10gMetaClassE ; IOR
801D5A0A         LDR.W       R12, =( __ZN11OSMetaClassC2EPKcPKS_j+1)
801D5A0E         MOV         R4, R0
801D5A10         BLX        R12 ; OSMetaClass::OSMetaClass(char const*,O
801D5A12         LDR         R3, =off_8026A25C
801D5A14         STR         R3, [R4]
801D5A16         POP         {R4,R7,PC}
801D5A16 ; End of function IOService::MetaClass::MetaClass(void)
801D5A16
```

R1 = Object Name  
R2 = Parent's MetaClass  
R3 = Methods of MetaClass

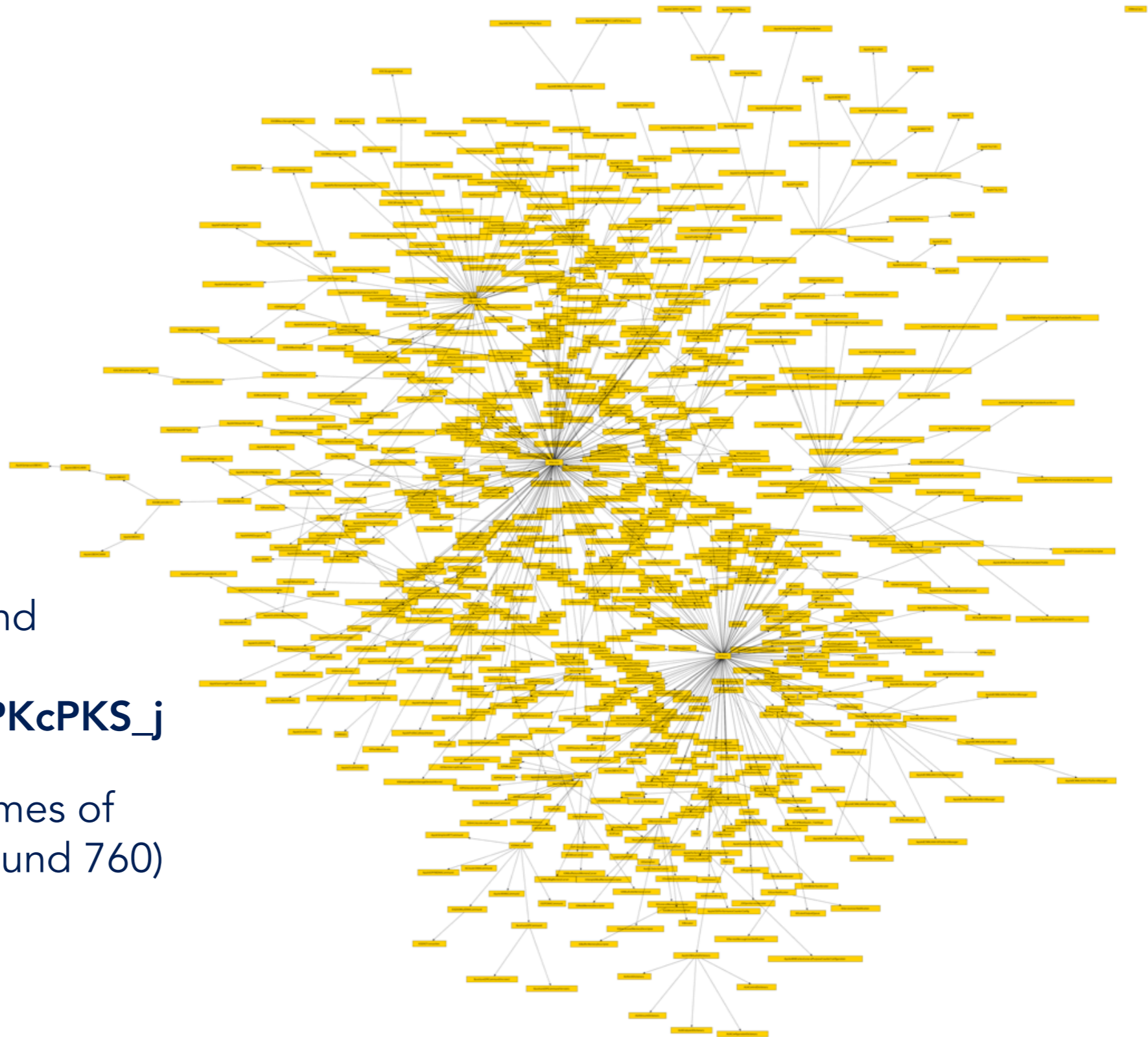


# IOKit Object Hierarchy - Full View

all MetaClasses can be found  
through xrefs of  
**\_\_ZN11OSMetaClassC2EPKcPKS\_j**

allows to determine the names of  
almost all IOKit classes (around 760)

and allows to build the  
IOKit object hierarchy tree





# Part IV

iOS Kernel Where Are your Symbols?

# iOS Kernel Symbols ???

- iOS kernel contains around 4000 symbols
- but more than 30000 functions and many more variables
- Apple won't help us (at least willingly)
- need to combine several methods to get more symbols

# Kernel Symbols - Manual Symbolization

The screenshot shows the IDA Pro interface with a 'Rename address' dialog box open. The dialog box has the following fields and options:

- Address: 0x802534EC
- Name: `_dispatch_table`
- Maximum length of new names: 15
- Local name prefix: @@
- Options:
  - Local name
  - Include in names list
  - Public name
  - Autogenerated name
  - Weak name
  - Create name anyway
- Buttons: Help, Cancel, OK

The background shows the IDA Pro main window with the following assembly code visible:

```
005DC4FE db 0
005DC4FF db 0
005DC500 _dispatch_table dd offset _kdp_connect ; DATA XREF: _kdp_packet+DA|r
005DC504 dd offset _kdp_disconnect
005DC508 dd offset _kdp_hostinfo
005DC50C dd offset _kdp_version
005DC510 dd offset _kdp_maxbytes
005DC514 dd offset _kdp_readmem
00A9F514 005DC514: __const:005DC514
```

At the bottom of the screen, a system status bar shows: Python, AU: idle, Up, Disk: 51GB.

**Manual Symbolization will only take forever...**

# Little Helpers

- porting all symbols manually will take forever
- we can automate porting common structs
  - pointer list
  - arrays of structs
- special helper for porting `sysctl_set`

# Zynamic's BinDiff

- Zynamic's BinDiff is a great tool
  - not only to find differences in binaries
  - but also to port symbols
  - even cross platform
- Using BinDiff to diff OS X kernel against iOS 4.3.2
  - works but initially gives bad results
  - other ways to add symbols are required
  - BinDiff can then be repeated

# Zynamic's BinDiff - Demo (I)

IDA - idb\_before\_bindiff.idb (kernelcache.iPod4,1\_4.3.2\_8H7.decrypted)

Remote GDB debugger

Function r	similarity	confide	EA primary	name primary	EA secondary	name secondary
sub_	0.90	0.97	8004B494	_vm_object_reaper_init	0026CA55	_vm_object_reaper_init
sub_	0.90	0.97	801A8878	_ux_handler_init	004D1817	_ux_handler_init
sub_	0.90	0.97	801D317C	sub_801D317C_7246	005354EE	__ZN9IOService23syncNotificationHandle
sub_	0.90	0.97	801BEDDC	OSSymbolPool::operator new(ulong)	0050FF8B	__ZN12OSSymbolPoolnwEm
sub_	0.90	0.97	801B92AC	sub_801B92AC_7034	00507DF8	__ZN6OSKext24removeKextWithIdentifier
sub_	0.89	0.97	8027850C	sub_8027850C_8592	0061B09C	_ipc_init
sub_	0.89	0.96	80279194	sub_80279194_8609	0061C02A	_task_init
sub_	0.89	0.96	80151070	sub_80151070_5221	0043A1F6	_hfs_removefile_callback
sub_	0.89	0.96	8006CA04	sub_8006CA04_2403	002A05BE	_ml_get_max_cpus
sub_	0.89	0.96	801A81F0	sub_801A81F0_6760	004D1386	_macx_backing_store_suspend
sub_	0.89	0.96	801ECFC4	IONMultiMemoryDescriptor::MetaClass::alloc(void)	005582F6	__ZNK23IONMultiMemoryDescriptor9Meta
sub_	0.89	0.96	80161034	sub_80161034_5484	00462FC6	_aio_decrement_total_count
sub_	0.89	0.96	801747C8	_is_suser	00480295	_is_suser
sub_	0.89	0.96	801E8B08	IONaturalMemoryCursor::MetaClass::alloc(void)	005527D0	__ZNK21IONaturalMemoryCursor9MetaC
sub_	0.89	0.96	801BD69C	OSSerialize::MetaClass::alloc(void)	0050DF66	__ZNK11OSSerialize9MetaClass5allocEv
sub_	0.89	0.96	801E1D34	IOPMPowerSourceList::MetaClass::alloc(void)	00547996	__ZNK19IOPMPowerSourceList9MetaClas
sub_	0.89	0.96	801D35A8	IOService::MetaClass::alloc(void)	00524050	__ZNK10IOService9MetaClass5allocEv

Line 3332 Line 6 of 10889

Output window

```
Sending result to BinDiff GUI...
Sending result to BinDiff GUI...
Sending result to BinDiff GUI...
2 comments ported for function 801a8878
2 comments ported for function 8004b494
Sending result to BinDiff GUI...
Sending result to BinDiff GUI...
```

Python

AU: idle Up Disk: 51GB



# Zynamic's BinDiff - Demo (II)

BinDiff 3.2.1

Project Help

Search   Regular Expression  Case sensitive

Flowgraph Assembler

sub\_8006CA04\_2403

\_ml\_get\_max\_cpus

### primary

```
8006ca04
ca04 PUSH      {R4, R7, LR}
ca06 ADD       R7, SP, 4
ca08 MOVS    R0, 0
ca0a BLX     b2 _ml_set_interrupts_enabled
ca0e MOV     R4, R0
ca10 LDR    R0, [off_8006CA34 ]
ca12 LDR    R3, [R0]
ca14 CMP   R3, 1
ca16 BEQ   b2 loc_8006CA28
```

```
8006ca18
ca18 MOVS    R3, 2
ca1a MOVS    R1, 0
ca1c STR    R3, [R0]
ca1e BL     b2 _assert_wait
ca22 MOVS    R0, 0
ca24 BL     b2 _thread_block
```

```
8006ca28
ca28 MOV     R0, R4
ca2a BLX     b2 _ml_set_interrupts_enabled
ca2e LDR    R3, [off_8006CA38 ]
ca30 LDR    R0, [R3, 8]
ca32 POP   {R4, R7, PC}
```

### secondary

```
002a05be
05be push     ebp
05bf mov     ebp, esp
05c1 push   ebx
05c2 sub    esp, 0x14
05c5 mov    ss:[esp], 0
05cc call  _ml_set_interrupts_enabled
05d1 mov    ebx, eax
05d3 cmp   ds:[_max_cpus_initialized ], 1
05da jz    loc_2A0606
```

```
002a05dc
05dc mov    ds:[_max_cpus_initialized ], 2
05e6 mov    ss:[esp+4], 0
05ee mov    ss:[esp], _max_cpus_initialized
05f5 call  _assert_wait
05fa mov    ss:[esp], 0
0601 call  _thread_block
```

```
002a0606
0606 mov    ss:[esp], ebx
0609 call  _ml_set_interrupts_enabled
060e mov    eax, ds:[dword_847D68 ]
0613 add   esp, 0x14
0616 pop   ebx
0617 leave
0618 retn
```

# Using IOKit Class Hierarchy for Symbols

- most IOKit classes are without symbols
- however they are derived from base IOKit classes with symbols
- we can create symbols for overloaded methods

## Some Methods from AppleBasebandUserClient

```
___const:8043A270      DCD   ___ZN9IOService12tellChangeUpEm+1
___const:8043A274      DCD   ___ZN9IOService16allowPowerChangeEm+1
___const:8043A278      DCD   ___ZN9IOService17cancelPowerChangeEm+1
___const:8043A27C      DCD   ___ZN9IOService15powerChangeDoneEm+1
___const:8043A280      DCD   loc_80437D80+1
___const:8043A284      DCD   ___ZN12IOUserClient24registerNotificationPortEP8ipc_portmy+1
___const:8043A288      DCD   ___ZN12IOUserClient12initWithTaskEP4taskPvmP12OSDictionary+1
___const:8043A28C      DCD   ___ZN12IOUserClient12initWithTaskEP4taskPvm+1
___const:8043A290      DCD   sub_80437D5C+1
___const:8043A294      DCD   ___ZN12IOUserClient10clientDiedEv+1
___const:8043A298      DCD   ___ZN12IOUserClient10getServiceEv+1
___const:8043A29C      DCD   ___ZN12IOUserClient24registerNotificationPortEP8ipc_portmm+1
___const:8043A2A0      DCD   ___ZN12IOUserClient24getNotificationSemaphoreEmPP9semaphore+1
```

# Using IOKit Class Hierarchy for Symbols

- n
- h
- v

## Same Methods from IOUserClient

```
___ const:80270100 DCD ___ ZN9IOService12tellChangeUpEm+1
___ const:80270104 DCD ___ ZN9IOService16allowPowerChangeEm+1
___ const:80270108 DCD ___ ZN9IOService17cancelPowerChangeEm+1
___ const:8027010C DCD ___ ZN9IOService15powerChangeDoneEm+1
___ const:80270110 DCD ___ ZN12IOUserClient14externalMethodEjP25IOExternalMet...
___ const:80270114 DCD ___ ZN12IOUserClient24registerNotificationPortEP8ipc_portmy+1
___ const:80270118 DCD ___ ZN12IOUserClient12initWithTaskEP4taskPvmP12OSDictionary+1
___ const:8027011C DCD ___ ZN12IOUserClient12initWithTaskEP4taskPvm+1
___ const:80270120 DCD ___ ZN12IOUserClient11clientCloseEv+1
___ const:80270124 DCD ___ ZN12IOUserClient10clientDiedEv+1
___ const:80270128 DCD ___ ZN12IOUserClient10getServiceEv+1
___ const:8027012C DCD ___ ZN12IOUserClient24registerNotificationPortEP8ipc_portmm+1
___ const:80270130 DCD ___ ZN12IOUserClient24getNotificationSemaphoreEmPP9semaphore+1
```

## Some I

```
___ const:8043A270 DCD ___ ZN9IOService17cancelPowerChangeEm+1
___ const:8043A27C DCD ___ ZN9IOService15powerChangeDoneEm+1
___ const:8043A280 DCD loc_80437D80+1
___ const:8043A284 DCD ___ ZN12IOUserClient24registerNotificationPortEP8ipc_portmy+1
___ const:8043A288 DCD ___ ZN12IOUserClient12initWithTaskEP4taskPvmP12OSDictionary+1
___ const:8043A28C DCD ___ ZN12IOUserClient12initWithTaskEP4taskPvm+1
___ const:8043A290 DCD sub_80437D5C+1
___ const:8043A294 DCD ___ ZN12IOUserClient10clientDiedEv+1
___ const:8043A298 DCD ___ ZN12IOUserClient10getServiceEv+1
___ const:8043A29C DCD ___ ZN12IOUserClient24registerNotificationPortEP8ipc_portmm+1
___ const:8043A2A0 DCD ___ ZN12IOUserClient24getNotificationSemaphoreEmPP9semaphore+1
```

# Using IOKit Class Hierarchy for Symbols

➔ borrowing from the parent class we get

- AppleBasebandUserClient::externalMethod(unsigned int, IOExternalMethodArguments \*, IOExternalMethodDispatch \*, OSObject \*, void \*)
- AppleBasebandUserClient::clientClose(void)

## Symbolized Methods from AppleBasebandUserClient

```
___const:8043A270      DCD   ___ZN9IOService12tellChangeUpEm+1
___const:8043A274      DCD   ___ZN9IOService16allowPowerChangeEm+1
___const:8043A278      DCD   ___ZN9IOService17cancelPowerChangeEm+1
___const:8043A27C      DCD   ___ZN9IOService15powerChangeDoneEm+1
___const:8043A280      DCD   ___ZN23AppleBasebandUserClient14externalMethodEjP25IOExtern...
___const:8043A284      DCD   ___ZN12IOUserClient24registerNotificationPortEP8ipc_portmy+1
___const:8043A288      DCD   ___ZN12IOUserClient12initWithTaskEP4taskPvmP12OSDictionary+1
___const:8043A28C      DCD   ___ZN12IOUserClient12initWithTaskEP4taskPvm+1
___const:8043A290      DCD   ___ZN23AppleBasebandUserClient11clientCloseEv+1
___const:8043A294      DCD   ___ZN12IOUserClient10clientDiedEv+1
___const:8043A298      DCD   ___ZN12IOUserClient10getServiceEv+1
___const:8043A29C      DCD   ___ZN12IOUserClient24registerNotificationPortEP8ipc_portmm+1
___const:8043A2A0      DCD   ___ZN12IOUserClient24getNotificationSemaphoreEmPP9semaphore+1
```

# Exporting Symbols

- IDA cannot export symbols back into Mach-O files
- no easy way to use symbols with GDB
- little helper IDAPython symbol exporter was developed

# Part V

## iOS Kernel Attack Surface

# iOS Kernel Attack Surface

- **simple rule** you can only attack the kernel where it interfaces with
  - user space code
  - the network
  - the hardware
  - the filesystem

# Attacking from User Space - Syscalls

- syscalls are directly callable from user space
- for all OS X syscalls source code is available
- however iOS has 8 additional syscalls
- after syscall table is found syscall handlers can be audited



# Finding and Marking the Syscall Table

```
data:802926E8 sysent <1, 0, 0, unk_80182DFD, 0, 0, 1, 0xC, 0>; 417
data:802926E8 sysent <3, 0, 0, unk_80182DFD, 0, 0, 1, 0xC, 0>; 417
data:802926E8 sysent <0, 0, 0, unk_80182465, 0, 0, 1, 0, 0>; 418
data:802926E8 sysent <0, 0, 0, unk_80182465, 0, 0, 1, 0, 0>; 419
data:802926E8 sysent <1, 0, 0, unk_8019FFC9, 0, 0, 1, 4, 0>; 420
data:802926E8 sysent <3, 0, 0, unk_801621A1, 0, 0, 1, 0xC, 0>; 421
data:802926E8 sysent <2, 0, 0, unk_80178445, 0, 0, 1, 8, 0>; 422
data:802926E8 sysent <7, 0, 0, unk_80178889, 0, 0, 1, 0x1C, 0>; 423
data:802926E8 sysent <5, 0, 0, unk_80093C49, 0, 0, 1, 0x14, 0>; 424
data:802926E8 sysent <2, 0, 0, unk_802067C1, 0, 0, 1, 8, 0>; 425
data:802926E8 sysent <5, 0, 0, unk_800933DD, 0, 0, 1, 0x14, 0>; 426
data:802926E8 sysent <5, 0, 0, unk_8008FA2D, 0, 0, 6, 0x14, 0>; 427
data:802926E8 sysent <0, 0, 0, unk_8015E139, 0, 0, 2, 0, 0>; 428
data:802926E8 sysent <1, 0, 0, unk_8015E13D, 0, 0, 1, 4, 0>; 429
data:802926E8 sysent <2, 0, 0, unk_80166D25, 0, 0, 1, 8, 0>; 430
data:802926E8 sysent <1, 0, 0, unk_801673CD, 0, 0, 1, 4, 0>; 431
data:802926E8 sysent <2, 0, 0, unk_8015E141, 0, 0, 1, 8, 0>; 432
data:802926E8 sysent <1, 0, 0, unk_801A7C71, 0, 0, 1, 4, 0>; 433
data:802926E8 sysent <1, 0, 0, unk_801A7C01, 0, 0, 1, 4, 0>; 434
data:802926E8 sysent <1, 0, 0, unk_801A7BB9, 0, 0, 1, 4, 0>; 435
data:802926E8 sysent <2, 0, 0, unk_801A7B19, 0, 0, 1, 8, 0>; 436
data:802926E8 sysent <5, 0, 0, unk_801A73C5, 0, 0, 1, 0x14, 0>; 437
data:80294FF8 _nsysent DCD 0x1B6 ; 0
data:80294FFC EXPORT _kdebug_enable
data:80294FFC ; unsigned int kdebug_enable
data:80294FFC _kdebug_enable DCD 0 ; DATA XREF: __text:80015E20|r
data:80294FFC ; __text:8001FB02|r
data:80295000 DCB 0
```

- Apple removed symbols ***\_sysent*** and ***\_nsysent***
- however the syscall table is still easy to find
  - ➔  $\_nsysent = \_kdebug\_enable - 4$
  - ➔  $\_sysent = \_nsysent - (*\_nsysent * 36)$

# Attacking from User Space - Mach-Traps

- Mach-traps are the “syscalls” of the mach subsystem
- harder to find because no symbols nearby
- best solution is to search for string references
- interesting string is “kern\_invalid mach trap”
- function “kern\_invalid” will be repeatedly referenced from mach trap handler table

# Attacking through Network Protocols

- network protocols are added by **net\_add\_proto()**
- script scanning for xrefs can find all defined network protocols
- dumping content of **protosw** and **domain** structures
- interesting for vulnerability research are
  - setsockopt handler
  - network packet parser

# Attacking through Network Protocols (II)

```
main kernel
-----
net_add_proto() call at 800eb3c6
type: 0      - protocol: 00000000 - domain: internet

type: DGRAM  - protocol: 00000011 - domain: internet
-> setsockopt handler at 800f8e95
-> packet parser at 800f9001

type: STREAM - protocol: 00000006 - domain: internet
-> setsockopt handler at 800f7a95
-> packet parser at 800ef249

type: RAW    - protocol: 000000ff - domain: internet
-> setsockopt handler at 800edfc1
-> packet parser at 800ee28d

type: RAW    - protocol: 00000001 - domain: internet
-> setsockopt handler at 800edfc1
-> packet parser at 800e8fa5
```

# Attacking through Network Protocols (III)

```
net_add_proto() call at 8027ce2c
type: STREAM - protocol: 00000000 - domain: unix
-> setsockopt handler at 8019e7b5
```

```
type: DGRAM - protocol: 00000000 - domain: unix
-> setsockopt handler at 8019e7b5
```

```
com.apple.nke.ppp
```

```
-----
```

```
net_add_proto() call at 808179ca
type: RAW - protocol: 00000001 - domain: PPP
```

```
com.apple.nke.pptp
```

```
-----
```

```
net_add_proto() call to complex for this script at 80a84774
---
```

```
com.apple.nke.lhttp
```

```
-----
```

```
net_add_proto() call to complex for this script at 8081f714
```

# Attacking through Devices

- character and block devices added by the functions
  - `cdevsw_add()`
  - `cdevsw_add_with_bdev()`
  - `bdevsw_add()`
- script scanning for xrefs can find all defined devices
- interesting for vulnerability research are the ioctl handlers

# Attacking through Devices (II)

```
com.apple.driver.AppleOnboardSerial
```

```
-----  
_cdevsw_add() call at 8042842a  
-> ioctl handler at 804282e1
```

```
com.apple.driver.AppleReliableSerialLayer
```

```
-----  
_cdevsw_add() call at 8043373e  
-> ioctl handler at 80432525
```

```
com.apple.iokit.IO80211Family
```

```
-----  
_cdevsw_add() call at 8057252c  
-> ioctl handler at 80571ab9
```

```
com.apple.driver.AppleSerialMultiplexer
```

```
-----  
_cdevsw_add() call at 80456e26  
-> ioctl handler at 80455d2d
```

```
_cdevsw_add() call at 8045cbd4  
-> ioctl handler at 8018243d
```

```
com.company.driver.modulename
```

```
-----  
_cdevsw_add() call at 80490a08  
-> ioctl handler at 8049184d
```

```
_cdevsw_add() call at 8049118c  
-> ioctl handler at 8049184d
```

```
_bdevsw_add() call at 804909ee  
-> ioctl handler at 80492201
```

```
_bdevsw_add() call at 80491172  
-> ioctl handler at 80492201
```

```
com.apple.iokit.IOCryptoAcceleratorFamily
```

```
-----  
_cdevsw_add() call at 805410d0  
-> ioctl handler at 80540529
```

```
_cdevsw_add() call at 80542014  
-> ioctl handler at 805419a9
```

# Attacking from User-Land: Sysctl

- sysctl is interface that gives user-land access to kernel variables
- sysctl variables get added by the functions
  - **sysctl\_register\_oid()**
  - **sysctl\_register\_set() / sysctl\_register\_all()**
- script scanning for xrefs can find all defined sysctl variables
- interesting for vulnerability research are
  - sysctl handlers
  - writeable variables



# Dumping List of Sysctl Handlers

```
main kernel
```

```
-----
```

```
sysctl handler at 8017a805 (sub_8017A804)
sysctl handler at 8017c015 (_sysctl_handle_quad)
sysctl handler at 8017ae21 (sub_8017AE20)
sysctl handler at 80089625 (sub_80089624)
sysctl handler at 8017b2b1 (sub_8017B2B0)
sysctl handler at 8019ce29 (sub_8019CE28)
sysctl handler at 8017c231 (sub_8017C230)
sysctl handler at 8017e23d (sub_8017E23C)
sysctl handler at 8017a1b5 (sub_8017A1B4)
sysctl handler at 8017a441 (sub_8017A440)
sysctl handler at 800f4445 (sub_800F4444)
sysctl handler at 8011cc49 (sub_8011CC48)
sysctl handler at 8017a84d (sub_8017A84C)
sysctl handler at 8008c051 (sub_8008C050)
sysctl handler at 8017e1b9 (sub_8017E1B8)
```

```
...
```

```
com.apple.iokit.AppleProfileFamily
```

```
-----
```

```
sysctl handler at 8039ef51 (sub_8039EF50)
```

```
com.apple.driver.AppleD1815PMU
```

```
-----
```

```
sysctl handler at 807b513d
```

```
com.apple.iokit.IOUSBFamily
```

```
-----
```

```
sysctl handler at 803cd165 (sub_803CD164)
```

```
com.apple.iokit.IOUSBMassStorageClass
```

```
-----
```

```
sysctl handler at 808dd019
```

```
com.apple.driver.AppleARMPlatform
```

```
-----
```

```
sysctl handler at 8036ecf1 (sub_8036ECF0)
```

```
com.apple.iokit.IO SCSIArchitectureModelFamily
```

```
-----
```

```
sysctl handler at 80794cd1 (sub_80794CD0)
```

# Dumping Writable Sysctl Variables

```
com.apple.iokit.IOSCSIArchitectureModelFamily
```

```
-----  
sysctl_register_oid() call at 80794e1c - struct at 80796a88  
-> sysctl name:      debug.SCSIArchitectureModel  
-> sysctl handler: 80794cd1 (sub_80794CD0)
```

```
sysctl_register_oid() call at 80794ef0 - struct at 80796a88  
-> sysctl name:      debug.SCSIArchitectureModel  
-> sysctl handler: 80794cd1 (sub_80794CD0)
```

```
com.apple.driver.AppleProfileThreadInfoAction
```

```
-----  
sysctl_register_oid() call at 803f1c6e - struct at 803f2700  
-> sysctl name:      appleprofile.actions.threadinfo.default_continuous_buffer_size  
-> sysctl handler: 8017bfb9 (_sysctl_handle_int)  
-> var address:      803f2760 00000000
```

```
sysctl_register_oid() call at 803f1c72 - struct at 803f2730  
-> sysctl name:      appleprofile.actions.threadinfo.max_memory  
-> sysctl handler: 8017bfb9 (_sysctl_handle_int)  
-> var address:      803f281c 00000000
```

```
com.apple.security.sandbox
```

```
-----  
sysctl_register_oid() call at 8093647a - struct at 8093b57c  
-> sysctl name:      security.mac.sandbox.debug_mode  
-> sysctl handler: 8017bfb9 (_sysctl_handle_int)  
-> var address:      8093b548 00000000
```

# Attacking from User-Land: IOKit Drivers

- IOKit drivers can also talk with user-space through their objects
- all classes derived from IOUserClient can communicate with kernel
- script can list all classes derived from IOUserClient
- e.g. user-space baseband method calls will go through this method
  - `AppleBasebandUserClient::externalMethod(unsigned int, IOExternalMethodArguments *, IOExternalMethodDispatch *, OSObject *, void *)`

# Part VI

## iOS Kernel Debugging

# iOS Kernel Debugging

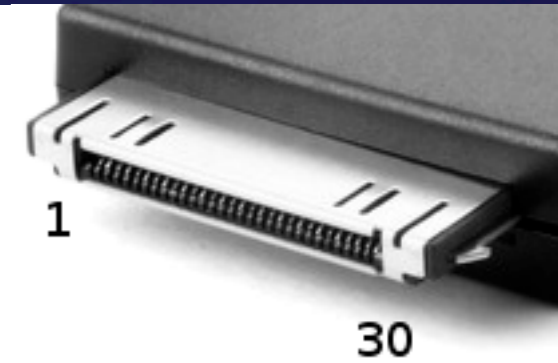
- no support for kernel level debugging by iOS SDK
- developers are not supposed to do kernel work anyway
- strings inside kernelcache indicate the presence of debugging code
- boot arg "debug" is used
- and code of KDP seems there

# KDP on iOS 4

- the OS X kernel debugger KDP is obviously inside the iOS kernel
- but KDP does only work via ethernet or serial interface
- how to communicate with KDP?
- the iPhone / iPad do not have ethernet or serial, do they?

# iPhone Dock Connector (Pin-Out)

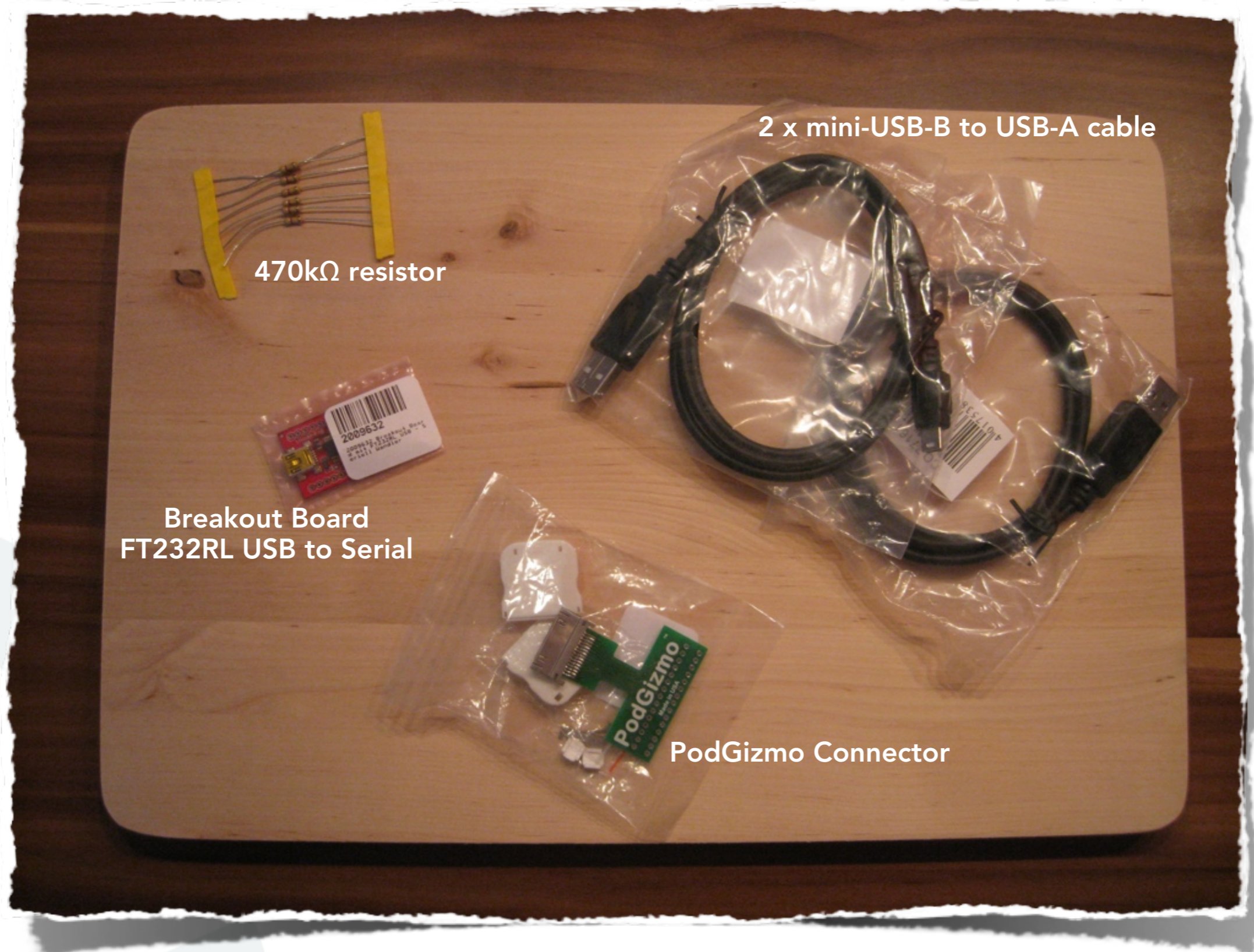
PIN	Desc
1,2	GND
3	Line Out - R+
4	Line Out - L+
5	Line In - R+
6	Line In - L+
8	Video Out
9	S-Video CHR Output
10	S-Video LUM Output
11	GND
12	Serial TxD
13	Serial RxD
14	NC
15,16	GND
17	NC
18	3.3V Power
19,20	12V Firewire Power
21	Accessory Indicator/Serial Enable
22	FireWire Data TPA-
23	USB Power 5 VDC
24	FireWire Data TPA+
25	USB Data -
26	FireWire Data TPB-
27	USB Data +
28	FireWire Data TPB+
29,30	GND



iPhone Dock Connector has PINs for

- Line Out / In
- Video Out
- USB
- FireWire
- **Serial**

# USB Serial to iPhone Dock Connector



470kΩ resistor

2 x mini-USB-B to USB-A cable

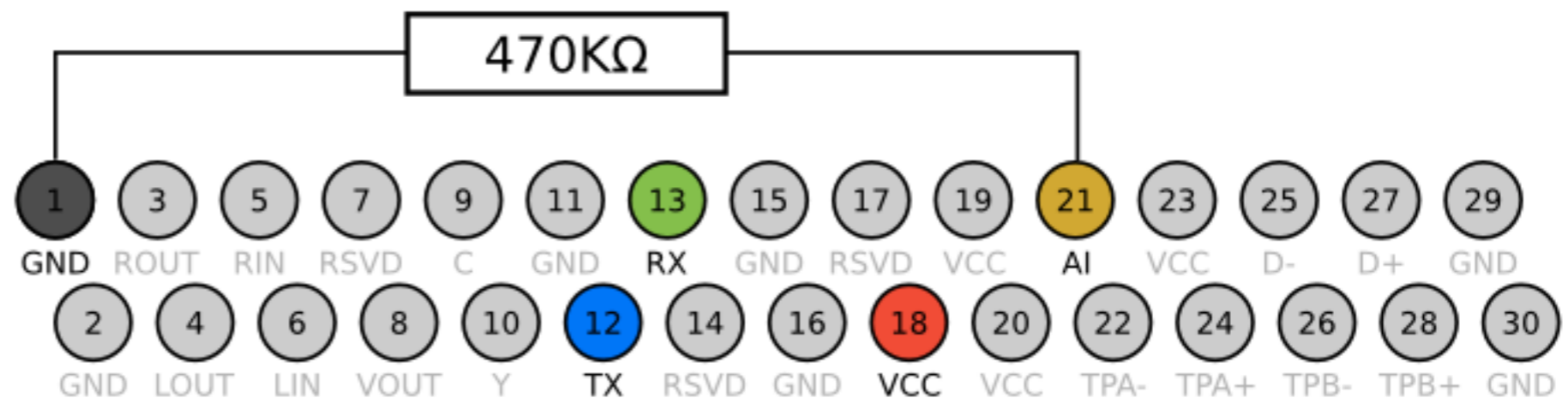
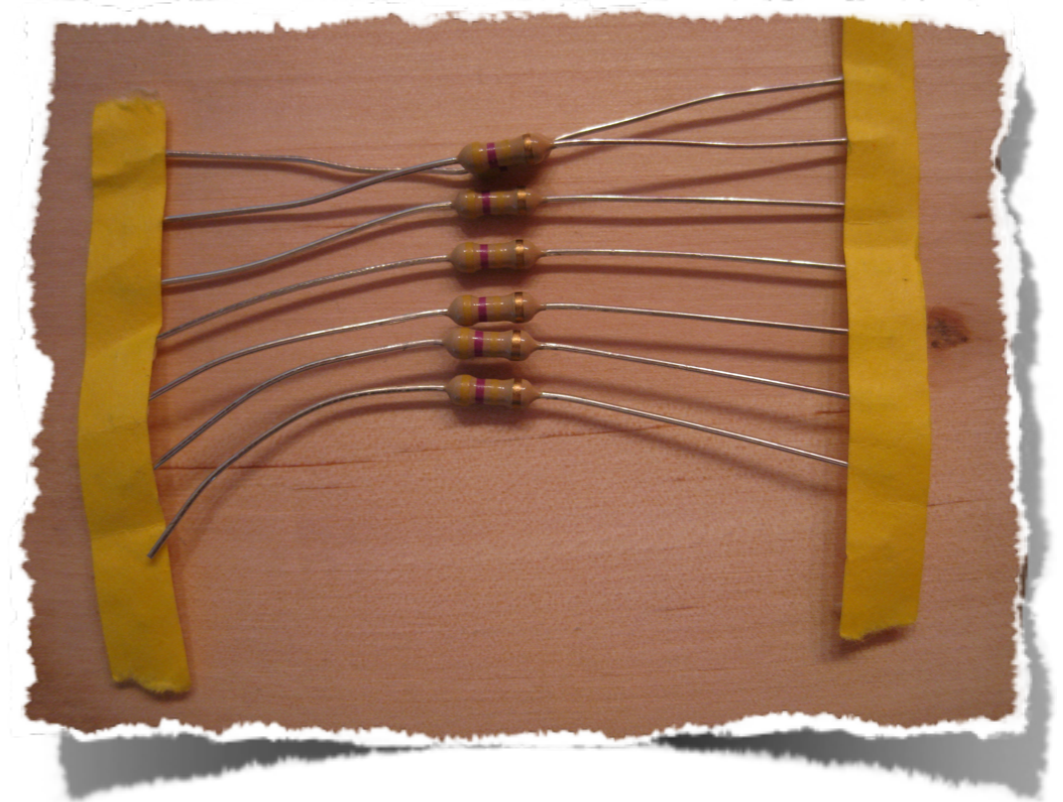
Breakout Board  
FT232RL USB to Serial

PodGizmo Connector



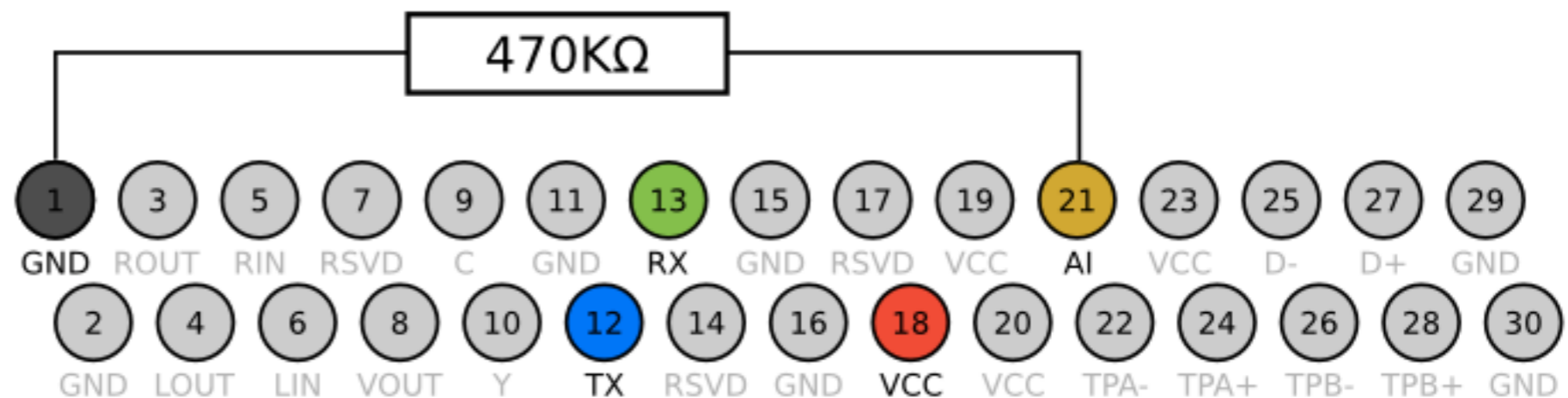
# Ingredients (I)

- 470 k $\Omega$  resistor
- used to bridge pin 1 and 21
- activates the UART
- costs a few cents



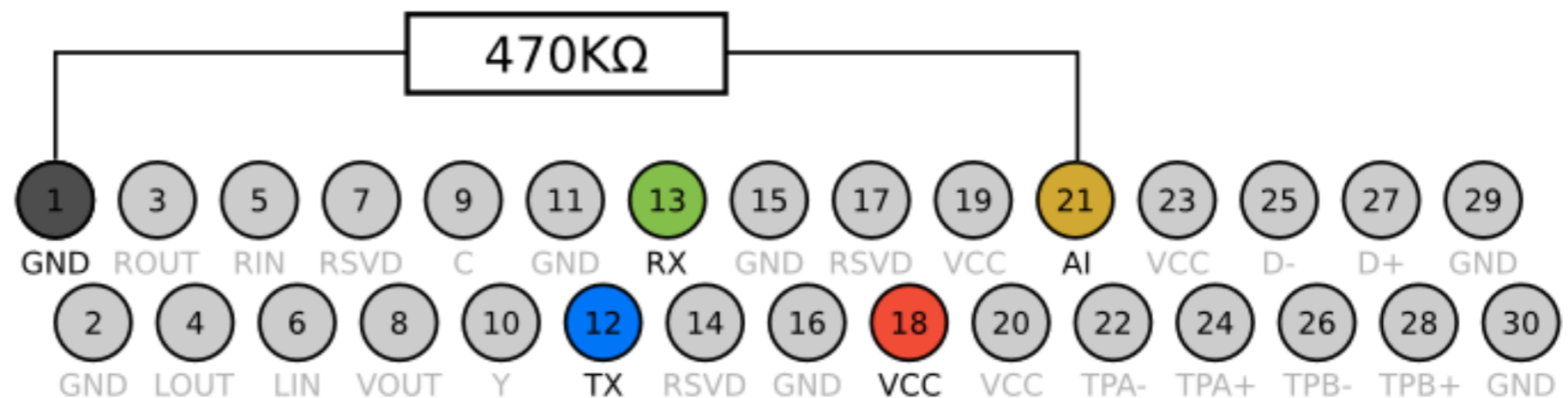
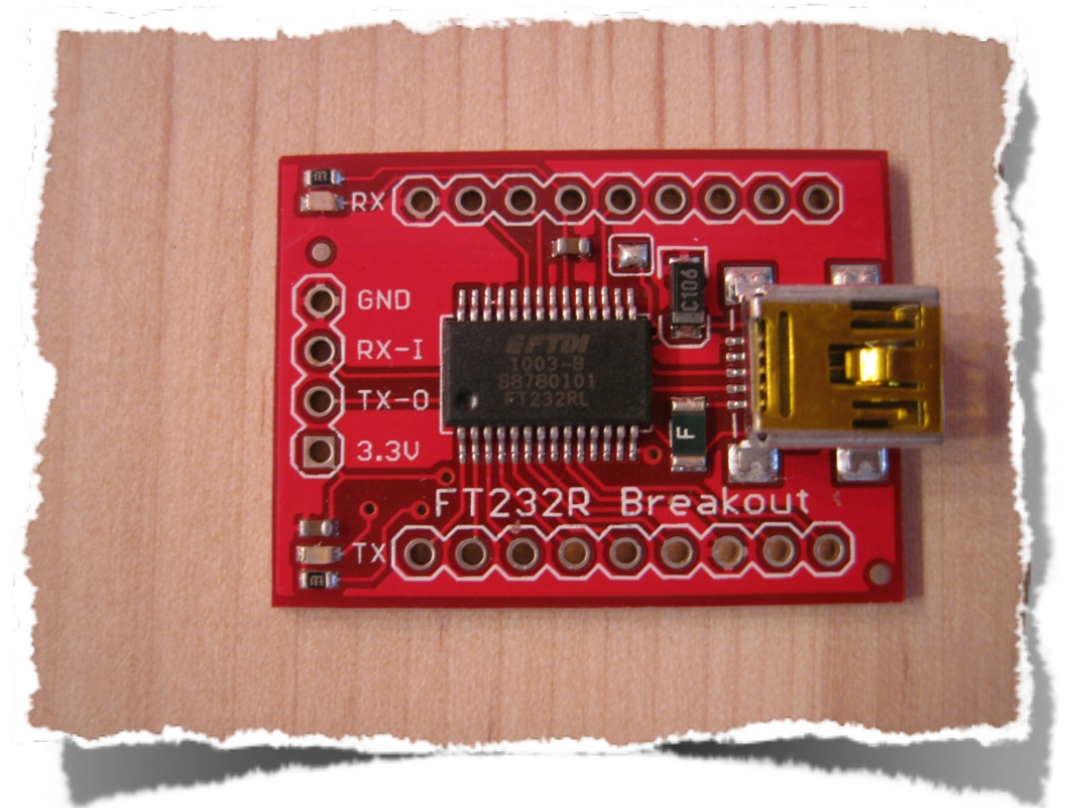
# Ingredients (II)

- PodBreakout
- easy access to dock connector pins
- some revisions have reversed pins
- even I was able to solder this
- about 12 EUR



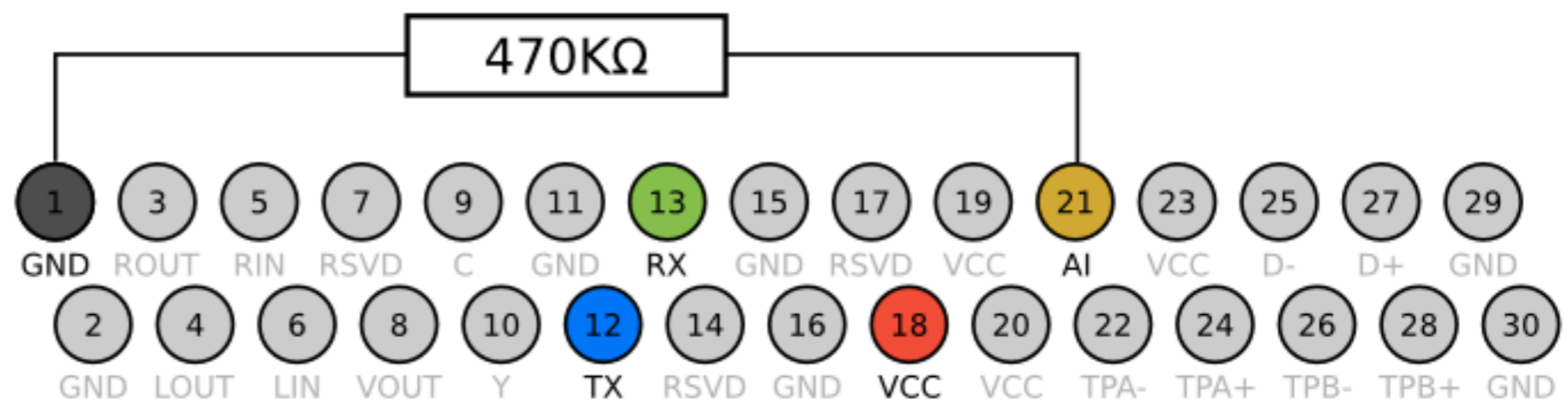
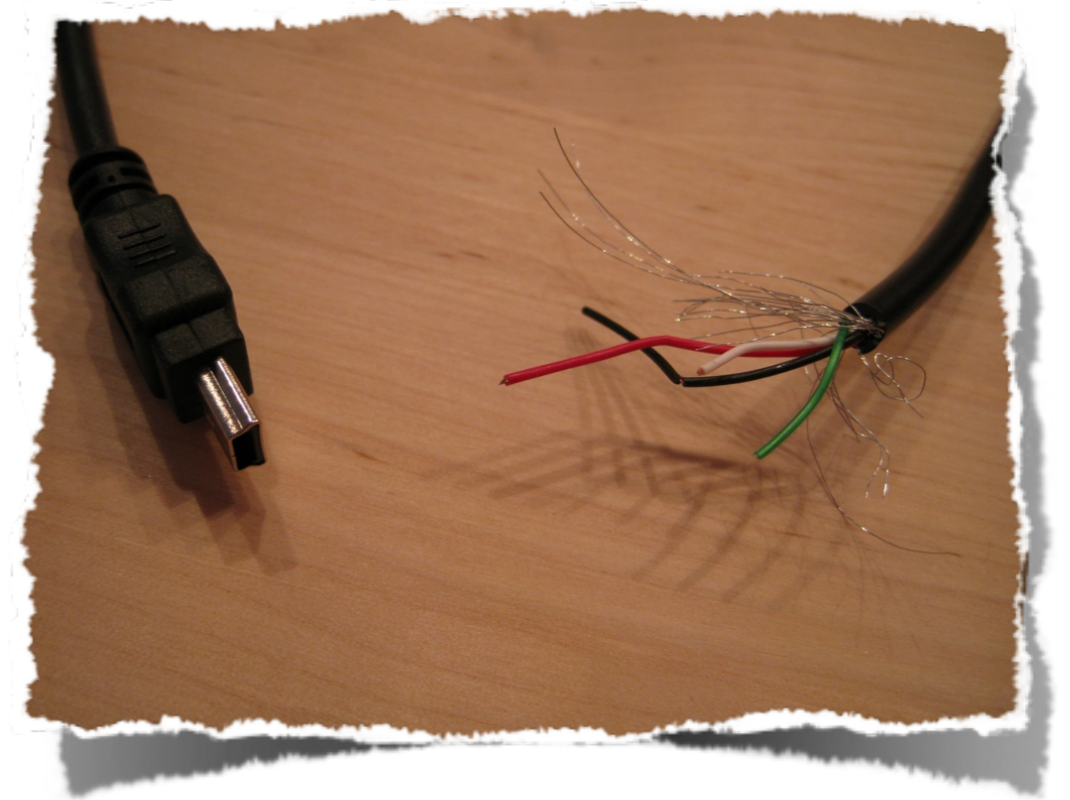
# Ingredients (III)

- FT232RL Breakout Board
- USB to Serial Convertor
- also very easy to solder
- about 10 EUR

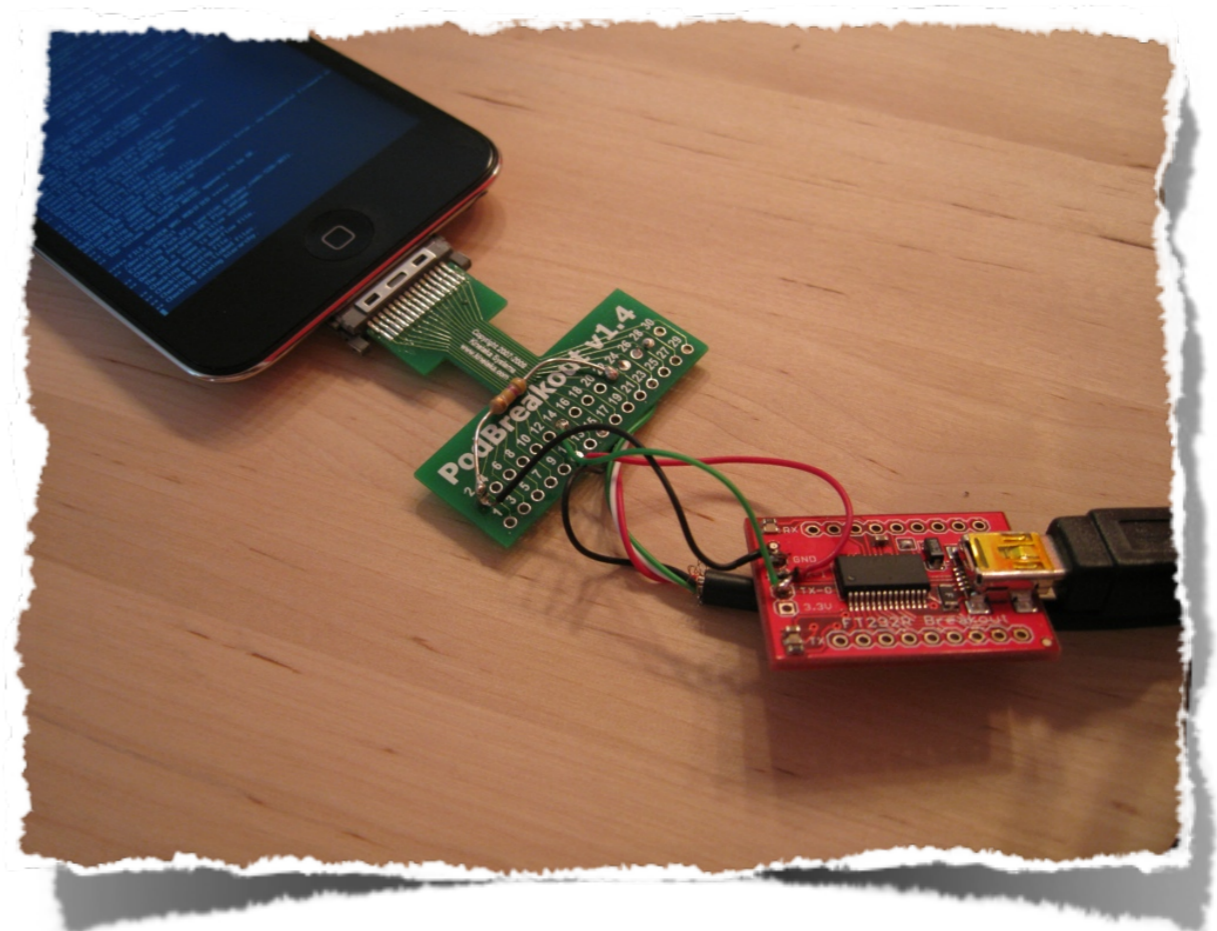
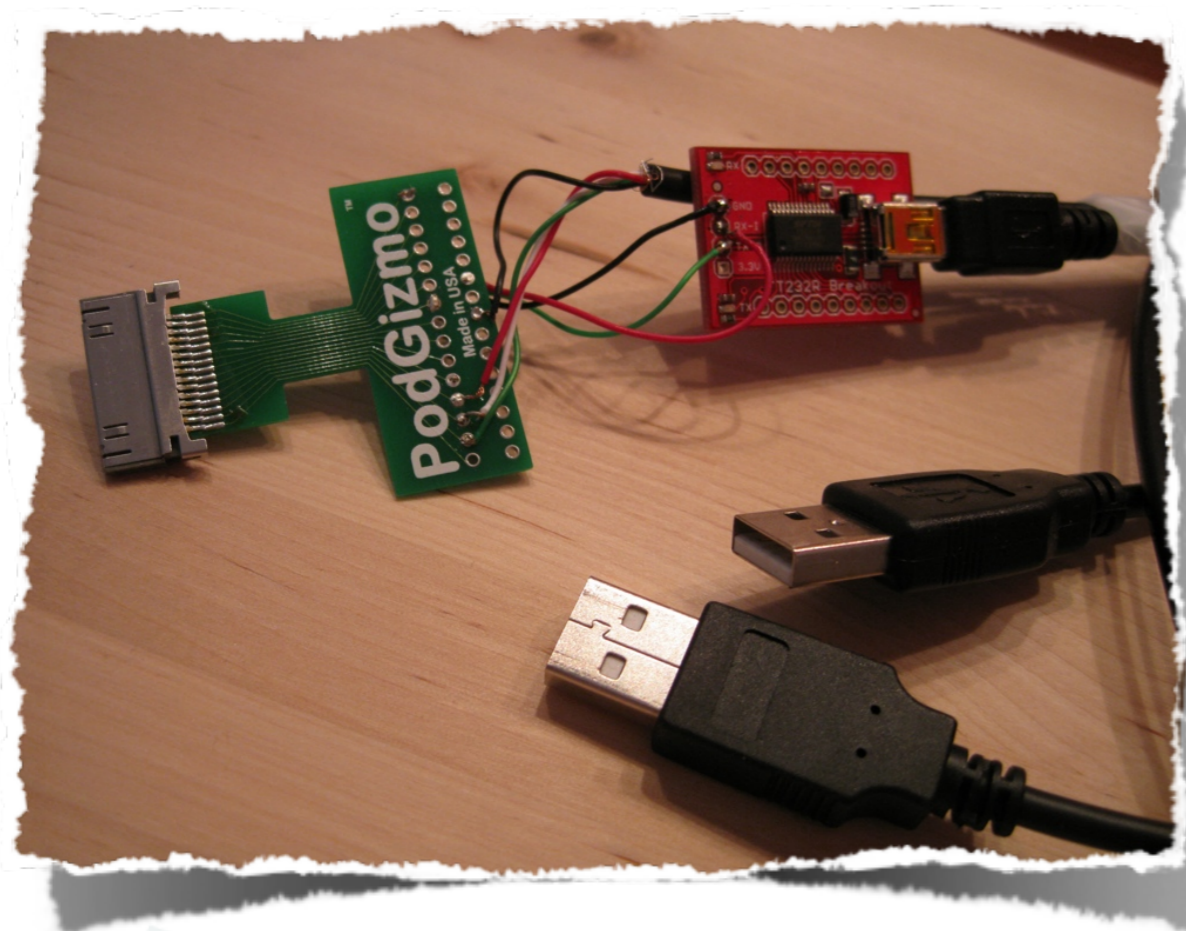


# Ingredients (IV)

- USB cables
- type A -> mini type B
- provides us with wires and connectors
- costs a few EUR



# Final USB and USB Serial Cable



- attaching a USB type A connector to the USB pins is very useful
- we can now do SSH over USB
- and kernel debug via serial line at the same time

# GDB and iOS KDP

- GDB coming with the iOS SDK has ARM support
- it also has KDP support
- however it can only speak KDP over UDP
- KDP over serial is not supported

# KDP over serial

- KDP over serial is sending fake ethernet UDP over serial
- SerialKDPProxy by David Elliott is able to act as serial/UDP proxy

```
$ SerialKDPProxy /dev/tty.usbserial-A600exos
Opening Serial
Waiting for packets, pid=362
^@AppleS5L8930XI0::start: chip-revision: C0
AppleS5L8930XI0::start: PIO Errors Enabled
AppleARMPL192VIC::start: _vicBaseAddress = 0xccaf5000
AppleS5L8930XGPIOIC::start: gpioicBaseAddress: 0xc537a000
AppleARMPerformanceController::traceBufferCreate: _pcTraceBuffer: 0xcca3a000 ...
AppleS5L8930XPerformanceController::start: _pcBaseAddress: 0xccb3d000
AppleARMPerformanceController configured with 1 Performance Domains
AppleS5L8900XI2SController::start: i2s0 i2sBaseAddress: 0xcb3ce400 i2sVersion: 2
...
AppleS5L8930XUSBPhy::start : registers at virtual: 0xcb3d5000, physical: 0x86000000
AppleVXD375 - start (provider 0x828bca00)
AppleVXD375 - compiled on Apr  4 2011 10:19:48
```

# Activating KDP on the iPhone

- KDP is only activated if the boot-arg "debug" is set
- boot-args can be set with special version of redsn0w / syringe
- or faked with a custom kernel
- patch your kernel to get into KDP anytime (e.g. breakpoint in unused syscall)

Name	Value	Meaning
DB_HALT	0x01	Halt at boot-time and wait for debugger attach.
DB_KPRT	0x08	Send kernel debugging kprintf output to serial port.
...	...	Other values might work but might be complicated to use.



# Using GDB...

```
$ /Developer/Platforms/iPhoneOS.platform/Developer/usr/bin/gdb -arch armv7 \  
kernelcache.iPod4,1_4.3.2_8H7.symbolized  
GNU gdb 6.3.50-20050815 (Apple version gdb-1510) (Fri Oct 22 04:12:10 UTC 2010)  
...  
(gdb) target remote-kdp  
(gdb) attach 127.0.0.1  
Connected.  
(gdb) i r  
r0          0x00  
r1          0x11  
r2          0x00  
r3          0x11  
r4          0x00  
r5          0x8021c814    -2145269740  
r6          0x00  
r7          0xc5a13efc    -979288324  
r8          0x00  
r9          0x27      39  
r10         0x00  
r11         0x00  
r12         0x802881f4    -2144828940  
sp          0xc5a13ee4    -979288348  
lr          0x8006d971    -2147034767  
pc          0x8006e110    -2147032816
```

Thank you for listening...

**QUESTIONS ?**

- xpwntool - <https://github.com/iH8sn0w/xpwn>
- SerialKDProxy - <http://tgwbd.org/svn/Darwin/SerialKDProxy/trunk/>
- IDA Scripts used during presentation soon at - <http://antid0te.com/idaiostoolkit/>