

# Adding ASLR to jailbroken iPhones

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



# 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
- Head of Research & Development at SektionEins GmbH

# Part I

## Introduction

# iPhone Security

- in 2010 iPhone Security got hit badly
- during the PWN2OWN contest it was demonstrated how to steal the SMS database with a ROP attack
- later this year jailbreakme.com presented a chain of exploits in safari, the kernel and a user - space application to jailbreak devices from remote
- an iPhone worm abusing these vulnerabilities was feared but did not surface
- all exploits have in common that payloads need to be pure ROP

# Jailbroken iPhones and Security

- as demonstrated in previous years the security of jailbroken iPhones is worse
- major reason for this is that jailbreaks disable several security features
- ROP is only required to kickstart the payload (mmap, mprotect)
- aside from that the payload can be normal ARM shellcode

# Protection Overview

Factory iPhone	Jailbroken iPhone
Codesigning Active	Codesigning Disabled
eXecutable Pages cannot be made writable and vice versa	Pages can be RWX
exploitation requires ROP	requires ROP / RET2LIBC to kickstart shellcode
no ASLR	no ASLR ( <i>until now</i> )
non executable stack / heap	non executable stack / heap
application sandboxing	sandboxing partly broken

# Codesigning ???

- every binary on the iPhone is digitally signed
- for every page there is a sha1 hash
- signatures are checked by the kernel
- check is performed when a page that is marked as executable is accessed
- if signature is invalid the process is killed by the kernel
- **disabled on jailbroken iPhones to allow homebrew applications to run**

# No ASLR on the iDevices

- iPhone / iPad / iPod does not have any address space randomization
- libraries are always mapped at the same address for performance reasons
- ASLR is considered costly by Apple
- iOS performance optimizations and the codesigning feature make ASLR tricky or impossible => it is unlikely that we will see ASLR in factory iPhones soon
- however for every firmware and for every device class the libraries are in a different position

# Example Crashdump

CrashReporter Key: bde777e1d1e21b0238bb9d7b65020f7eb2bc2410  
Hardware Model: iPhone3,1  
Process: gdb [226]  
Path: /usr/bin/gdb  
Identifier: gdb  
Version: ??? (???)  
Code Type: ARM (Native)  
Parent Process: sh [223]

Date/Time: 2010-11-25 09:22:26.271 +0100  
OS Version: iPhone OS 4.1 (8B117)  
Report Version: 104

Exception Type: EXC\_BAD\_ACCESS (SIGSEGV)  
Exception Codes: KERN\_INVALID\_ADDRESS at 0x5ffff7bc  
Crashed Thread: 0

## Thread 0 Crashed:

0 libSystem.B.dylib 0x307f93b8 0x30709000 + 983992  
...  
9 dyld 0x2fe01058 0x2fe00000 + 4184

## Thread 0 crashed with ARM Thread State:

r0: 0x00000002 r1: 0x2ffffb78 r2: 0x2ffffb84 r3: 0x2ffffbcc  
r4: 0x5ffff798 r5: 0x2fe2809c r6: 0x00000005 r7: 0x2ffff2a4  
r8: 0x307f93b5 r9: 0x00400b94 r10: 0x2fe2993c r11: 0x0000000f  
ip: 0x2fe280f4 sp: 0x2ffff288 lr: 0x2fe0c09d pc: 0x307f93b8  
cpsr: 0x00000030

## Binary Images:

0x1000 - 0x225fff +gdb arm /usr/bin/gdb  
0x2a1000 - 0x2e0fff libncurses.5.dylib arm /usr/lib/libncurses.5.dylib  
0x2fe00000 - 0x2fe26fff dyld armv7 &lt;a11905c8ef7906bf4b8910fc551f9dbb&gt; /usr/lib/dyld  
0x30130000 - 0x301f0fff libobjc.A.dylib armv7 &lt;49029949741e10f21b178b0a4b2df979&gt; /usr/lib/libobjc.A.dylib  
0x30709000 - 0x30816fff libSystem.B.dylib armv7 &lt;0792bef82e8cde31cb32d06e80262288&gt; /usr/lib/libSystem.B.dylib  
...  
0x33b79000 - 0x33bc3fff libstdc++.6.dylib armv7 &lt;7b2a8cf02f12c636c6db7f5e1906f9f0&gt; /usr/lib/libstdc++.6.dylib  
</string>

```
<key>displayName</key><string>gdb</string>
<key>name</key><string>gdb</string>
<key>os_version</key><string>iPhone OS 4.1 (8B117)</string>
<key>system_ID</key><string></string>
<key>version</key> <string>??? (???)</string>
```

</dict>

</plist>

# Lack of ASLR major weakness

- generation of ROP payload easy due to lack of ASLR
- attacker has hundreds of libraries to choose gadgets from
- payload can be easily ported between iPhone / iPad / iPod
- adding ASLR to (jailbroken) iPhones would increase security big time

# Part II

## Dyld Shared Cache and ASLR

# ASLR on jailbroken iPhones

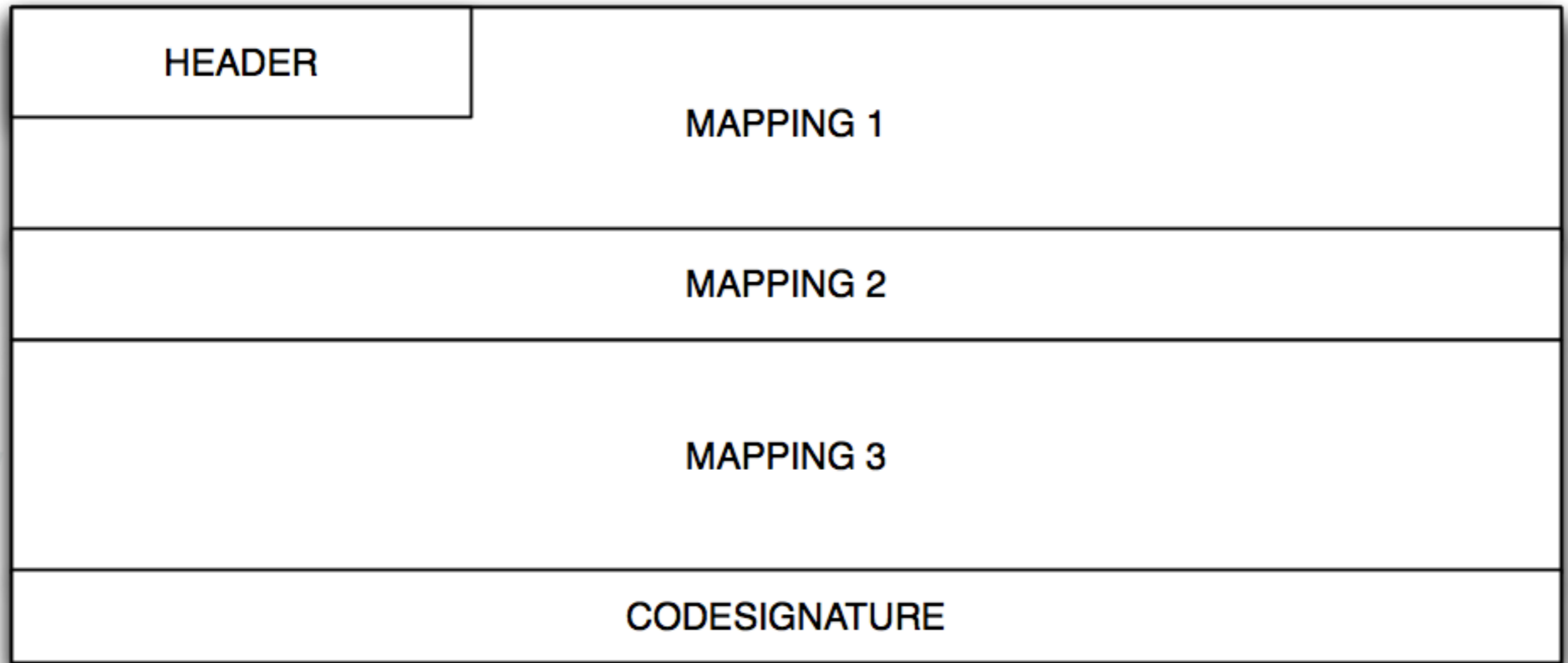
- goal is to add ASLR protection to jailbroken iPhones
- ASLR should be an extension to current jailbreaks
- ASLR should not destroy too much of the Apple optimizations
- ASLR should rebase libraries, dyld, stack and (in the future) heap

# Libraries where are thou?

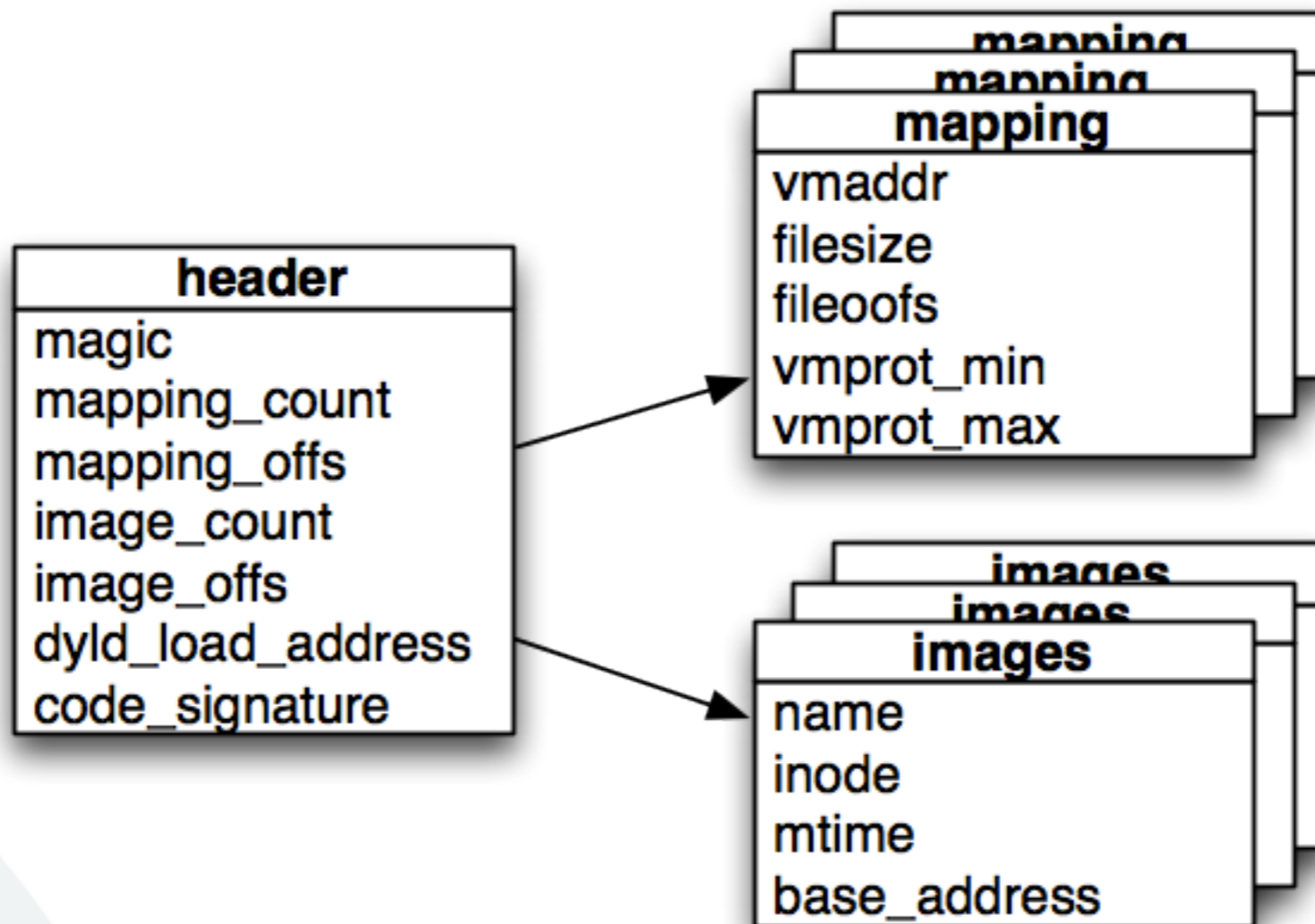
- since iPhoneOS / iOS 3.x shared libraries disappeared from the device
- because loading libraries is considered costly (time / memory)
- Apple moved all libraries into dyld\_shared\_cache
- technique also used in Snow Leopard

```
$ ls -la /Volumes/Jasper8C148.N900S/usr/lib/
total 336
drwxr-xr-x  6 sesser  staff    476 17 Nov 09:56 .
drwxr-xr-x  7 sesser  staff    238 17 Nov 08:46 ..
drwxr-xr-x  5 sesser  staff    170 17 Nov 09:06 dic
-rwxr-xr-x  1 sesser  staff 232704 22 Okt 06:15 dyld
drwxr-xr-x  2 sesser  staff    102 22 Okt 05:49 info
lrwxr-xr-x  1 sesser  staff     59 17 Nov 09:56 libIIOKit.A.dylib -> /System/Library/Frameworks/IIOKit...work/Versions/A/IIOKit
lrwxr-xr-x  1 sesser  staff     16 17 Nov 09:56 libIIOKit.dylib -> libIIOKit.A.dylib
lrwxr-xr-x  1 sesser  staff     16 17 Nov 09:06 libMatch.dylib -> libMatch.1.dylib
lrwxr-xr-x  1 sesser  staff     18 17 Nov 09:52 libcharset.1.0.0.dylib -> libcharset.1.dylib
lrwxr-xr-x  1 sesser  staff     15 17 Nov 09:52 libedit.dylib -> libedit.3.dylib
lrwxr-xr-x  1 sesser  staff     16 17 Nov 09:53 libexslt.dylib -> libexslt.0.dylib
lrwxr-xr-x  1 sesser  staff     18 17 Nov 09:23 libsandbox.dylib -> libsandbox.1.dylib
drwxr-xr-x  2 sesser  staff     68 22 Okt 06:10 libxslt-plugins
drwxr-xr-x  2 sesser  staff     68 22 Okt 05:47 system
```

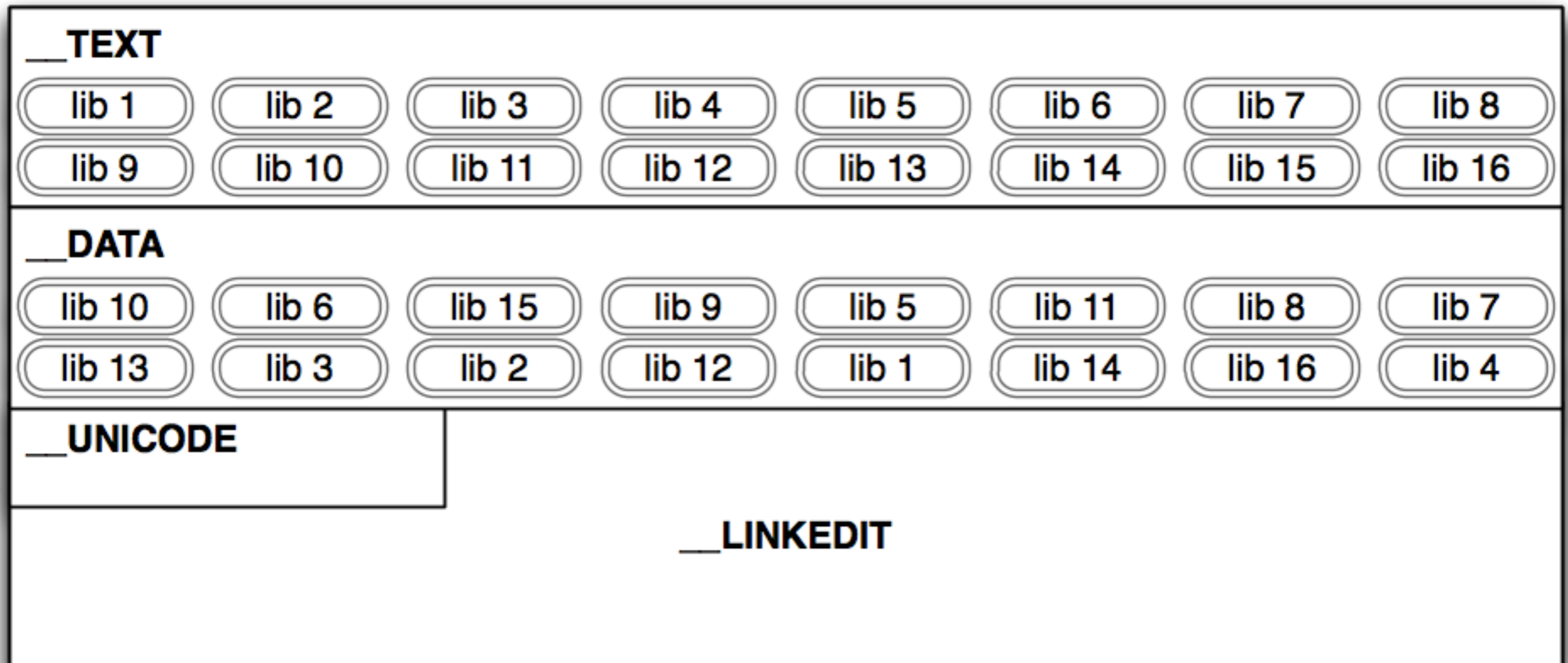
# dyld\_shared\_cache



# dyld\_shared\_cache Header



# dyld\_shared\_cache in detail



# Rebasing the Libraries...

- okay there are no libraries on the phone
- they are all in the dyld\_shared\_cache
- ➔ **problem: can we get them out of it ?**
  
- libraries in dyld\_shared\_cache have rebasing info stripped
- cannot rebase without rebasing information
- ➔ **problem: how do we get rebasing information ?**

# the iPhone SDK to the rescue? (I)



- wait a second ! there is the iPhone SDK
- it bundles the plain version of the libraries
- even better they contain rebasing information
- and they contain the armv6 and armv7 versions

# the iPhone SDK to the rescue? (II)



- 1st thought : „Hooray!“
- but then reality kicked in
- SDK bundles only a subset of libraries
- rebasing information cannot be used for segment splitting
  - ➔ rebasing information insufficient

# Segment Splitting ?

- linker groups `__TEXT` and `__DATA` sections
- one block of read / execute sections
- one block of read / write sections
- (base of `__DATA` - base of `__TEXT`) fixed in SDK libraries
  - ➔ however in `dyld_shared_cache` case we need to change delta

# Okay what now ?

looking at different shared caches revealed the following

- ➔ they seem to be made on the same machine
- ➔ the same binaries are used during construction
- ➔ library base addresses differ due to random load order

<b>/usr/lib/libSystem.B.dylib</b>			
	<b>iPhone 4</b>	<b>iPod 4</b>	<b>iPad</b>
<i>inode</i>	0x0933DE37	0x0933DE37	0x0933DE37
<i>mtime</i>	0x4CC1050A	0x4CC1050A	0x4CC1050A
<i>base</i>	<b>0x33B5C000</b>	<b>0x31092000</b>	<b>0x30D03000</b>

<b>/usr/lib/libobjc.dylib</b>			
	<b>iPhone 4</b>	<b>iPod 4</b>	<b>iPad</b>
<i>inode</i>	0x093AF2FC	0x093AF2FC	0x093AF2FC
<i>mtime</i>	0x4CC10998	0x4CC10998	0x4CC10998
<i>base</i>	<b>0x33476000</b>	<b>0x33A03000</b>	<b>0x34A7D000</b>

# How does this help us ?

- same binaries but different load address allows diffing
- in theory memory should only differ in places that require relocation
- simply diffing two caches should get us all rebasing positions
- ➔ in reality it is not that simple => many complications

# Obvious Complications

- different CPU type
  - ARMv6 => iPod 2G, iPhone 3G
  - ARMv7 => iPod 3G, iPod 4G, iPhone 3GS, iPhone 4, iPad
- iPod / iPhone / iPad have different features
  - libraries exist in one cache but not in the other
  - nothing to diff against ?

# More Comparison Partners

- diff against different CPU type => failed
- diff against beta version => failed
- diff against previous release => often fails
  - ➔ the 4.2, 4.2b, 4.2.1, 4.2.1a debacle ensures enough partners
- merging diffs => works for some devices
  - ➔ merge diff between iPhone 3GS and iPhone 4G  
and diff between iPhone 4G and iPod 4G

# Let's start diffing

- Python implementation
- uses macholib
- understands the dyld\_shared\_cache format
- diffs mach-o files
  - ensures same section (name, size, ...)
  - diffs section by section
  - diff is performed 4 byte aligned
  - ignores \_\_LINKEDIT
- differences printed to stdout

# Results of first diffing attempts

- found different types of differences
  - ➔ 2 large unknown values
  - ➔ 2 pointers inside the relocated binary
  - ➔ 2 pointers outside the relocated binary
  - ➔ 2 small unknown values
  - ➔ 1 small value vs. 1 pointer
  - ➔ 1 pointer vs. 1 small value

# Analysing the results (I)

- **Expected results**

- 2 pointers inside same binary => normal rebasing
- 2 pointers outside binary => imports

- **Unexpected results**

- 2 large values
- 2 small values
- 1 pointer vs. 1 small value

# Analysing the results (II)

more careful evaluation revealed even worse fact

- ➔ when 2 pointers are found they do not always point to the same symbol
- ➔ luckily this only occurs inside some `__objc_*` sections
- ➔ thought -> must be some ObjC weirdness

```
__objc_const:3E7C33C8
__objc_const:3E7C33CC
__objc_const:3E7C33D0
__objc_const:3E7C33D4
__objc_const:3E7C33D8
__objc_const:3E7C33DC
__objc_const:3E7C33E0
__objc_const:3E7C33E4
```

```
DCD 0xF
DCD 7
DCD 0x32CA2574
DCD aV804 ; "v8@0:4"
DCD __WebArchivePrivate_dealloc__+1
DCD 0x32CA58FB
DCD a804 ; "@8@0:4"
DCD __WebArchivePrivate_init__+1
```

```
__objc_const:3E3593C8
__objc_const:3E3593CC
__objc_const:3E3593D0
__objc_const:3E3593D4
__objc_const:3E3593D8
__objc_const:3E3593DC
__objc_const:3E3593E0
__objc_const:3E3593E4
```

```
DCD 0xF
DCD 7
DCD aInitwithcorear ; "initWithCoreArchive:"
DCD aI204Passrefptr ; "@I2@0:4{PassRefPtr<WebCore::LegacyWebAr"...
DCD __WebArchivePrivate_initWithCoreArchive__+1
DCD aSetcorearchive ; "setCoreArchive:"
DCD aV1204Passrefpt ; "v12@0:4{PassRefPtr<WebCore::LegacyWebAr"...
DCD __WebArchivePrivate_setCoreArchive__+1
```

# What are the two large unknown values ?

- very common in \_\_text section
- first believed to be a code difference
- using IDA to look at it revealed it is caused by different \_\_DATA - \_\_TEXT delta

```
__text:301FFB60
__text:301FFB60
__text:301FFB60  _mach_init          ; CODE XREF: j__mach_init+4!j
__text:301FFB60          ; DATA XREF: __la_symbol_ptr:mach_init_ptr!o
__text:301FFB60          PUSH                {R7,LR}
__text:301FFB62          ADD                 R7, SP, #0
__text:301FFB64          LDR                 R0, =(_mach_init_inited - 0x301FFB6A)
__text:301FFB66          ADD                 R0, PC
__text:301FFB68          LDR                 R0, [R0]
__text:301FFB6A          CBZ                 R0, loc_301FFB70
__text:301FFB6C          MOVS                R0, #0
__text:301FFB6E          B                   locret_301FFB7C
__text:301FFB70          ; -----
__text:301FFB70  loc_301FFB70          ; CODE XREF: _mach_init+A!j
__text:301FFB70          LDR                 R3, =(_mach_init_inited - 0x301FFB78)
__text:301FFB72          MOVS                R2, #1
__text:301FFB74          ADD                 R3, PC
__text:301FFB76          STR                 R2, [R3]
__text:301FFB78          BLX                 j__mach_init_doit
__text:301FFB7C  locret_301FFB7C          ; CODE XREF: _mach_init+E!j
__text:301FFB7C          POP                 {R7,PC}
__text:301FFB7C          ; End of function _mach_init
__text:301FFB7C          ; -----
__text:301FFB7E          ALIGN 0x10
__text:301FFB80  off_301FFB80          DCD _mach_init_inited - 0x301FFB6A
__text:301FFB80          ; DATA XREF: _mach_init+4!r
__text:301FFB84  off_301FFB84          DCD _mach_init_inited - 0x301FFB78
__text:301FFB84          ; DATA XREF: _mach_init:loc_301FFB70!r
__text:301FFB88
```



# Small Values and Pointers

- some files contain small values that do not match
- sometimes there is a small value in one file and a pointer in the other
- occurs only in `__objc_*` sections
- emphasizes the need of objc reversing

```
055/321 /.../DataAccess.framework/DataAccess
-----
__text
-----
...
__objc_imageinfo
-----
__objc_const
small value + ptr 0000000f 337d1611
small value + ptr 00000012 32bcc832
ptr + small value 30b12832 0000000f
ptr + small value 30af14cd 0000000d
-----
__objc_selrefs
-----
__objc_classrefs
-----
__objc_superrefs
-----
__objc_data
-----
__data
global 10836
address 5917
delta 4916
sel 0
```

# Reversing the objc differences

- grabbed objc-4 source code from <http://developer.apple.com/>
- tried to find the responsible code
- soon turned out to be more complicated
- source code matches only partially

# iPhone libobjc does not match the source (I)

```
struct objc_selopt_t {  
  
    uint32_t version; /* this is version 3: external cstrings */  
    uint32_t capacity;  
    uint32_t occupied;  
    uint32_t shift;  
    uint32_t mask;  
    uint32_t zero;  
    uint64_t salt;  
    uint64_t base;  
  
    uint32_t scramble[256];  
    /* tab[mask+1] */  
    uint8_t tab[0];  
    /* offsets from &version  
       to cstrings  
       int32_t offsets[capacity];  
    */  
};
```

```
objc_opt_ro:3003A2D0 __objc_opt_data DCD 8 ; DATA XREF: __nl_symbol_ptr:__objc_opt_data_ptr[0]  
objc_opt_ro:3003A2D4 version DCD 8  
objc_opt_ro:3003A2D8 capacity DCD 131072  
objc_opt_ro:3003A2DC occupied DCD 65801  
objc_opt_ro:3003A2E0 shift DCD 47  
objc_opt_ro:3003A2E4 mask DCD 0x1FFF  
objc_opt_ro:3003A2E8 zero_low DCD 0  
objc_opt_ro:3003A2EC padding DCD 0  
objc_opt_ro:3003A2F0 salt_low DCD 0x72873569  
objc_opt_ro:3003A2F4 salt_high DCD 0xBE1E08C4  
objc_opt_ro:3003A2F8 scramble DCD 0, 0x5524, 0xA517, 0x1EB8E, 0x13D51, 0xAA74, 0x1B4AB, 0x1A10, 0xAAB, 0x5  
objc_opt_ro:3003A2F8 DCD 0x14A98, 0x1B032, 0x7111, 0x1E8E8, 0xEF08, 0x640C, 0xDD, 0xD635, 0x1A245  
objc_opt_ro:3003A2F8 DCD 0x5F9B, 0x19F1E, 0x1D870, 0x34BC, 0x1FADF, 0x11A70, 0x1E796, 0x1E71B, 0x  
objc_opt_ro:3003A2F8 DCD 0xDA6D, 0x1625A, 0x1D55F, 0x18649, 0x77BD, 0x11EA2, 0x11C9A, 0x1EF74, 0x  
objc_opt_ro:3003A2F8 DCD 0x1ACEC, 0x9C1B, 0x8DB, 0xF142, 0x16641, 0x32FD, 0x188C2, 0x73C6, 0x1980  
objc_opt_ro:3003A2F8 DCD 0x1628A, 0x188A0, 0x10214, 0xB2A0, 0x2A59, 0x15727, 0x244F, 0x409A, 0xBF  
objc_opt_ro:3003A2F8 DCD 0x7ACE, 0x15314, 0xECF0, 0x14C2D, 0xC38E, 0x1D7ED, 0xDC59, 0xA532, 0x790  
objc_opt_ro:3003A2F8 DCD 0xBC6A, 0xBDBD, 0x9FE5, 0x1BC57, 0x8F60, 0x1A7F8, 0x3A39, 0x1187D, 0xF5E  
objc_opt_ro:3003A2F8 DCD 0x158A, 0xF430, 0xC95D, 0x1E42C, 0x1EDF8, 0x12C86, 0x9217, 0x16EA9, 0x17  
objc_opt_ro:3003A2F8 DCD 0x1589C, 0x8D11, 0xAD4C, 0x851F, 0x1D3D2, 0x13558, 0x10BC0, 0x7B07, 0x50  
objc_opt_ro:3003A2F8 DCD 0x1187E, 0x1733F, 0xF2DC, 0x9406, 0x38B8, 0x10FBF, 0x14162, 0x19050, 0x0  
objc_opt_ro:3003A2F8 DCD 0x13C2D, 0x17A59, 0x1AD64, 0xD486, 0x9FBB, 0x4604, 0x1E76B, 0x37A2, 0x70  
objc_opt_ro:3003A2F8 DCD 0x1FF0B, 0x1D172, 0x9134, 0x10543, 0x1CB14, 0x12507, 0x1B562, 0x1599F, 0  
objc_opt_ro:3003A2F8 DCD 0x10B24, 0x91B7, 0x6139, 0xC88, 0x18B62, 0x6661, 0x5A64, 0x1FD4F, 0x1BD  
objc_opt_ro:3003A2F8 DCD 0x94AA, 0x1C179, 0x84E2, 0x15C1E, 0xE210, 0x6B04, 0x1A9F1, 0xBB59, 0xD50  
objc_opt_ro:3003A2F8 DCD 0x182E8, 0x16EE7, 0x4BC9, 0x77F1, 0x175B3, 0x1F725, 0x78E0, 0x178DD, 0x  
objc_opt_ro:3003A6F8 tab DCB 6, 0, 0x18, 1, 7, 0x22, 0xF, 0x12, 8, 0x1A, 2, 2, 8; 0  
objc_opt_ro:3003A6F8 DCB 0, 6, 7, 0xD, 0x2D, 0xA, 5, 2, 0xE, 0, 0, 0, 0x23; 13  
objc_opt_ro:3003A6F8 DCB 0, 5, 0x26, 0x3A, 4, 0x1F, 2, 0x1D, 9, 3, 6, 3, 0x1D; 26  
objc_opt_ro:3003A6F8 DCB 0x53, 4, 7, 0xB, 3, 2, 3, 0x24, 1, 0x53, 2, 0xA, 3; 39  
objc_opt_ro:3003A6F8 DCB 1, 0, 0x2A, 8, 0x45, 0xC, 7, 0, 0x1F, 0x13, 4, 2, 0xB; 52  
objc_opt_ro:3003A6F8 DCB 0, 0x16, 0x4C, 0, 3, 4, 0, 1, 6, 0x32, 1, 2, 2, 0xE; 65  
objc_opt_ro:3003A6F8 DCB 5, 0xE, 0x11, 7, 7, 0xE, 7, 0xF, 0xA, 0xB, 0xB, 0x15; 79  
objc_opt_ro:3003A6F8 DCB 0xE, 1, 0x21, 2, 0x21, 8, 4, 0, 4, 1, 0, 5, 1, 0x22; 91  
objc_opt_ro:3003A6F8 DCB 0xE, 8, 6, 0x10, 0, 0, 0xB, 0, 1, 2, 0x12, 2, 0x10; 105  
objc_opt_ro:3003A6F8 DCB 0xF, 0, 0, 8, 0x39, 4, 0, 1, 0, 0x27, 1, 5, 0, 0x25; 118  
objc_opt_ro:3003A6F8 DCB 0, 0, 4, 0, 3, 2, 0, 7, 0x14, 0, 4, 0, 5, 1, 2, 7; 132  
objc_opt_ro:3003A6F8 DCB 4, 5, 3, 3, 0x28, 3, 0xE, 0xD, 8, 0x14, 0, 2, 0x41; 148  
objc_opt_ro:3003A6F8 DCB 2, 6, 0, 0, 0x11, 6, 0, 0, 0xD, 2, 0x13, 9, 2, 5, 0; 161  
objc_opt_ro:3003A6F8 DCB 3, 0, 7, 0x22, 0, 0x27, 5, 1, 0xB, 3, 0, 9, 0x1E, 0x1D; 176  
objc_opt_ro:3003A6F8 DCB 4, 0, 8, 0, 1, 0, 0, 0x22, 0x10, 5, 6, 0, 8, 0, 0x1B; 190  
objc_opt_ro:3003A6F8 DCB 0xD, 0xE, 1, 0, 0, 0, 0, 1, 0, 1, 0xA, 6, 0x25, 0; 205  
objc_opt_ro:3003A6F8 DCB 8, 0x14, 7, 0x1E, 0, 0x13, 0, 3, 2, 0x16, 0x2F, 0xC; 219  
objc_opt_ro:3003A6F8 DCB 0x15, 0x2, 0x34, 3, 0, 0x17, 8, 0x2, 0x12, 0x16, 0, 231
```

# iPhone libobjc does not match the source (I)

- unknown large blob is the offset table
- which is a list of offsets to selector names
- knowing the content it is easy to relocate
  
- on the iPhone the offset table is followed by an unknown table
- unknown table has capacity many entries of size 1 byte
- according to twitter it is a one byte checksum of the selector name

# Analysing the different pointer problem

```
__objc_const:3E7C33C6
__objc_const:3E7C33C7
__objc_const:3E7C33C8
__objc_const:3E7C33CC
__objc_const:3E7C33D0
__objc_const:3E7C33D4
__objc_const:3E7C33D8
__objc_const:3E7C33DC
__objc_const:3E7C33E0
__objc_const:3E7C33E4
__objc_const:3E7C33E8
__objc_const:3E7C33EC
__objc_const:3E7C33F0
__objc_const:3E7C33F4
__objc_const:3E7C33F8
__objc_const:3E7C33FC
__objc_const:3E7C3400
__objc_const:3E7C3404
__objc_const:3E7C3408
__objc_const:3E7C340C
__objc_const:3E7C3410
__objc_const:3E7C3414
__objc_const:3E7C3418
__objc_const:3E7C341C
__objc_const:3E7C3420
__objc_const:3E7C3424
__objc_const:3E7C3425

DCB 0
DCB 0
DCD 0xF
DCD 7
DCD 0x32CA2574
DCD aV804 ; "v8@0:4"
DCD __WebArchivePrivate_dealloc_+1
DCD 0x32CA58FB
DCD a804 ; "@e@0:4"
DCD __WebArchivePrivate_init_+1
DCD aInitwithcorear ; "initWithCoreArchive:"
DCD a1204Passrefptr ; "@l2@0:4{PassRefPtr<WebCore::LegacyWebAr"...
DCD __WebArchivePrivate_initWithCoreArchive_+1
DCD aSetcorearchive ; "setCoreArchive:"
DCD aV1204Passrefpt ; "v12@0:4{PassRefPtr<WebCore::LegacyWebAr"...
DCD __WebArchivePrivate_setCoreArchive_+1
DCD aCorearchive ; "coreArchive"
DCD aLegacywebarchi ; "^{LegacyWebArchive=i{RefPtr<WebCore::Ar"...
DCD __WebArchivePrivate_coreArchive_+1
DCD 0x3507FFA8
DCD aV804 ; "v8@0:4"
DCD __WebArchivePrivate_cxx_destruct_+1
DCD 0x3508088C
DCD a804 ; "@e@0:4"
DCD __WebArchivePrivate_cxx_construct_+1
DCB 0x14
DCB 0
```

looking at it with IDA reveals that method tables are simply resorted

```
__objc_const:3E3593C6
__objc_const:3E3593C7
__objc_const:3E3593C8
__objc_const:3E3593CC
__objc_const:3E3593D0
__objc_const:3E3593D4
__objc_const:3E3593D8
__objc_const:3E3593DC
__objc_const:3E3593E0
__objc_const:3E3593E4
__objc_const:3E3593E8
__objc_const:3E3593EC
__objc_const:3E3593F0
__objc_const:3E3593F4
__objc_const:3E3593F8
__objc_const:3E3593FC
__objc_const:3E359400
__objc_const:3E359404
__objc_const:3E359408
__objc_const:3E35940C
__objc_const:3E359410
__objc_const:3E359414
__objc_const:3E359418
__objc_const:3E35941C
__objc_const:3E359420
__objc_const:3E359424
__objc_const:3E359425

DCB 0
DCB 0
DCD 0xF
DCD 7
DCD aInitwithcorear ; "initWithCoreArchive:"
DCD a1204Passrefptr ; "@l2@0:4{PassRefPtr<WebCore::LegacyWebAr"...
DCD __WebArchivePrivate_initWithCoreArchive_+1
DCD aSetcorearchive ; "setCoreArchive:"
DCD aV1204Passrefpt ; "v12@0:4{PassRefPtr<WebCore::LegacyWebAr"...
DCD __WebArchivePrivate_setCoreArchive_+1
DCD aCorearchive ; "coreArchive"
DCD aLegacywebarchi ; "^{LegacyWebArchive=i{RefPtr<WebCore::Ar"...
DCD __WebArchivePrivate_coreArchive_+1
DCD 0x33023574
DCD aV804 ; "v8@0:4"
DCD __WebArchivePrivate_dealloc_+1
DCD 0x330268FB
DCD a804 ; "@e@0:4"
DCD __WebArchivePrivate_init_+1
DCD 0x335DEFA8
DCD aV804 ; "v8@0:4"
DCD __WebArchivePrivate_cxx_destruct_+1
DCD 0x335DF88C
DCD a804 ; "@e@0:4"
DCD __WebArchivePrivate_cxx_construct_+1
DCB 0x14
DCB 0
```

# Analysing the small values

- reason for differences in small values was not discovered until dyld\_shared\_cache was relocated and applications did not work
- objc applications could not find selectors
- problem was finally found with reverse engineering
- lower 2 bits of size field used as a flag
- method list sorted by selectors => allows faster lookup

```
typedef struct method_t {  
    SEL name;  
    const char *types;  
    IMP imp;  
} method_t;
```

```
typedef struct method_list_t {  
    uint32_t entsize_NEVER_USE; // low 2 bits used for fixup markers  
    uint32_t count;  
    struct method_t first;  
} method_list_t;
```

# What needs to be rebased?

- images must be shifted around
- image pointers in dyld\_shared\_cache header
- Mach-O-Headers
  - segment addresses / segment file offsets
  - section addresses / section file offsets
  - LC\_ROUTINES
  - symbols
  - export trie
- section content according to collected differences
- \_\_objc\_opt\_ro selector table in libobjc.dylib

# Part III

Relocating the rest ?

# The Dynamic Linker

- iPhone OS / iOS does not relocate the dynamic linker
- dyld is always loaded at 0x2FE00000
- contains enough code to kickstart injected code on jailbroken iPhone
- if not relocated dyld\_shared\_cache relocation is futile

```
... from 0x2fe00000 to 0x2fe01000 (r-x, max r-x; copy, private, not-reserved)
... from 0x2fe01000 to 0x2fe02000 (r-x, max rwx; copy, private, not-reserved)
... from 0x2fe02000 to 0x2fe28000 (r-x, max r-x; copy, private, not-reserved)
... from 0x2fe28000 to 0x2fe53000 (rw-, max rw-; copy, private, not-reserved) (2 sub-regions)
... from 0x2fe53000 to 0x2fe63000 (r--, max r--; copy, private, not-reserved)
```

# Relocating the Dynamic Linker (I)

- dyld linked to 0x2FE00000 but contains relocation information
- relocation should be possible with the standard tools

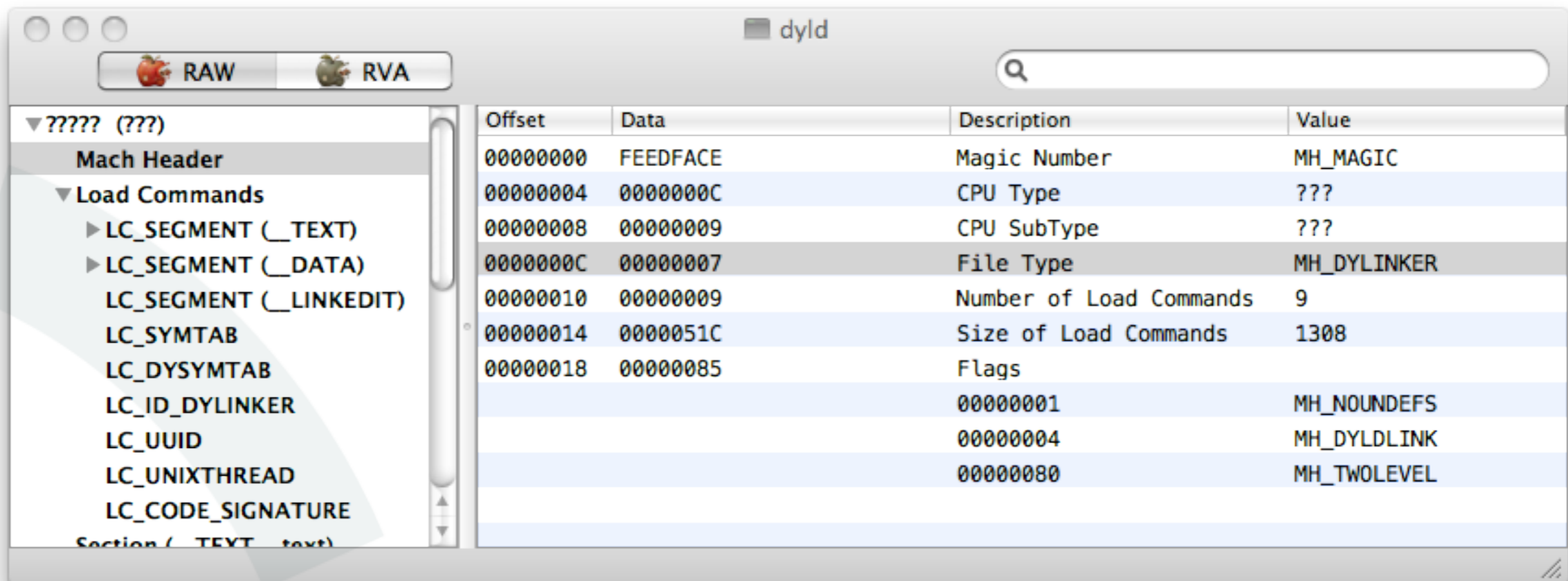
```
$ dyldinfo -rebase /Volumes/Jasper8C148.N900S/usr/lib/dyld | more
rebase information (from local relocation records and indirect symbol table):
segment  section          address          type
__DATA   __mod_init_func  0x2FE28000      pointer
__DATA   __mod_init_func  0x2FE28004      pointer
__DATA   __mod_init_func  0x2FE28008      pointer
__DATA   __mod_init_func  0x2FE2800C      pointer
__DATA   __mod_init_func  0x2FE28010      pointer
__DATA   __const          0x2FE281C0      pointer
__DATA   __const          0x2FE281C4      pointer
__DATA   __const          0x2FE281C8      pointer
__DATA   __const          0x2FE281CC      pointer
__DATA   __const          0x2FE281D0      pointer
__DATA   __const          0x2FE281D4      pointer
__DATA   __const          0x2FE281D8      pointer
__DATA   __const          0x2FE281DC      pointer
__DATA   __const          0x2FE281E0      pointer
__DATA   __const          0x2FE281E4      pointer
__DATA   __const          0x2FE281E8      pointer
...
```

# Relocating the Dynamic Linker (II)

- rebase tool fails because it does not support MH\_DYLINKER files

```
$ rebase -v -low_address 0x23456000 /Volumes/Jasper8C148.N900S/usr/lib/dyld
rebase warning: file is not a dylib or bundle for /Volumes/Jasper8C148.N900S/usr/lib/dyld
```

- file type can be modified with e.g. Mach-O-View



# Relocating the Dynamic Linker (III)

- rebase tool does work now

```
$ rebase -v -low_address 0x23456000 /Volumes/Jasper8C148.N900S/usr/lib/dyld
arm 0x2FE00000 -> 0x23456000 /Volumes/Jasper8C148.N900S/usr/lib/dyld
```

- however the rebased linker will crash on the iPhone
- the reason is that rebase does not relocate the entrypoint

```
$ otool -l ./dyld
...
Load command 7
  cmd LC_UNIXTHREAD
  cmdsize 84
  flavor ARM_THREAD_STATE
  count ARM_THREAD_STATE_COUNT
    r0 0x00000000 r1 0x00000000 r2 0x00000000 r3 0x00000000
    r4 0x00000000 r5 0x00000000 r6 0x00000000 r7 0x00000000
    r8 0x00000000 r9 0x00000000 r10 0x00000000 r11 0x00000000
    r12 0x00000000 r13 0x00000000 r14 0x00000000 r15 0x2fe01028
    r16 0x00000000
...
```

# Stack Randomization (I)

- stack of main thread is allocated near dyld
- usually mapped at  
`0x2FD00000 -> 0x2FDFFFFFF` or `0x2FF10000 -> 0x2FFFFFFF`
- predictable location allows for easier exploitation of e.g. stack based buffer overflows
- relocation can be done in user - space

# Stack Randomization (II)

- implemented by adding code to dyld
- dyld entrypoint is relocated to our code
  - a new 1 MB memory block is mapped (at a random address)
  - stack guard page is set to PROT\_NONE
  - stack data is copied into new block
  - stack pointer is moved into new block
  - old block is unmapped
  - execution is transferred to dyld\_start

# Stack Randomization Test

```
int main()
{
    char buffer[1024];

    printf("%08x\n", buffer);
}
```

```
lib:/tmp root# ./xxx
2ffffef88
lib:/tmp root# ./xxx
2ccd8f88
lib:/tmp root# ./xxx
2bf5af88
lib:/tmp root# ./xxx
2e7a5f88
lib:/tmp root# ./xxx
2e9b0f88
lib:/tmp root# ./xxx
2abf3f88
lib:/tmp root# ./xxx
2ce31f88
```

# Part IV

## Installing it on the iPhone

# Installing on the iPhone

- current iOS 4.1 jailbreaks are user - space jailbreaks
- only iOS 4.2.1 jailbreak for iPod 2G / iPhone 3G and 3GS (old bootrom) are not user - space
- replacing dyld / dyld\_shared\_cache not possible due to codesigning
- „replacing“ must happen after jailbreak is active

# Analysis of current Jailbreak

- /sbin/launchd launched in a way that libgmalloc.dylib is loaded
- libgmalloc.dylib is a fake dylib that just replaces stack with payload
- stack payload launches a ROP attack
- ROP is used for sysctl()
  - security.mac.proc\_enforce=0
  - security.mac.vnode\_enforce=0
- and executing /usr/lib/pf2
- pf2 is a kernel exploit that patches the kernel and re-executes /sbin/launchd

# Installing only the alternative cache

- pf2 is patched to execute `/sbin/antid0te` instead of `/sbin/launchd`
- antid0te sets environment variables
  - `DYLD_SHARED_REGION = private`
  - `DYLD_SHARED_CACHE_DONT_VALIDATE = 1`
  - `DYLD_SHARED_CACHE_DIR = /System/Library/Caches/antid0te`
- and executes `/sbin/launchd`

# Installing an alternative dyld

- pf2 is patched to execute `/sbin/antid0te` instead of `/sbin/launchd`
- `antid0te` copies patches dyld to `/usr/lib/dyld.antid0te`
- copies `/usr/lib/dyld` to `/usr/lib/dyld.orig`
- creates directory `/usr/lib/antid0te`
- copies `antif0te.hfs` to `/usr/lib/antid0te.hfs`
- creates a symbolic link in `/usr/lib/antid0te` pointing to `../dyld.orig`
- replaces `/usr/lib/dyld` with symbolic link to `antid0te/dyld`
- binds `/dev/vn0` to `/usr/lib/antid0te.hfs`
- `/dev/vn0` is mounted to `/usr/lib/antid0te`

# Installing the easy way

- because installation is kinda complicated there is a simple tool
- tool is called antid0te and will be released in a few days
- <http://www.antid0te.com>

Thank you for listening...

**DEMO**  
**of**  
**antid0te**

Thank you for listening...

**QUESTIONS ?**