SektionEins
http://www.sektioneins.de

Antidote 2.0 - ASLR in iOS

Stefan Esser <stefan.esser@sektioneins.de>

HITBSECCONF2011
AMSTERDAM
MAY 17 - 20 @ NH GRAND KRASNAPOLSKY

Who am I?

Stefan Esser

- from Cologne / Germany
- in information security since 1998
- PHP core developer since 2001
- Month of PHP Bugs and Suhosin
- recently focused on iPhone security (ASLR, jailbreak)
- Head of R&D at SektionEins GmbH

Part I

Introduction

iPhone Security in 2010

- iPhone security got strucked twice
    - first during PWN2OWN (SMS database stolen with ROP payload)
    - again by jailbreakme.com (full remote jailbreak)
- lack of ASLR in iOS recognized as major weakness
- in december Antid0te demonstrated an ASLR solution for jailbroken iPhones

iPhone Security in 2011

- Apple released their own ASLR implementation with iOS 4.3
- several iOS updates to solve remotely exploitable flaws in MobileSafari
- another iOS update to solve the location gate problem
- but no updates to fix local kernel vulnerability used for current jailbreaks
- more security researchers concentrate on iOS kernel vulnerabilities

Topics

- What were the challenges in adding ASLR to the iPhone
- How did Antid0te's ASLR work around them without the help of Apple
- How does Apple's own ASLR implementation work
- How combining both implementation is even more secure
- What are the limitations of ASLR on the iPhone

Part II

ASLR vs. iOS

ASLR vs. iOS

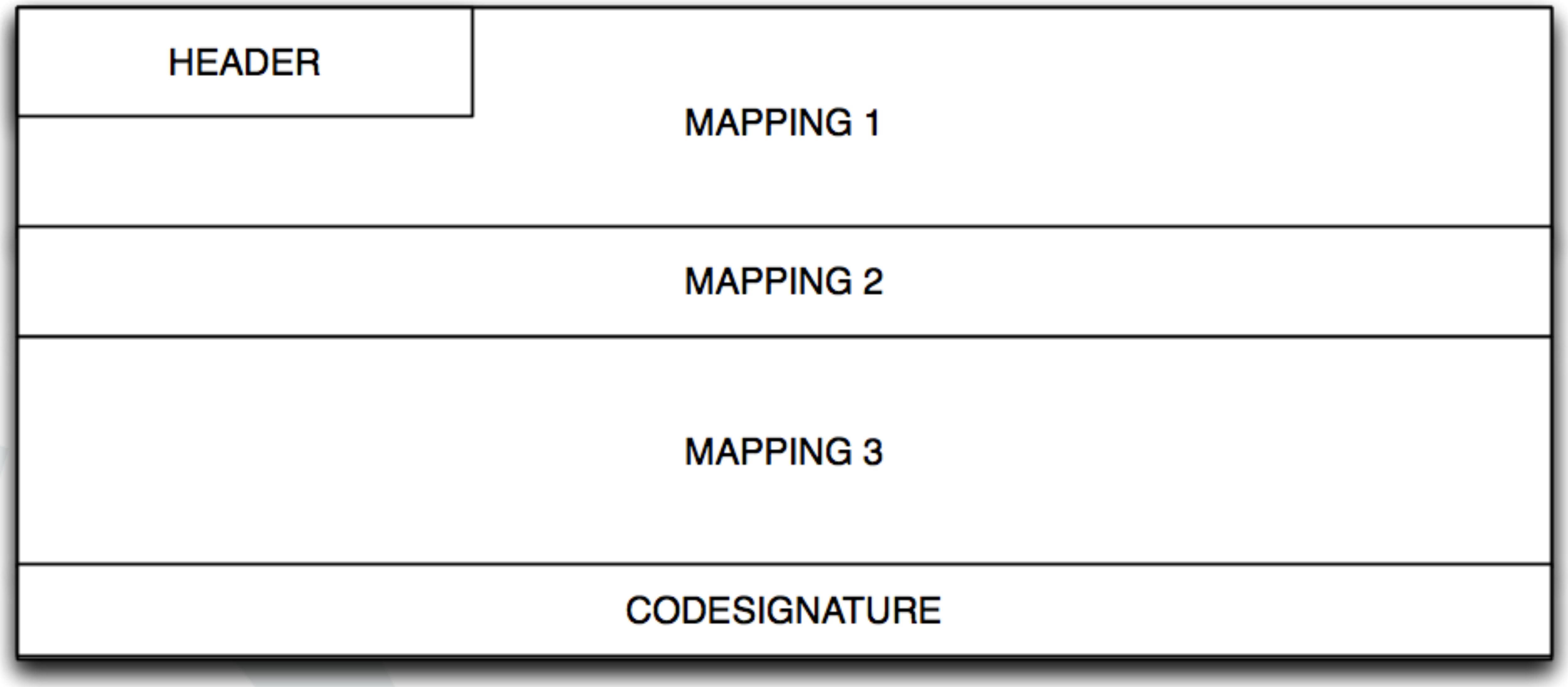
- iOS 4.2.x had no randomization at all (libs, dyld, stack, heap, ...)
- ASLR hard to implement due to Apple's optimizations (dyld_shared_cache)
- Codesigning major roadblock for adding effective ASLR
- binaries don't have relocation information

Libraries where are thou?

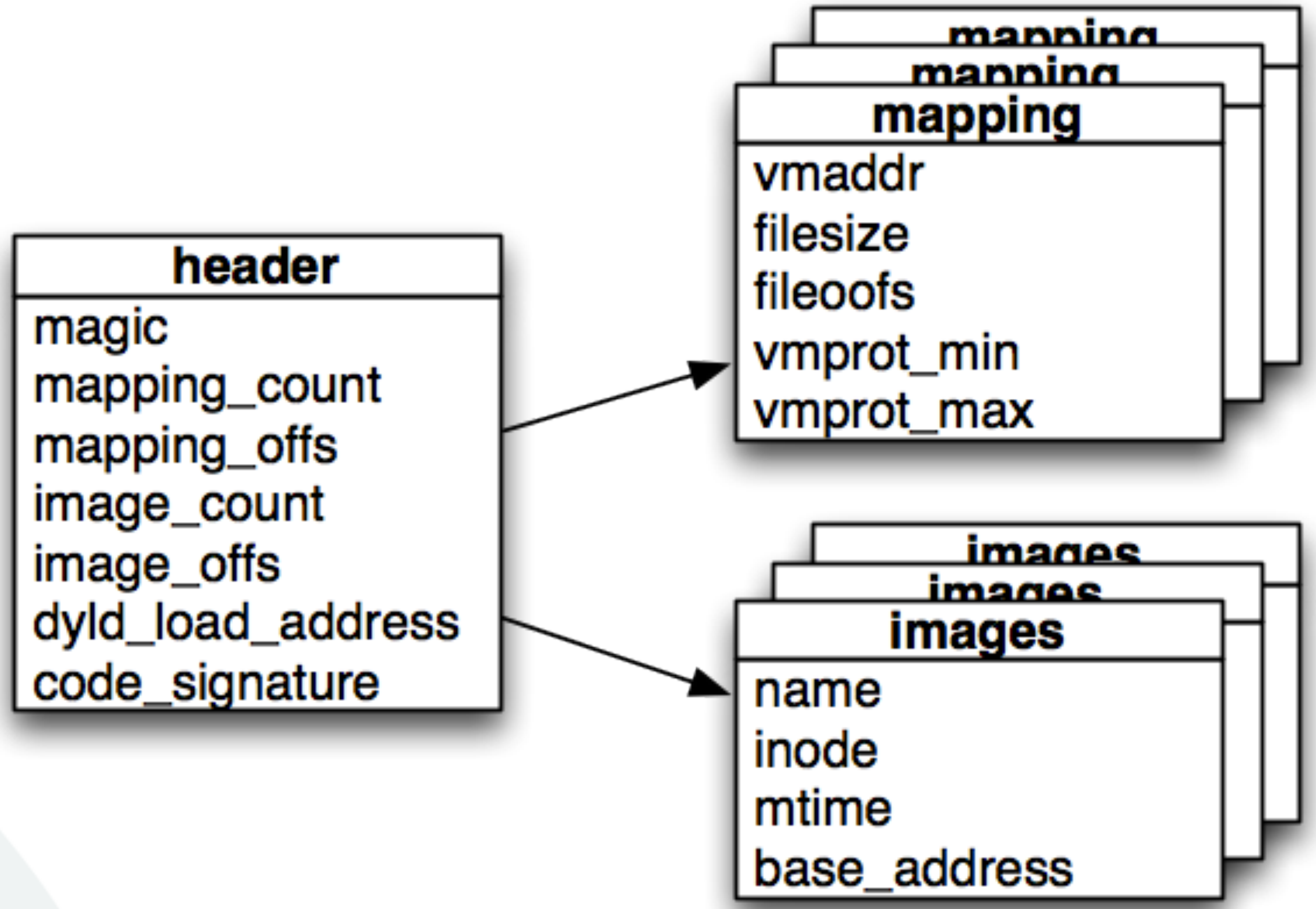
- since iPhoneOS / iOS 3.x shared libraries disappeared from the device
- because loading libraries is considered costly (time / memory)
- Apple moved all libraries into dyld_shared_cache
- technique also used in Snow Leopard

```
$ ls -la /Volumes/Jasper8C148.N90OS/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 libIOKit.A.dylib -> /System/Library/Frameworks/IOKit...work/Versions/A/IOKit
lrwxr-xr-x  1 sesser  staff      16 17 Nov 09:56 libIOKit.dylib -> libIOKit.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 in iOS <= 4.2.x

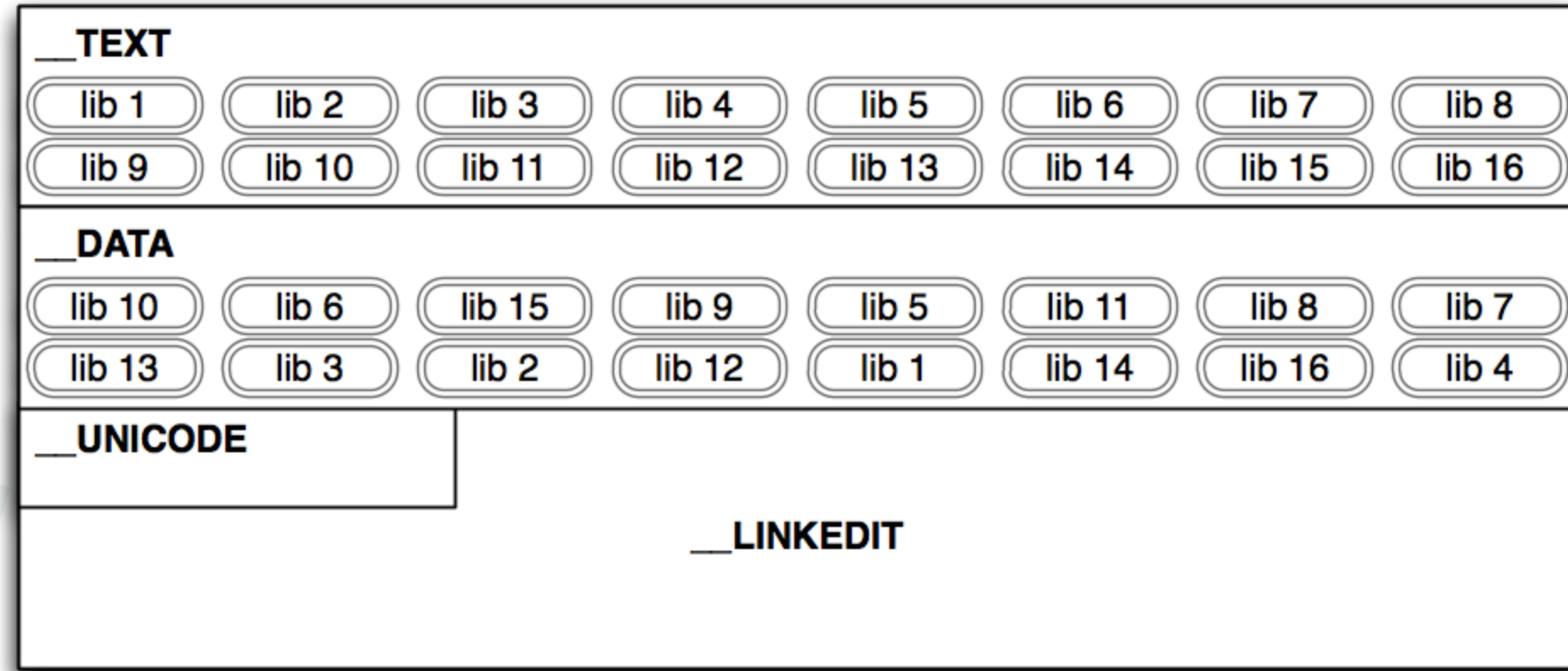


dyld_shared_cache Header in iOS <= 4.2.x



SektionEins

dyld_shared_cache in detail



dyld_shared_cache vs. ASLR

- libraries in cache are loaded at a fixed base address
- moving or shuffling requires to know fixup addresses
- no relocation information in binaries

- segment splitting - code and data compiled to specific delta
- moving or shuffling libraries requires to adjust delta
- positions of deltas unknown and also not in usual reloc info

Part III

Antid0te 1.0 - How did it work?

Antid0te

- Antid0te's goals were
  - to add ASLR to jailbroken iPhones
  - to not destroy the optimizations performed by Apple
- Codesigning not a problem because it is disabled on jailbroken phones
- Lack of relocation information major problem

Okay what did we do ?

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

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

/usr/lib/libobjc.dylib			
	iPhone 4	**iPod 4**	**iPad**
inode	0x093AF2FC	0x093AF2FC	0x093AF2FC
mtime	0x4CC10998	0x4CC10998	0x4CC10998
base	**0x33476000**	**0x33A03000**	**0x34A7D000**

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 ?

What to compare against each other ?

- 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 ensured enough partners
  - ➡ the rushed release of 4.3 / 4.3.1 / 4.3.2 helps again
  - ➡ 4.3.3 for iPad is problematic
- 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                 DCD 0xF
__objc_const:3E7C33CC                 DCD 7
__objc_const:3E7C33D0                 DCD 0x32CA2574
__objc_const:3E7C33D4                 DCD aV804               ; "v8@0:4"
__objc_const:3E7C33D8                 DCD __WebArchivePrivate_dealloc_+1
__objc_const:3E7C33DC                 DCD 0x32CA58FB
__objc_const:3E7C33E0                 DCD a804                ; "@8@0:4"
__objc_const:3E7C33E4                 DCD __WebArchivePrivate_init_+1

__objc_const:3E3593C8                 DCD 0xF
__objc_const:3E3593CC                 DCD 7
__objc_const:3E3593D0                 DCD aInitwithcorear     ; "initWithCoreArchive:"
__objc_const:3E3593D4                 DCD a1204Passrefptr     ; "@12@0:4{PassRefPtr<WebCore::LegacyWebAr"...
__objc_const:3E3593D8                 DCD __WebArchivePrivate_initWithCoreArchive__+1
__objc_const:3E3593DC                 DCD aSetcorearchive     ; "setCoreArchive:"
__objc_const:3E3593E0                 DCD aV1204Passrefpt     ; "v12@0:4{PassRefPtr<WebCore::LegacyWebAr"...
__objc_const:3E3593E4                 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                 EXPORT _mach_init
__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
__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
__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: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
```

Large unknown values in libobjc.dylib

- inside libobjc.dylib there is a huge blob of unknown large values that differs
- had no idea what this was - made me fear a roadstop
- source code access or reversing libobjc.dylib required => see later

```
__objc_opt_ro:3003A6F8                DCB 1, 2, 0, 0x21, 9, 0, 0x2D, 1, 0xD, 0xC, 0x27, 0x33; 8125
__objc_opt_ro:3003A6F8                DCB 8, 0x1A, 0x30, 1, 0x23, 5, 0, 0xD, 0x42, 0x21, 0x26; 8137
__objc_opt_ro:3003A6F8                DCB 0xA, 0x19, 0x32, 0xB, 0x16, 0x20, 5, 0x3B, 0x13, 0x6C; 8148
__objc_opt_ro:3003A6F8                DCB 7, 0x23, 0x14, 4, 0x28, 0xB, 0x15, 0x2F, 0x49, 0x3C; 8158
__objc_opt_ro:3003A6F8                DCB 9, 0xA, 0x45, 2, 0x1B, 0x2D, 0, 0, 4, 2, 0, 3, 2, 2; 8168
__objc_opt_ro:3003A6F8                DCB 0x12, 0, 1, 0x17, 0x2D, 0x2A, 8, 5, 0, 8; 8182
__objc_opt_ro:3003C6F8 unknown_blob   DCD 0x92DE39, 0x10, 0xCB4F1E, 0x1DF6055, 0x51991C8, 0x1DE8D8B, 0x4A7AEA2, 0x159D4D9; 0
__objc_opt_ro:3003C6F8                DCD 0x1DE2C5C, 0x10, 0x10, 0x44FB7E1, 0x10, 0x643545, 0x10, 0x16C12A8; 8
__objc_opt_ro:3003C6F8                DCD 0x10, 0x10, 0x502BA49, 0x10, 0x10, 0xB4FD74, 0x10, 0x10; 16
__objc_opt_ro:3003C6F8                DCD 0x10, 0x4F88B58, 0x38C998E, 0x10, 0x301AD1E, 0x10, 0x10, 0x502D3A5; 24
__objc_opt_ro:3003C6F8                DCD 0x10, 0x10, 0x10, 0x181EF66, 0x50332AB, 0xB514EF, 0xAD48ED, 0x659FB0; 32
__objc_opt_ro:3003C6F8                DCD 0x3000CC8, 0x1C089C, 0x10, 0x10, 0x4A721CA, 0x4A808EF, 0x10, 0x10; 40
__objc_opt_ro:3003C6F8                DCD 0x303144, 0x10, 0x192D468, 0x10, 0x10, 0x1929C8C, 0x10, 0x10; 48
__objc_opt_ro:3003C6F8                DCD 0x373692, 0x10, 0x10, 0x10, 0x16CCB86, 0x518D681, 0x10, 0x10; 56
__objc_opt_ro:3003C6F8                DCD 0x10, 0x10, 0x71A7B7, 0x30032DF, 0x37F3A21, 0xCFD239, 0x667AEC, 0x10; 64
__objc_opt_ro:3003C6F8                DCD 0x4A7CA11, 0x10, 0x302A14E, 0x1456CDF, 0x16CADEB, 0x3EA5C83, 0x65D426, 0x10; 72
__objc_opt_ro:3003C6F8                DCD 0x16C3C28, 0x518F304, 0xB51DAD, 0x10, 0x929605, 0x38B9575, 0x10, 0x38BBB68; 80
__objc_opt_ro:3003C6F8                DCD 0x10, 0x10, 0x10, 0x10, 0x10, 0x38BC543, 0x358F300, 0xCB4622; 88
__objc_opt_ro:3003C6F8                DCD 0x19210DD, 0x439E260, 0x10, 0x439B50B, 0x11E0ED1, 0x1D0A19F, 0x935B53, 0x10; 96
__objc_opt_ro:3003C6F8                DCD 0x10, 0x502A363, 0x10, 0x301D7E8, 0x10, 0x10, 0x10, 0x53CB5D8; 104
__objc_opt_ro:3003C6F8                DCD 0x10, 0x10, 0x10, 0x10, 0x93201E, 0x10, 0x10, 0x10; 112
__objc_opt_ro:3003C6F8                DCD 0x10, 0x30063BB, 0xB066A4, 0x306CDE, 0x300F9D1, 0x12540F8, 0x16D10CB, 0x10; 120
__objc_opt_ro:3003C6F8                DCD 0x64334B, 0x1DF475A, 0x666386, 0x10, 0x10, 0x53C9223, 0x136B47B, 0x10; 128
__objc_opt_ro:3003C6F8                DCD 0x10, 0x10, 0x10, 0x10, 0x16C2281, 0x1AF9124, 0x4A82743, 0x10; 136
__objc_opt_ro:3003C6F8                DCD 0x358149B, 0x10, 0x2FEB479, 0x10, 0x10, 0x1B6DAA3, 0x1AFF725, 0x1C1AEEB; 144
__objc_opt_ro:3003C6F8                DCD 0x10, 0x10, 0x66F48F, 0x10, 0x10, 0x3A97952, 0x38BA5C3, 0x644152; 152
__objc_opt_ro:3003C6F8                DCD 0x662FCB, 0x10, 0x10, 0x549A471, 0x2FEE06C, 0x37F35FC, 0x10, 0x10; 160
__objc_opt_ro:3003C6F8                DCD 0x10, 0x3026456, 0x10, 0x4A7E2D5, 0x1A7311B, 0x10, 0x10, 0x10; 168
__objc_opt_ro:3003C6F8                DCD 0x10, 0x10, 0x124D1E2, 0x1BFB29C, 0x645850, 0x3032717, 0x10, 0x10; 176
__objc_opt_ro:3003C6F8                DCD 0x10, 0x10, 0x10, 0x65B33D, 0x10, 0x10, 0x3018717, 0x10; 184
__objc_opt_ro:3003C6F8                DCD 0x10, 0x10, 0x10, 0x473DDA9, 0x124C4A5, 0x10, 0x660566, 0x3653799; 192
```

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|o
_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,
_objc_opt_ro:3003A2F8                 DCD 0x14A98, 0x1B032, 0x7111, 0x1E8E8, 0xEF08, 0x640C, 0xDD, 0xD635,
_objc_opt_ro:3003A2F8                 DCD 0x5F9B, 0x19F1E, 0x1D870, 0x34BC, 0x1FADF, 0x11A70, 0x1E796, 0x1E71B,
_objc_opt_ro:3003A2F8                 DCD 0xDA6D, 0x1625A, 0x1D55F, 0x18649, 0x77BD, 0x11EA2, 0x11C9A, 0x1EF74,
_objc_opt_ro:3003A2F8                 DCD 0x1ACEC, 0x9C1B, 0x8DB, 0xF142, 0x16641, 0x32FD, 0x188C2, 0x73C6,
_objc_opt_ro:3003A2F8                 DCD 0x1628A, 0x188A0, 0x10214, 0xB2A0, 0x2A59, 0x15727, 0x244F, 0x409A,
_objc_opt_ro:3003A2F8                 DCD 0x7ACE, 0x15314, 0xECF0, 0x14C2D, 0xC38E, 0x1D7ED, 0xDC59, 0xA532,
_objc_opt_ro:3003A2F8                 DCD 0xBC6A, 0xBDBD, 0x9FE5, 0x1BC57, 0x8F60, 0x1A7F8, 0x3A39, 0x1187D,
_objc_opt_ro:3003A2F8                 DCD 0x158A, 0xF430, 0xC95D, 0x1E42C, 0x1EDF8, 0x12C86, 0x9217, 0x16EA9,
_objc_opt_ro:3003A2F8                 DCD 0x1589C, 0x8D11, 0xAD4C, 0x851F, 0x1D3D2, 0x13558, 0x10BC0, 0x7B07,
_objc_opt_ro:3003A2F8                 DCD 0x1187E, 0x1733F, 0xF2DC, 0x9406, 0x38B8, 0x10FBF, 0x14162, 0x19050,
_objc_opt_ro:3003A2F8                 DCD 0x13C2D, 0x17A59, 0x1AD64, 0xD486, 0x9FBB, 0x4604, 0x1E76B, 0x37A2,
_objc_opt_ro:3003A2F8                 DCD 0x1FF0B, 0x1D172, 0x9134, 0x10543, 0x1CB14, 0x12507, 0x1B562, 0x1599F,
_objc_opt_ro:3003A2F8                 DCD 0x10B24, 0x91B7, 0x6139, 0xC88, 0x18B62, 0x6661, 0x5A64, 0x1FD4F,
_objc_opt_ro:3003A2F8                 DCD 0x94AA, 0x1C179, 0x84E2, 0x15C1E, 0xE210, 0x6B04, 0x1A9F1, 0xBB59,
_objc_opt_ro:3003A2F8                 DCD 0x182E8, 0x16EE7, 0x4BC9, 0x77F1, 0x175B3, 0x1F725, 0x78E0, 0x178DD,
_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
```

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                 DCB    0
__objc_const:3E7C33C7                 DCB    0
__objc_const:3E7C33C8                 DCD 0xF
__objc_const:3E7C33CC                 DCD 7
__objc_const:3E7C33D0                 DCD 0x32CA2574
__objc_const:3E7C33D4                 DCD aV804               ; "v8@0:4"
__objc_const:3E7C33D8                 DCD __WebArchivePrivate_dealloc_+1
__objc_const:3E7C33DC                 DCD 0x32CA58FB
__objc_const:3E7C33E0                 DCD a804                ; "@8@0:4"
__objc_const:3E7C33E4                 DCD __WebArchivePrivate_init_+1
__objc_const:3E7C33E8                 DCD aInitwithcorear     ; "initWithCoreArchive:"
__objc_const:3E7C33EC                 DCD a1204Passrefptr     ; "@12@0:4{PassRefPtr<WebCore::LegacyWebAr"...
__objc_const:3E7C33F0                 DCD __WebArchivePrivate_initWithCoreArchive__+1
__objc_const:3E7C33F4                 DCD aSetcorearchive     ; "setCoreArchive:"
__objc_const:3E7C33F8                 DCD aV1204Passrefpt     ; "v12@0:4{PassRefPtr<WebCore::LegacyWebAr"...
__objc_const:3E7C33FC                 DCD __WebArchivePrivate_setCoreArchive__+1
__objc_const:3E7C3400                 DCD aCorearchive        ; "coreArchive"
__objc_const:3E7C3404                 DCD aLegacywebarchi     ; "^{LegacyWebArchive=i{RefPtr<WebCore::Ar"...
__objc_const:3E7C3408                 DCD __WebArchivePrivate_coreArchive_+1
__objc_const:3E7C340C                 DCD 0x3507FFA8
__objc_const:3E7C3410                 DCD aV804               ; "v8@0:4"
__objc_const:3E7C3414                 DCD __WebArchivePrivate_.cxx_destruct_+1
__objc_const:3E7C3418                 DCD 0x3508088C
__objc_const:3E7C341C                 DCD a804                ; "@8@0:4"
__objc_const:3E7C3420                 DCD __WebArchivePrivate_.cxx_construct_+1
__objc_const:3E7C3424                 DCB 0x14
__objc_const:3E7C3425                 DCB    0
```

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

```
__objc_const:3E3593C6                 DCB    0
__objc_const:3E3593C7                 DCB    0
__objc_const:3E3593C8                 DCD 0xF
__objc_const:3E3593CC                 DCD 7
__objc_const:3E3593D0                 DCD aInitwithcorear     ; "initWithCoreArchive:"
__objc_const:3E3593D4                 DCD a1204Passrefptr     ; "@12@0:4{PassRefPtr<WebCore::LegacyWebAr"...
__objc_const:3E3593D8                 DCD __WebArchivePrivate_initWithCoreArchive__+1
__objc_const:3E3593DC                 DCD aSetcorearchive     ; "setCoreArchive:"
__objc_const:3E3593E0                 DCD aV1204Passrefpt     ; "v12@0:4{PassRefPtr<WebCore::LegacyWebAr"...
__objc_const:3E3593E4                 DCD __WebArchivePrivate_setCoreArchive__+1
__objc_const:3E3593E8                 DCD aCorearchive        ; "coreArchive"
__objc_const:3E3593EC                 DCD aLegacywebarchi     ; "^{LegacyWebArchive=i{RefPtr<WebCore::Ar"...
__objc_const:3E3593F0                 DCD __WebArchivePrivate_coreArchive_+1
__objc_const:3E3593F4                 DCD 0x33023574
__objc_const:3E3593F8                 DCD aV804               ; "v8@0:4"
__objc_const:3E3593FC                 DCD __WebArchivePrivate_dealloc_+1
__objc_const:3E359400                 DCD 0x330268FB
__objc_const:3E359404                 DCD a804                ; "@8@0:4"
__objc_const:3E359408                 DCD __WebArchivePrivate_init_+1
__objc_const:3E35940C                 DCD 0x335DEFA8
__objc_const:3E359410                 DCD aV804               ; "v8@0:4"
__objc_const:3E359414                 DCD __WebArchivePrivate_.cxx_destruct_+1
__objc_const:3E359418                 DCD 0x335DF88C
__objc_const:3E35941C                 DCD a804                ; "@8@0:4"
__objc_const:3E359420                 DCD __WebArchivePrivate_.cxx_construct_+1
__objc_const:3E359424                 DCB 0x14
__objc_const:3E359425                 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 IV

Apple's ASLR in iOS 4.3.x

How did ASLR end up in iOS?

- about three month into 2011 ASLR was discovered in the iOS 4.3 beta
- reason why it was introduced is unknown
- some believe it was introduced because Antid0te forced their hand
- but it is more likely that ASLR in Windows Phone 7 triggered it
- we will never know ...

Randomization in iOS 4.3

- jailbreakers with access to beta versions of iOS 4.3 posted crash dumps
- crash dumps revealed that
    - main binary load address is randomized
    - dyld load address is randomized
    - main binary and dyld are shifted by same offset (at execution time)
    - dyld_shared_cache load address is randomized (at boot time)

Randomization of Main Binary

- Applications are now compiled as position independent executables
- sets MH_PIE flag in mach-o header and adds relocation information
- no TEXT relocations therefore no problem with codesigning
- old applications cannot be randomized

➡ no magic, just using the features of mach-o that were already there

```
TestiPAD:~ root# ./test
Address Tester
Stack: 0x2fea0be0
Code:  0xa3e55
malloc_small: 0x1c8e7e00
malloc_large: 0xc1000
printf: 0x36735dd1
_dyld_get_image_header(0): 0xa2000

TestiPAD:~ root# ./test
Address Tester
Stack: 0x2fecbbe0
Code:  0xcee55
malloc_small: 0x1f861200
malloc_large: 0xec000
printf: 0x36735dd1
_dyld_get_image_header(0): 0xcd000
```

Randomization of Dyld

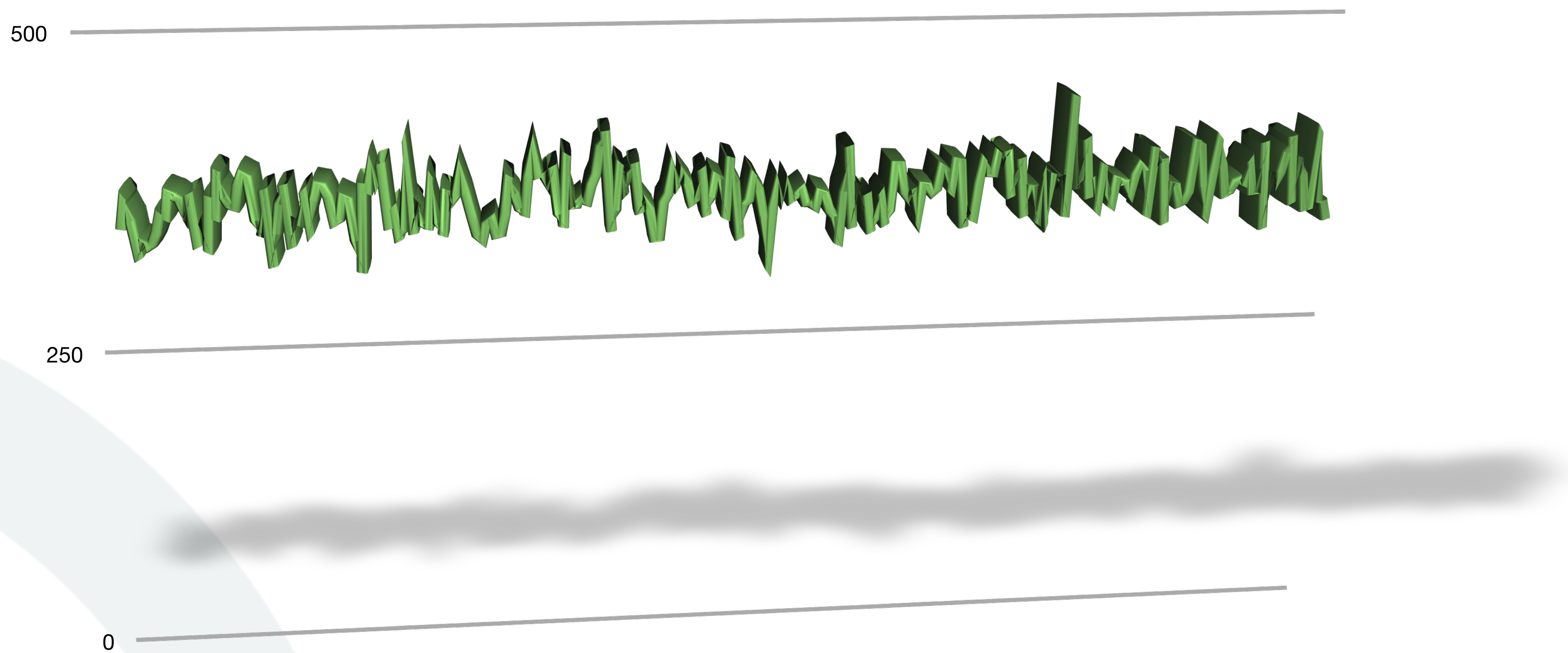
- dyld was already a PIE without TEXT relocs in older iOS versions
- even Antid0te could randomize it
- now randomization is done by the kernel on load
- however dyld is only slided the same amount as the main binary
- if main binary is not a PIE dyld is also not moved

```
Num Basename                Type Address        Reason | | Source
  | |                          | |                  | | | |
  1 test                       - 0x75000          exec Y Y /private/var/root/test at 0x75000 (offset 0x74000)
  2 dyld                       - 0x2fe74000       dyld Y Y /usr/lib/dyld at 0x2fe74000 (offset 0x74000) with ...

Num Basename                Type Address        Reason | | Source
  | |                          | |                  | | | |
  1 test                       - 0xc8000          exec Y Y /private/var/root/test at 0xc8000 (offset 0xc7000)
  2 dyld                       - 0x2fec7000       dyld Y Y /usr/lib/dyld at 0x2fec7000 (offset 0xc7000) with ...
```

How Random is the Baseaddress?

- randomized on page boundary
- only 256 possible base addresses between 0x1000 and 0x100000



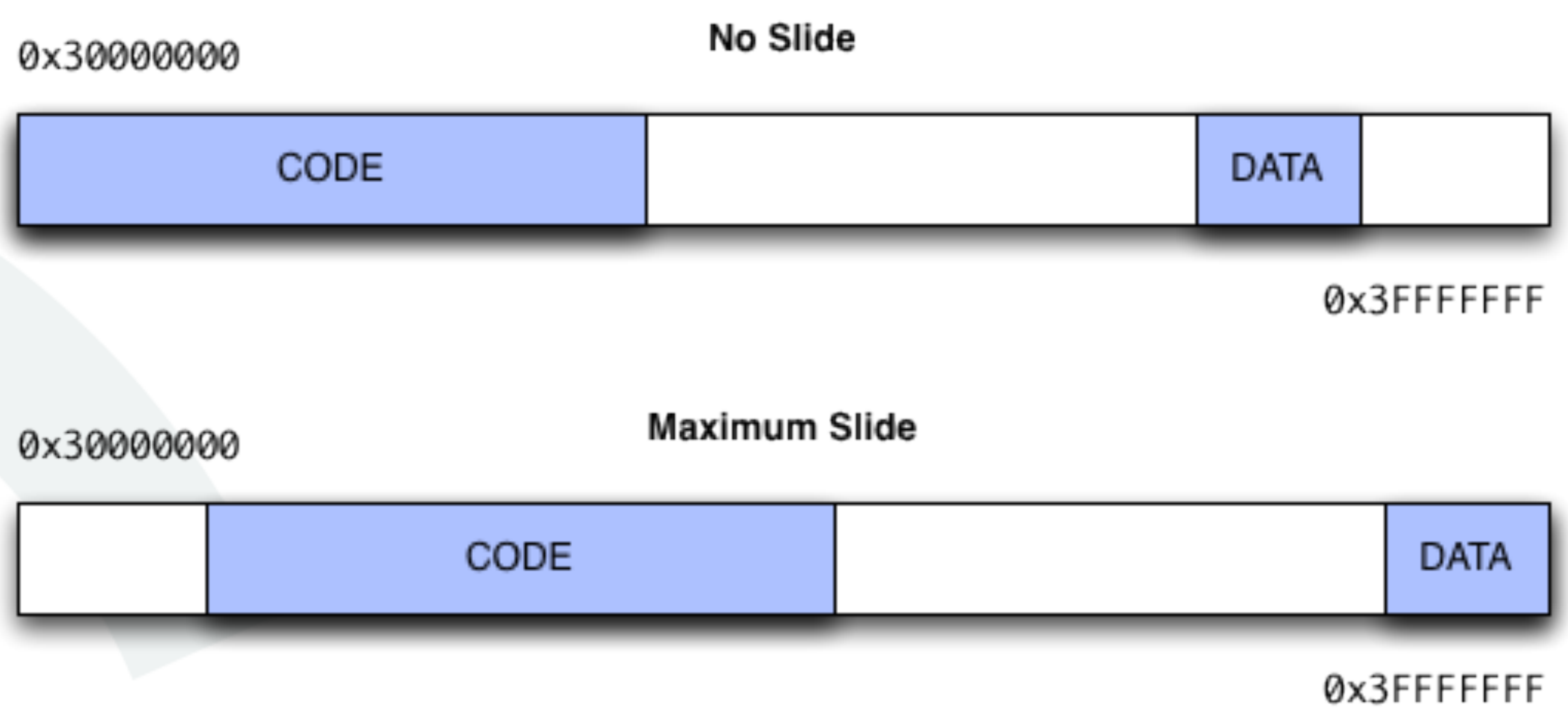
SektionEins

Randomization of dyld_shared_cache

- Sliding the dyld_shared_cache seems straight forward
- but Apple's implementation is complex and involves
    - randomization in dyld
    - a changed dyld_shared_cache file format
    - an undocumented relocation information format
    - a new syscall
    - a change in the memory page handling

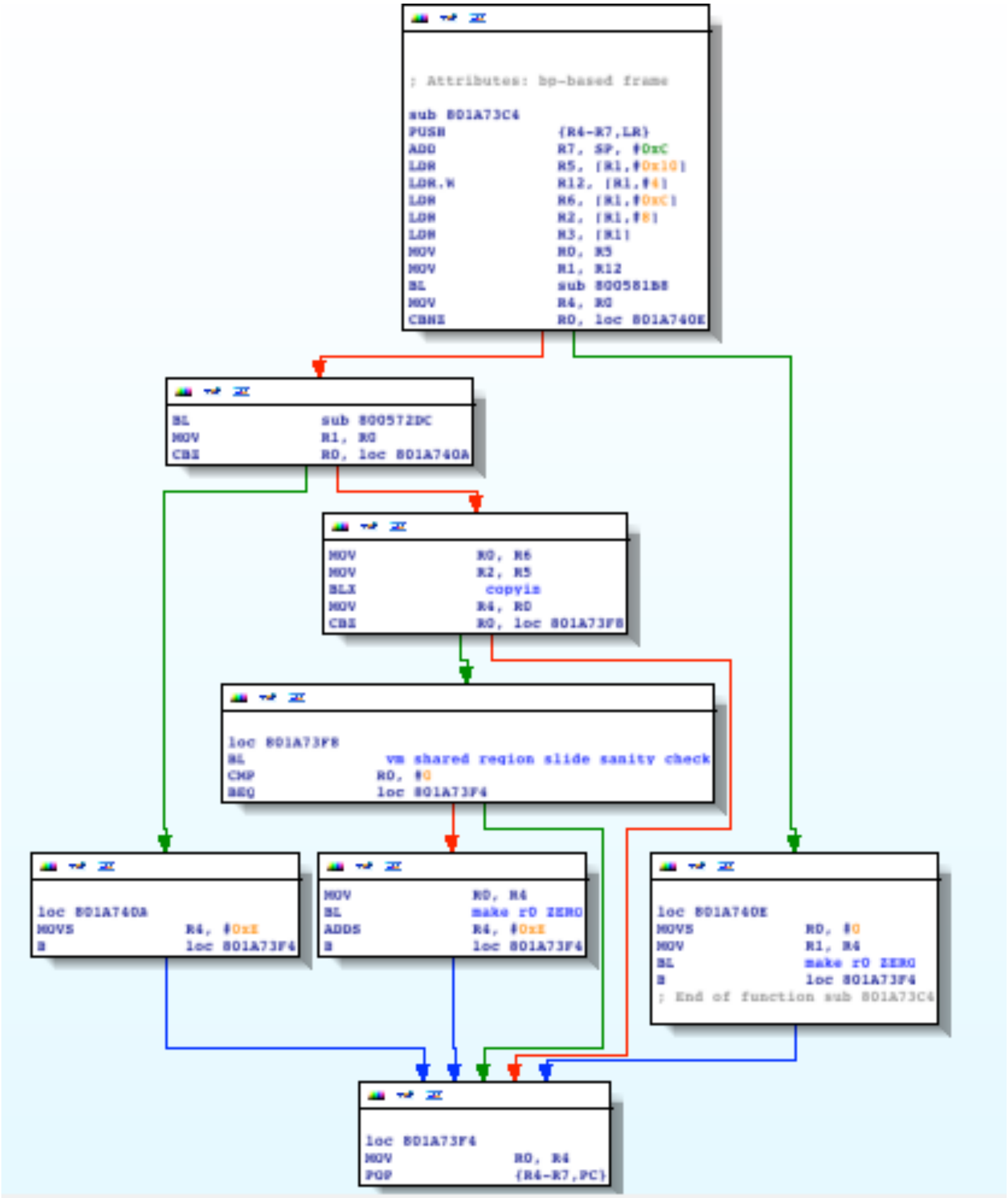
dyld_shared_cache sliding in dyld

- dyld has always been responsible for mapping the shared cache
- now it simply has to load it at a random address
- and tell the kernel about it (via new syscall)
- due to dyld_shared_cache structure only about 4200 different base addresses



New Syscall - vm_shared_region_slide

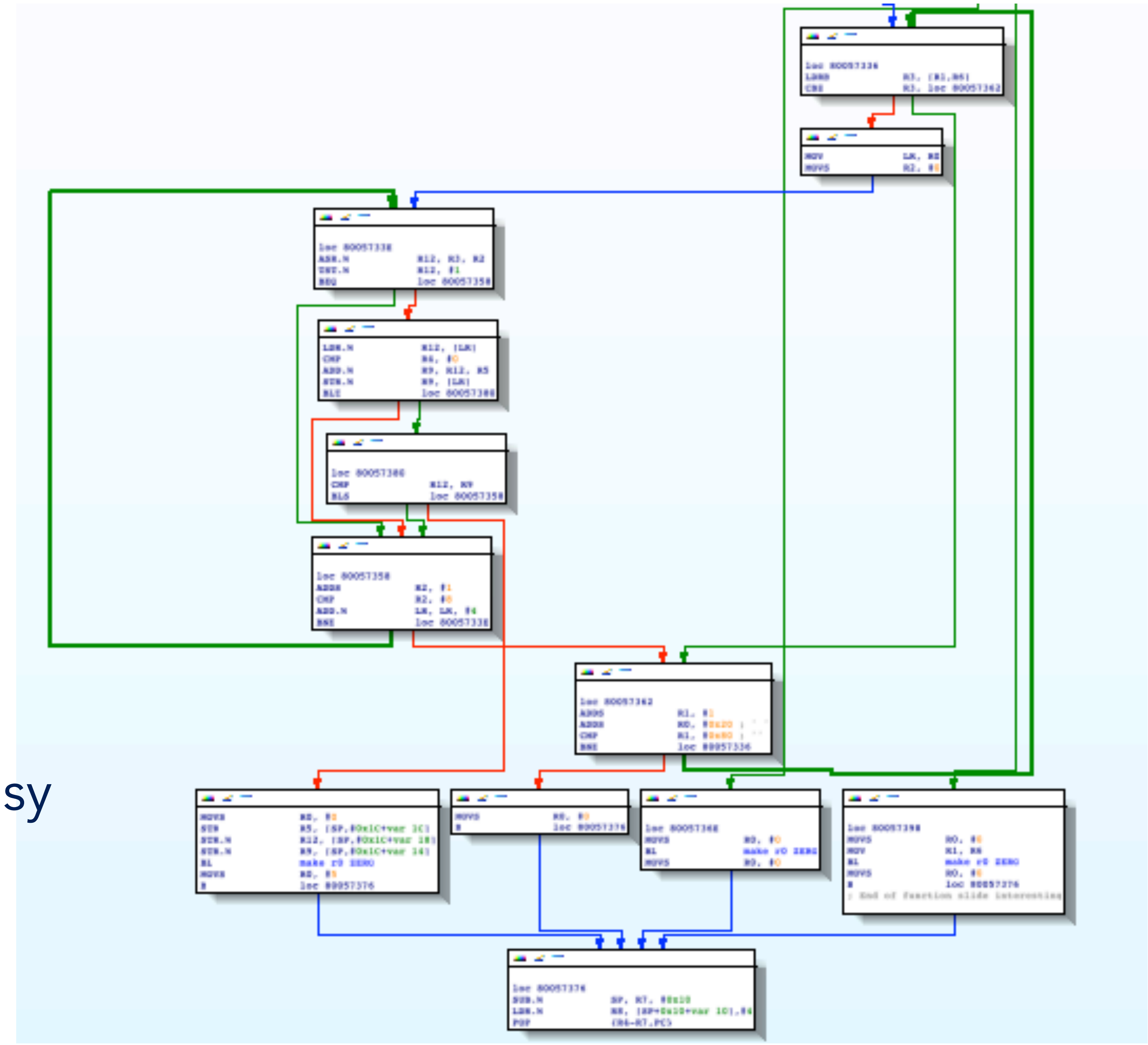
- iOS 4.3.x comes with a new syscall 437
- strings indicate that name is something like vm_shared_region_slide
- loads the dyld_shared_cache relocation information into kernel memory
- five parameters to this syscall
    1. slide delta
    2. address of region to slide
    3. size of region to slide
    4. address of reloc information
    5. size of reloc information



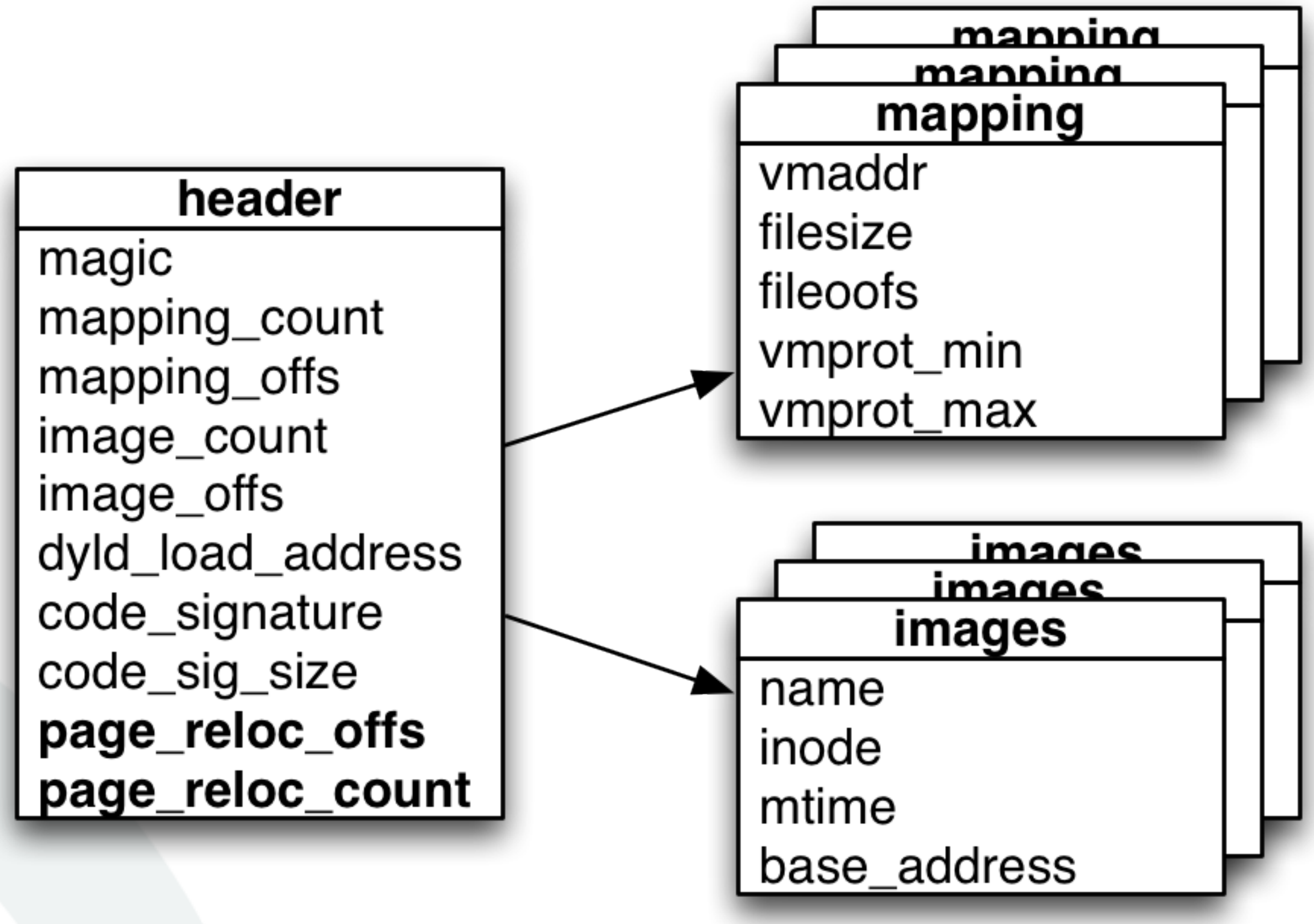
SektionEins

Changes in Memory Page Handling

- sliding whole cache is too slow
- Apple changed page handler to relocate each page on access
- works on the kernel buffer filled by syscall 437
- made decrypting the new dyld_shared_cache file format easy

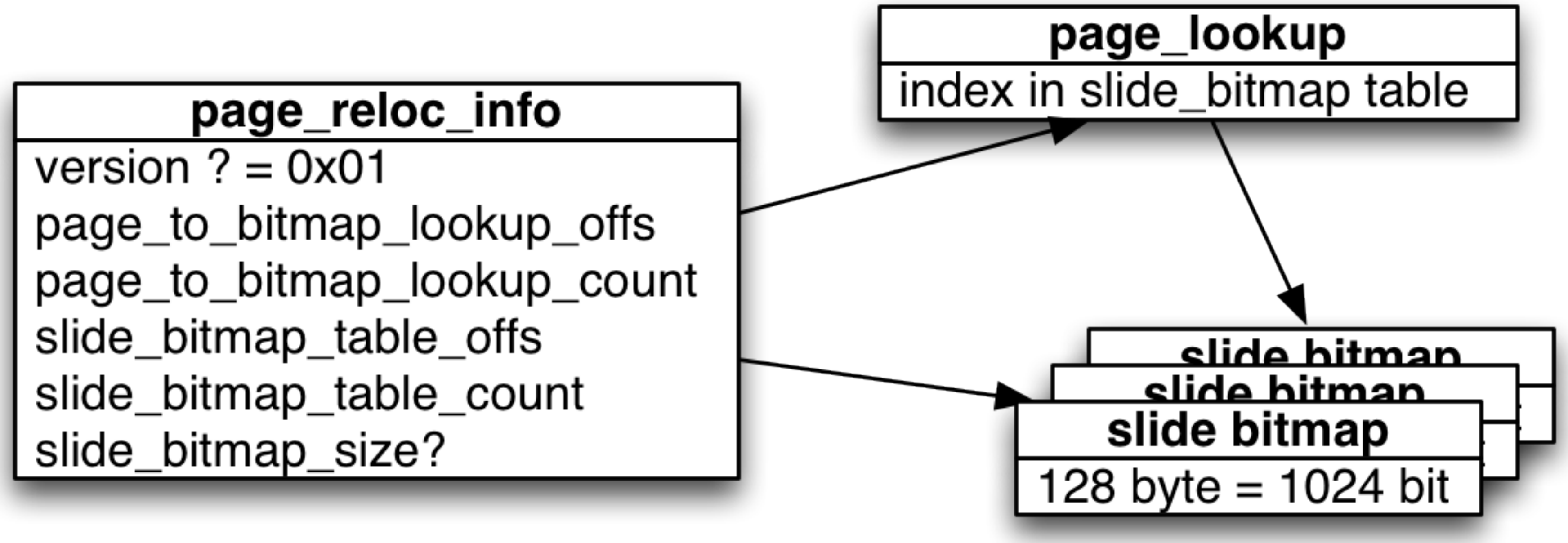


dyld_shared_cache Header in iOS 4.3.x



dyld_shared_cache relocation information

- relocation information is stored per page
- storage format 128 byte bitmap = 1024 bit
- each bit represents 4 aligned bytes
- if bit is set then add slide



SektionEins

Part V

Antid0te 2.0 ???

Antid0te 2.0 ???

- did iOS 4.3.x make Antid0te useless?
  - no, because iPhone 3G only runs up to 4.2.1
  - no, because iPhone 4 (CDMA) only runs 4.2.7 (feasibility not tested)
  - no, because Antid0te can extend the ASLR of iOS 4.3.x

What is different with iOS 4.3.x? (I)

- with iOS 4.3.x binaries come with relocation entries
- allows to select device specific base addresses for
  - main binary
  - dyld
- stack can still be randomized on the fly
- possible extensions
  - slide main binary and dyld separately
  - on the fly randomization with better randomness

What is different with iOS 4.3.x? (II)

- dyld_shared_cache comes also with relocation entries
- helps to partly verify the fixups detected by Antid0te
- but Antid0te still needs to detect relocations by diffing
    - no relocation entries for „delta“ access
    - objective c selector table needs to be detected and resorted
- relocation bitmap table entries need to be sorted

What is different with iOS 4.3.x? (III)

- kernel level changes make replacing the cache harder
- old on-the-fly method using DYLD environment variables just crashes
- for now tethered jailbreak with modified kernel is required
- crash problem might be solvable with patches to syscall 437 and dyld

➡ work in progress

Part VI

How Secure is ASLR on the iPhone

Why is the iPhone more Secure with ASLR

- targets are not respawning daemons
- attacks usually against non-respawning clients
- best target MobileSafari
- exploits are one shot
- not getting it right = ***crash***

```
Hardware Model:      iPad1,1
Process:         MobileSafari [302]
Path:            /Applications/MobileSafari.app/MobileSafari
Identifier:      MobileSafari
Version:         ??? (???)
Code Type:       ARM (Native)
Parent Process:  launchd [1]

Date/Time:       2011-05-19 01:03:18.012 +0200
OS Version:      iPhone OS 4.3.3 (8J3)
Report Version:  104

Exception Type:  EXC_BAD_ACCESS (SIGSEGV)
Exception Codes: KERN_INVALID_ADDRESS at 0x55555554
Crashed Thread:  0

Thread 0 name:  Dispatch queue: com.apple.main-thread
Thread 0 Crashed:
0   ???                                0x55555554 0 + 1431655764
1   WebCore                            0x32584d10 0x32519000 + 441616
2   WebCore                            0x32584c0c 0x32519000 + 441356
3   WebCore                            0x32584b08 0x32519000 + 441096
4   WebCore                            0x32582364 0x32519000 + 430948
5   WebCore                            0x3258499e 0x32519000 + 440734

...

Thread 0 crashed with ARM Thread State:
    r0: 0x2fedfed4    r1: 0x00000000      r2: 0x00000098      r3: 0x00000020
    r4: 0x0129bf54    r5: 0x55555555      r6: 0x2fedfed4      r7: 0x2fedfeb8
    r8: 0x00000001    r9: 0x01299000     r10: 0x55555555     r11: 0x2fee02a8
    ip: 0x32acc908    sp: 0x2fedec18      lr: 0x32584d15      pc: 0x55555554
  cpsr: 0x600f0030
```

Theoretical Limitations of ASLR on iPhone

- main binary, dynamic libs, dyld, heap and stack share 29bit address room
  - 0x00000000 - 0x2FFFFFFF
- single randomized page could be in $2^{29} - 2^{12} = 2^{17}$ = 131072 places

- address space for dyld_shared_cache is only 27bit wide
  - 0x30000000 - 0x37FFFFFF __TEXT
  - 0x38000000 - 0x3FFFFFFF __DATA
- single page can only be in $2^{27} - 2^{12} = 2^{15}$ = 32786 places

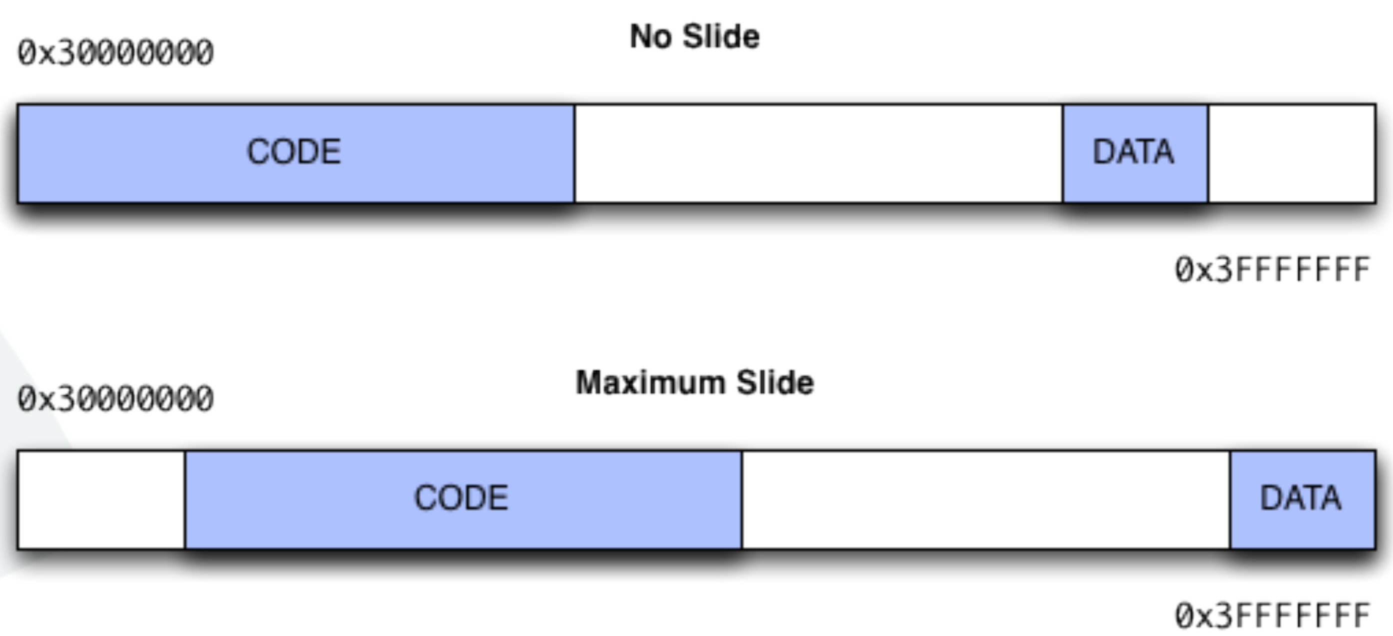
- ASLR implementations offer less randomization

Limitations of iOS 4.3.x ASLR (main binary/dyld)

- main binary and dyld slided same amount
- knowing address in one reveals addresses in the other
- only 256 possible base addresses
- stack always next to dyld base address
- if code segment is > 1 mb then page at 0x100000 is always readable

Limitations of iOS 4.3.x ASLR (dyld_shared_cache)

- whole dyld_shared_cache is slided as one block
- more than 100 mb of code can only be slided by 17 mb (about 4200 tries)
- large memory area is guaranteed to be readable
- order of libraries not randomized
- knowing the address of one symbol enough to know them all

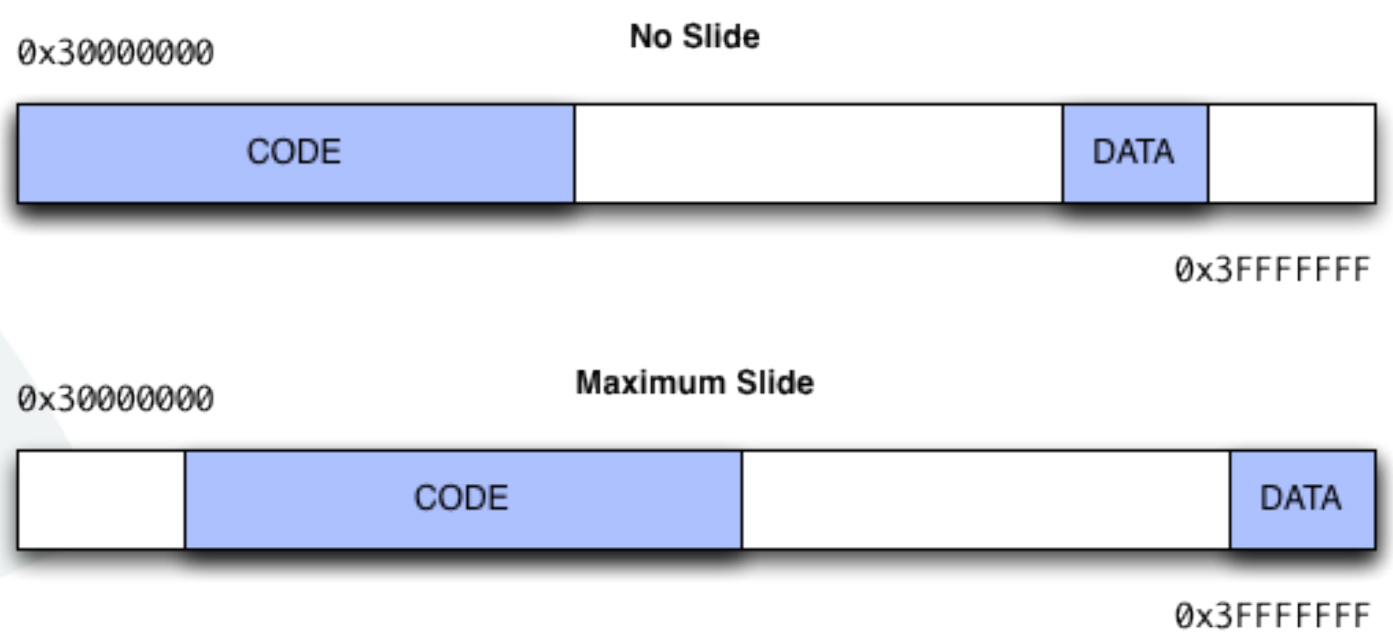


Limitations of Antid0te (main binary/dyld)

- only possible on jailbroken device
- standard base of main binary / dyld can be changed
- same limitations as iOS 4.3.x ASLR
- but base addresses different for every device

Limitations of Antid0te (dyld_shared_cache)

- does only work on jailbroken device (tethered for iOS 4.3.x)
- generating new caches only possible if comparison partners exists
- same sliding limitations as iOS 4.3.x but libraries are randomly shuffled
- extension could create unreadable memory gaps
- knowing the address of one symbol reveals addresses in same library



SektionEins

Final Words

- Antid0te 1.0 works perfect for iOS 4.2.1
- Antid0te 2.0 still work in progress for iOS 4.3.x
- expected release of Antid0te 2.0 in June *finally*
- more security tools for jailbroken iPhones soon (around BlackHat USA)

Questions ???

THE ELEVATOR

because the JailBreak community demanded to see it in action...

SektionEins