Antidote 2.0 - ASLR in iOS

Stefan Esser <stefan.esser@sektioneins.de>
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
• 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.N9OS/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
  -rw 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
dyld_shared_cache in detail

__TEXT
lib 1  lib 2  lib 3  lib 4  lib 5  lib 6  lib 7  lib 8  
lib 9  lib 10 lib 11 lib 12 lib 13 lib 14 lib 15 lib 16

__DATA
lib 10 lib 6  lib 15 lib 9  lib 5  lib 11 lib 8  lib 7  
lib 13 lib 3  lib 2  lib 12 lib 1 lib 14 lib 16 lib 4

__UNICODE

__LINKEDIT
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’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.

<table>
<thead>
<tr>
<th></th>
<th>iPhone 4</th>
<th>iPod 4</th>
<th>iPad</th>
</tr>
</thead>
<tbody>
<tr>
<td>inode</td>
<td>0x0933DE37</td>
<td>0x0933DE37</td>
<td>0x0933DE37</td>
</tr>
<tr>
<td>mtime</td>
<td>0x4CC1050A</td>
<td>0x4CC1050A</td>
<td>0x4CC1050A</td>
</tr>
<tr>
<td>base</td>
<td>0x33B5C000</td>
<td>0x31092000</td>
<td>0x30D03000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>iPhone 4</th>
<th>iPod 4</th>
<th>iPad</th>
</tr>
</thead>
<tbody>
<tr>
<td>inode</td>
<td>0x093AF2FC</td>
<td>0x093AF2FC</td>
<td>0x093AF2FC</td>
</tr>
<tr>
<td>mtime</td>
<td>0x4CC10998</td>
<td>0x4CC10998</td>
<td>0x4CC10998</td>
</tr>
<tr>
<td>base</td>
<td>0x33476000</td>
<td>0x33A03000</td>
<td>0x34A7D000</td>
</tr>
</tbody>
</table>
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
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
• 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
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
Reversing the objc differences

- tried to find the responsible code
- soon turned out to be more complicated
- source code matches only partially
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];
}

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

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

```c
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 relocations in older iOS versions
- even Antid0te could randomize it
- now randomization is done by the kernel on load
- however dyld is only slid by the same amount as the main binary
- if main binary is not a PIE dyld is also not moved

<table>
<thead>
<tr>
<th>Num</th>
<th>Basename</th>
<th>Type</th>
<th>Address</th>
<th>Reason</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>test</td>
<td>-</td>
<td>0x75000</td>
<td>exec Y Y /private/var/root/test at 0x75000 (offset 0x74000)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dyld</td>
<td>-</td>
<td>0x2fe74000</td>
<td>dyld Y Y /usr/lib/dyld at 0x2fe74000 (offset 0x74000) with ...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Num</th>
<th>Basename</th>
<th>Type</th>
<th>Address</th>
<th>Reason</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>test</td>
<td>-</td>
<td>0xc8000</td>
<td>exec Y Y /private/var/root/test at 0xc8000 (offset 0xc7000)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>dyld</td>
<td>-</td>
<td>0x2fec7000</td>
<td>dyld Y Y /usr/lib/dyld at 0x2fec7000 (offset 0xc7000) with ...</td>
<td></td>
</tr>
</tbody>
</table>
How Random is the Baseaddress?

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

- Sliding the dyld_shared_cache seems straightforward
- 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 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
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

- header
  - magic
  - mapping_count
  - mapping_offs
  - image_count
  - image_offs
  - dyld_load_address
  - code_signature
  - code_sig_size
  - page_reloc_offs
  - page_reloc_count

- mapping
  - vmaddr
  - filesize
  - fileofs
  - vmprot_min
  - vmprot_max

- images
  - name
  - inode
  - mtime
  - base_address
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
Part V

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 respawnning 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 0x55555555
Crashed Thread: 0
Thread 0 name: Dispatch queue: com.apple.main-thread
Thread 0 Crashed:
0   ???
 0x55555555 0 + 1431655764
1   WebCore
 0x32584d10 0x32519000 + 441616
2   WebCore
 0x32584c0c 0x32519000 + 441356
3   WebCore
 0x32584b08 0x32519000 + 441096
4   WebCore
 0x32582364 0x32519000 + 440734
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: 0x81299000  r10: 0x55555555  r11: 0x2f0e02a8
ip: 0x32acc908  sp: 0x2fedec18  lr: 0x32584d15  pc: 0x55555555
cpsr: 0x600f0030
Theoretical Limitations of ASLR on iPhone

- main binary, dynamic libs, dyld, heap and stack share 29bit address room
  - 0x00000000 - 0x2FFFFFFFF
- 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 slid 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
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...