SektionEins
http://www.sektioneins.de

iOS 6

Exploitation
280
Days Later

Stefan Esser <stefan.esser@sektioneins.de>

CAN SEC WEST

CanSecWest Vancouver

Who am I?

Stefan Esser

- from Cologne / Germany
- in information security since 1998
- initially did a lot of low level security
- from 2001 to 2010 focused on PHP / web app security
- since mid-2010 focused on iPhone security (ASLR, kernel exploitation)
- Head of Research and Development at SektionEins GmbH

What is this talk about?

- iOS 6 is the new major version of iOS with tons of new security features
- new kernel security mitigations already discussed by Mark Dowd/Tarjei Mandt
- but iOS 6.x has other not yet mentioned new security features
- and some kernel features require commentary
- basically an update to my CSW 2012 talk

- *280 days later because it was about 280 days later when I submitted to Dragos*

Part I

iOS Security Timeline 2012-2013

CanSecWest 2012 - iOS 5 An Exploitation Nightmare?

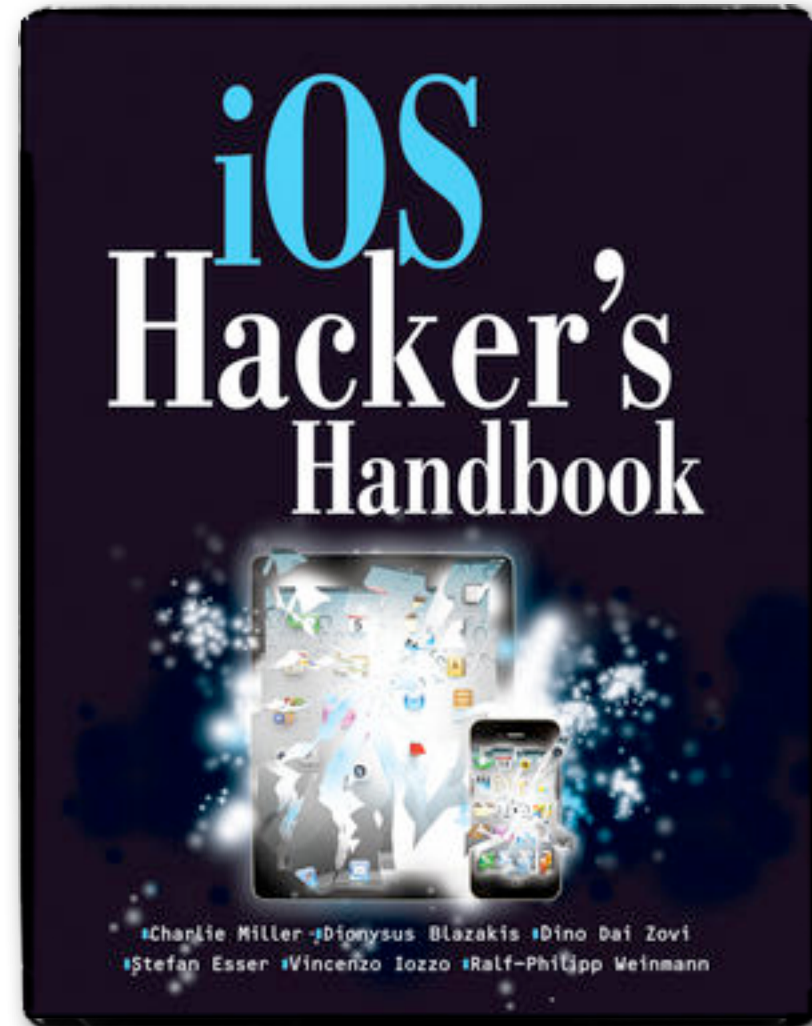
March 2012

- reasons why iOS 5 jailbreak took so long
- history of some iOS security features
- history of iOS security bugfixes
- getting kernel debugger running on new devices
- abusing BPF as kernel weird machine

URL: http://cansecwest.com/csw12/CSW2012_StefanEsser_iOS5_An_Exploitation_Nightmare_FINAL.pdf

iOS Hacker’s Handbook

April 2012



- **Charlie Miller - Dionysius Blazakis - Dino Dai Zovi**
- **Stefan Esser - Vincenzo Iozzo - Ralf-Philipp Weinmann**
- covers iOS 4 to iOS 5
- iOS Security Basics, iOS in the Enterprise
- Encryption, Code Signing and Memory Protection
- Sandboxing, Fuzzing iOS Applications
- Exploitation, Return-Oriented-Programming
- Kernel-Debugging and Exploitation, Jailbreaking, Baseband Attacks

URL: http://ca.wiley.com/WileyCDA/WileyTitle/productCd-1118204123.html

SyScan 2012 - iOS Kernel Heap Armageddon

April 2012

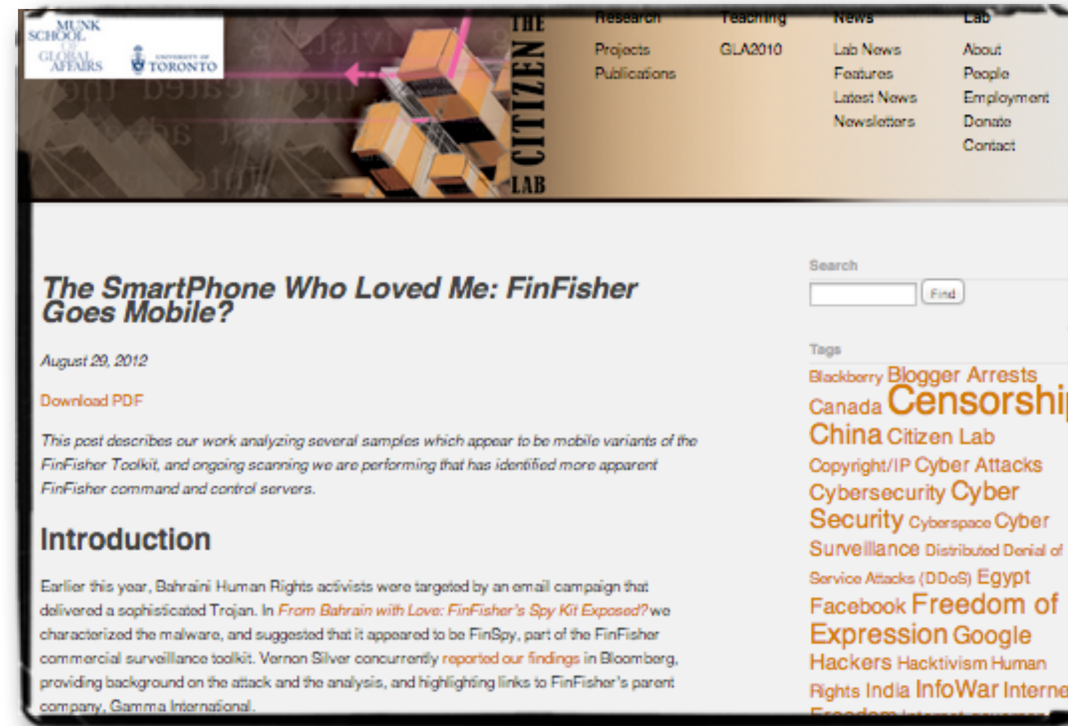
- different iOS kernel heap wrappers
- feasibility of cross zone / memory manager attacks
- attacking IOKit application data / object vtables instead of heap meta data
- using OSUnserializeXML() for generic kernel level heap feng shui
- talk updated for BlackHat USA & XCon 2012

URL 1: http://reverse.put.as/wp-content/uploads/2011/06/SyScan2012_StefanEsser_iOS_Kernel_Heap_Armageddon.pdf

URL 2: http://media.blackhat.com/bh-us-12/Briefings/Esser/BH_US_12_Esser_iOS_Kernel_Heap_Armageddon_WP.pdf

FinFisher Mobile - The Smartphone Who Loved Me

August 2012

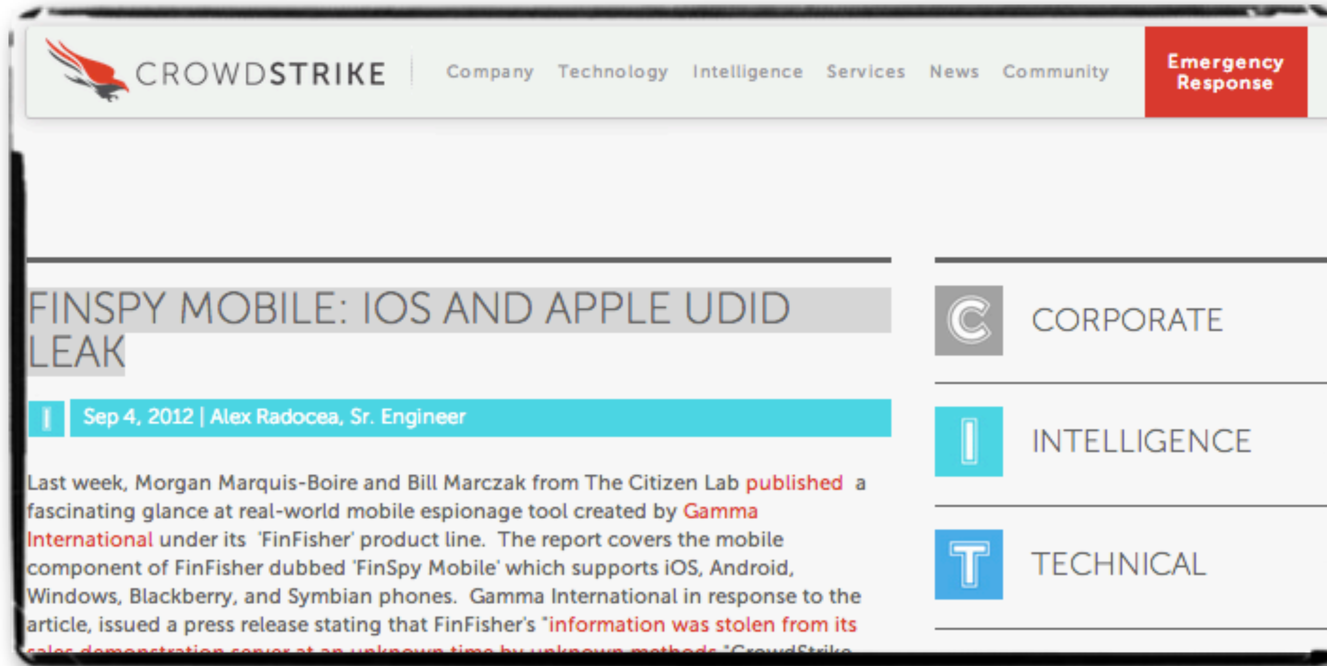


- **by CitizenLab**
- analysis of FinFisher for mobile devices
- samples caught in the wild
- iOS sample compiled for developer phones
- media wrongly assumed developer cert lets you write spy applications

URL: https://citizenlab.org/2012/08/the-smartphone-who-loved-me-finfisher-goes-mobile/

FinSpy Moile: iOS and Apple UDID Leak

September 2012



- **by Alex Radocea^Crowdstrike**
- deep analysis of FinFisher for iOS
- revealed that there was no iOS priv escape
  0-day in FinFisher iOS - just empty placeholder
- instead seems to heavily rely on being jailbroken
  with a public jailbreak prior to installation

URL: http://www.crowdstrike.com/blog/finspy-mobile-ios-and-apple-udid-leak/index.html

iOS 6 Released and J/"F"ailbroken on Day 1

September 2012

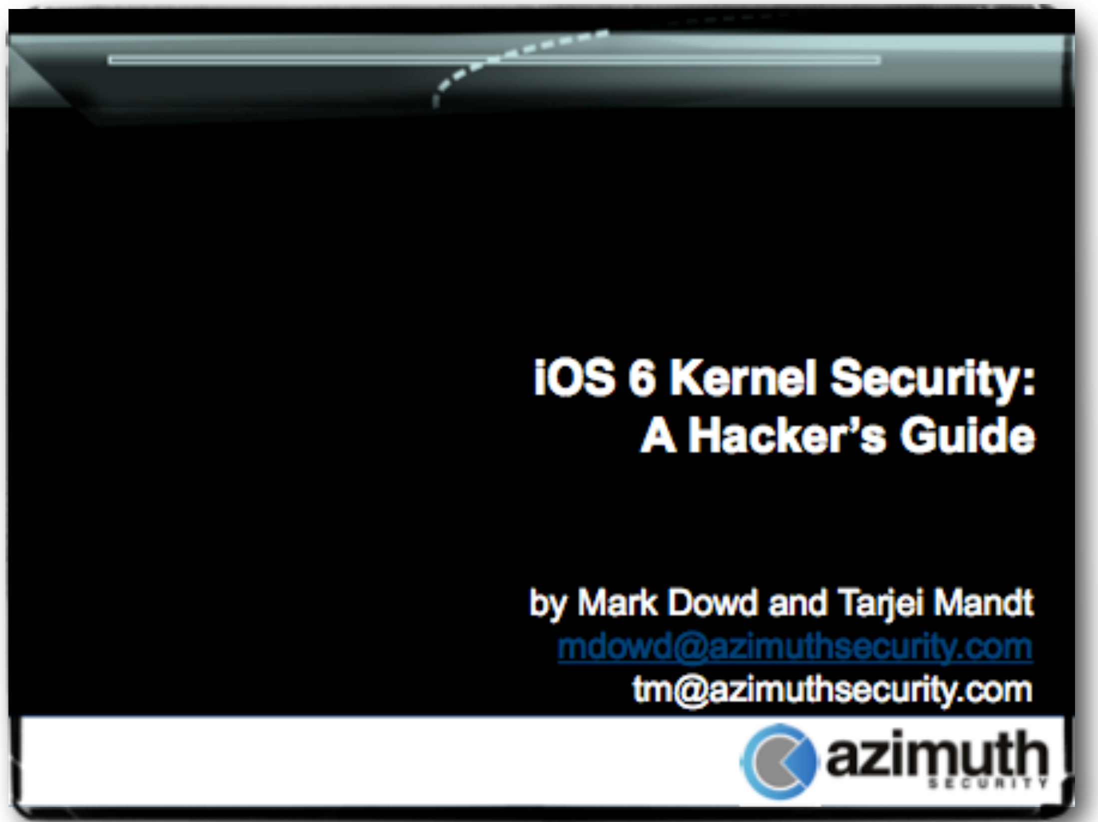
- **by Musclenerd**
- iOS 6 on pre-A5 already tethered jailbroken on day one

- **by CHPWN**
- iOS 6 on iPhone 5 already failbroken on day one
- failbroken means Cydia runs but no kernel payload

URL: https://twitter.com/chpwn/status/249249908094296064

HITB2012 - iOS 6 Kernel Security

October 2012



- **by Mark Dowd and Tarjei Mandt**
- deep analysis of new iOS 6 kernel exploit mitigations
- contained a 0-day kernel info leak vulnerability
- and the vm_map_copy exploitation technique heavily used by latest iOS 6 jailbreak

URL: http://conference.hackinthebox.org/hitbsecconf2012kul/materials/D1T2%20-%20Mark%20Dowd%20&%20Tarjei%20Mandt%20-%20iOS6%20Security.pdf

Video: http://www.youtube.com/watch?v=O-WZinEoki4

POC2012 - Find your own iOS kernel bug

November 2012

- **by Xu Hao and Chen Xiaobo**
- analysis of previous IOKit vulnerability
- about fuzzing iOKit for vulnerabilities

- later repeated at SyScan360 in December

URL: http://syscan.org/index.php/download/get/328bf4b37e6ae8b799472ff230465339/XuHao_Chen_Xiaobo_Find_your_own_iOS_kernel_bug.zip

Hackulo.us / Installous shutdown

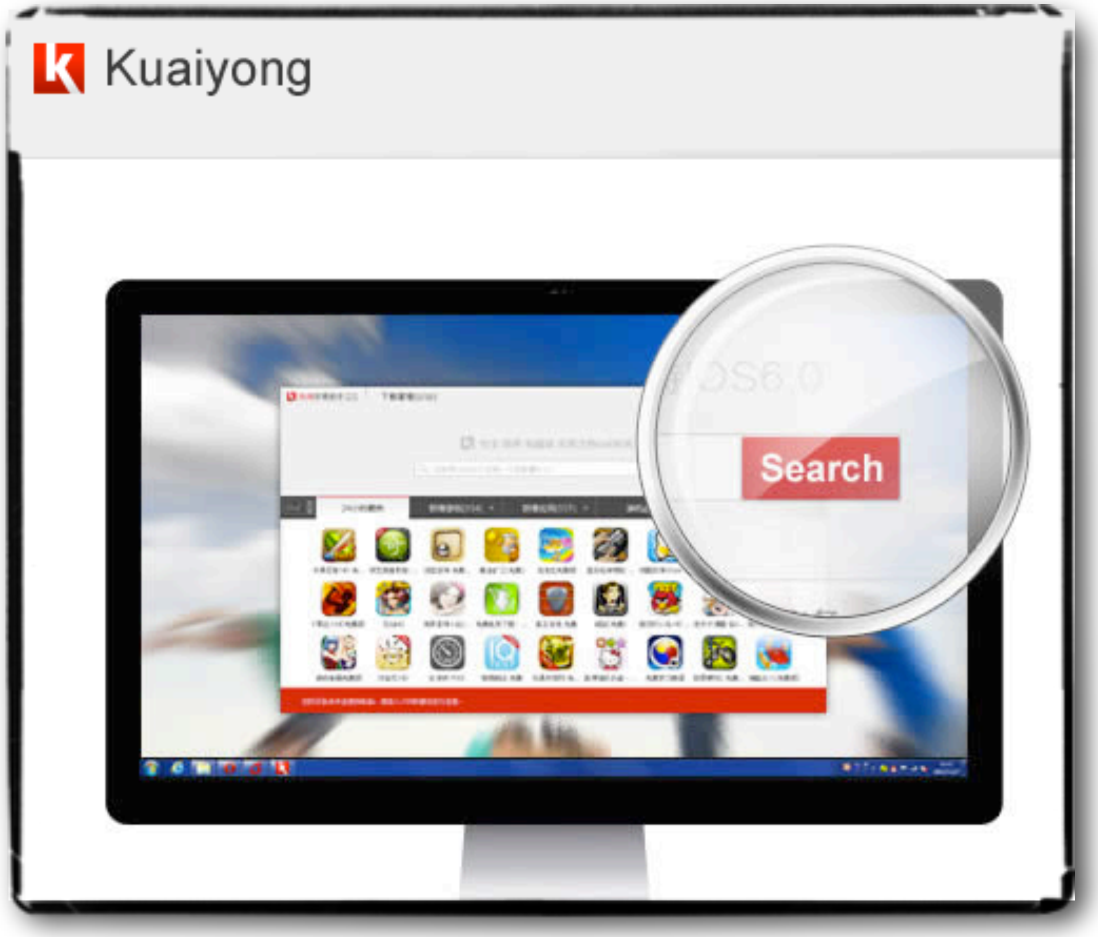
December 2012

- announcement that Hackulo.us shut down
- also took down Installous the notorious application used by iOS application pirates on jailbroken iPhones
- celebrated by media, jailbreak developers and iOS app developers around the world

URL: http://thanks-god-not-anymo.re

kuaiyong, Zeusmos, 25pp, ...

January 2013



- after installous is dead more and more iOS piracy solutions that do not require jailbreak
- solutions reportedly based on account sharing and/or some undisclosed exploit
- still active ?!?

URL 1: http://m.csoonline.com/article/725183/now-pirated-ios-apps-can-be-installed-without-jailbreak

URL 2: http://no.you.dont.get.the.url.you.want

Research Assistant: Marc Rogers

Community Milking and iOS 6 JB Release

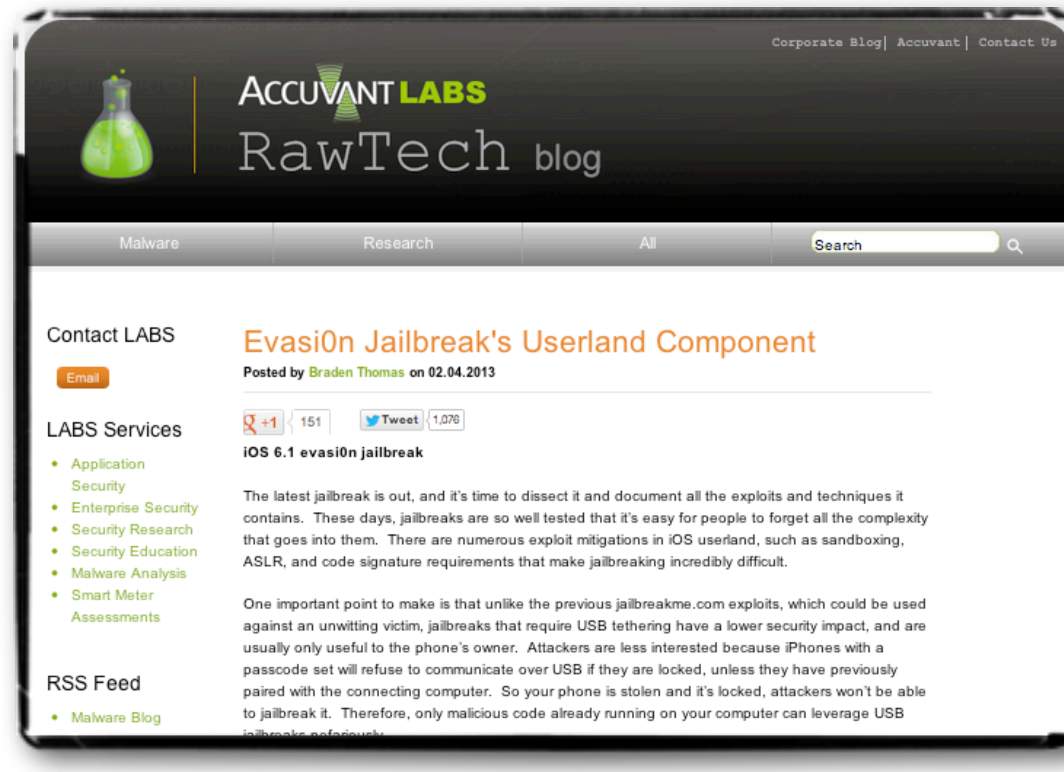
February 2013

- **by evad3rs**
- website with donation button and multiple banner ads
- told people repeatedly for about a week to check website for status updates
- about one week later release of iOS 6.0/6.1 jailbreak
- so far the most expensive jailbreak in terms of crowdfunding

URL: http://www.evasi0n.com/

evasi0n Jailbreak's Userland Components

February 2013

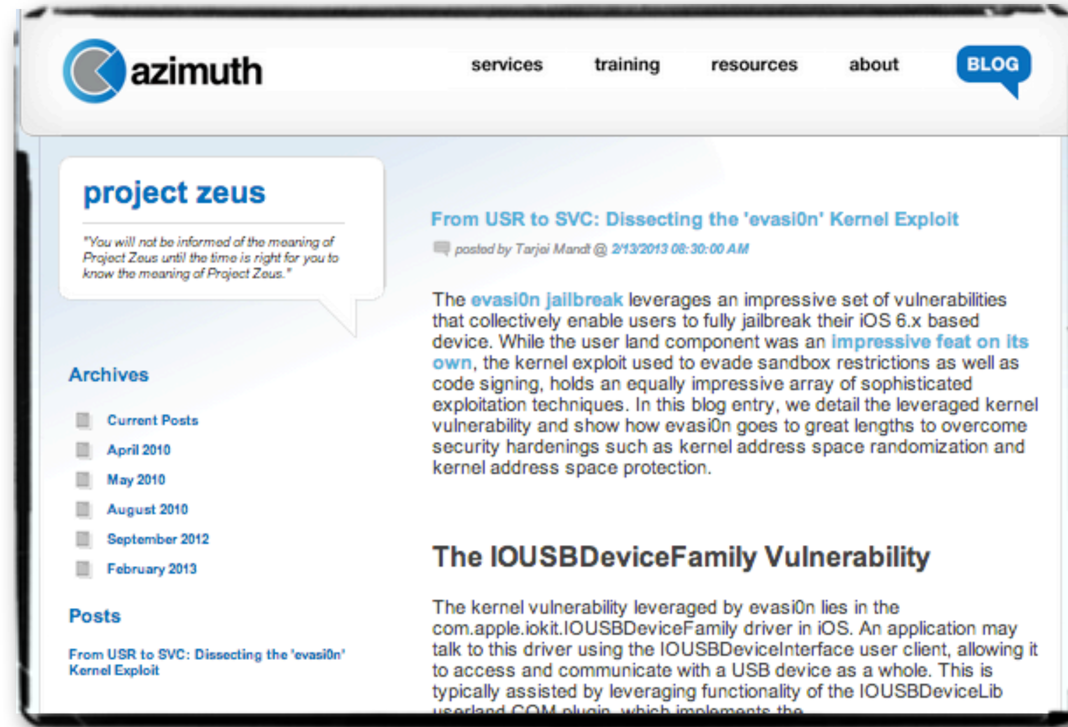


- **by Braden Thomas^AccuvantLabs**
- analysis of userland components of evasi0n jailbreak
- covers most of the userland bugs exploited by evasi0n

URL: http://blog.accuvantlabs.com/blog/bthomas/evasi0n-jailbreaks-userland-component

Dissecting the "evasi0n" Kernel Exploit

February 2013



- **by Tarjei Mandt^Azimuth**
- analysis of kernel components of evasi0n jailbreak
- shows how evasi0n is based on techniques discussed in the iOS 6 kernel security talk by azimuth

URL: http://blog.azimuthsecurity.com/2013/02/from-usr-to-svc-dissecting-evasi0n.html

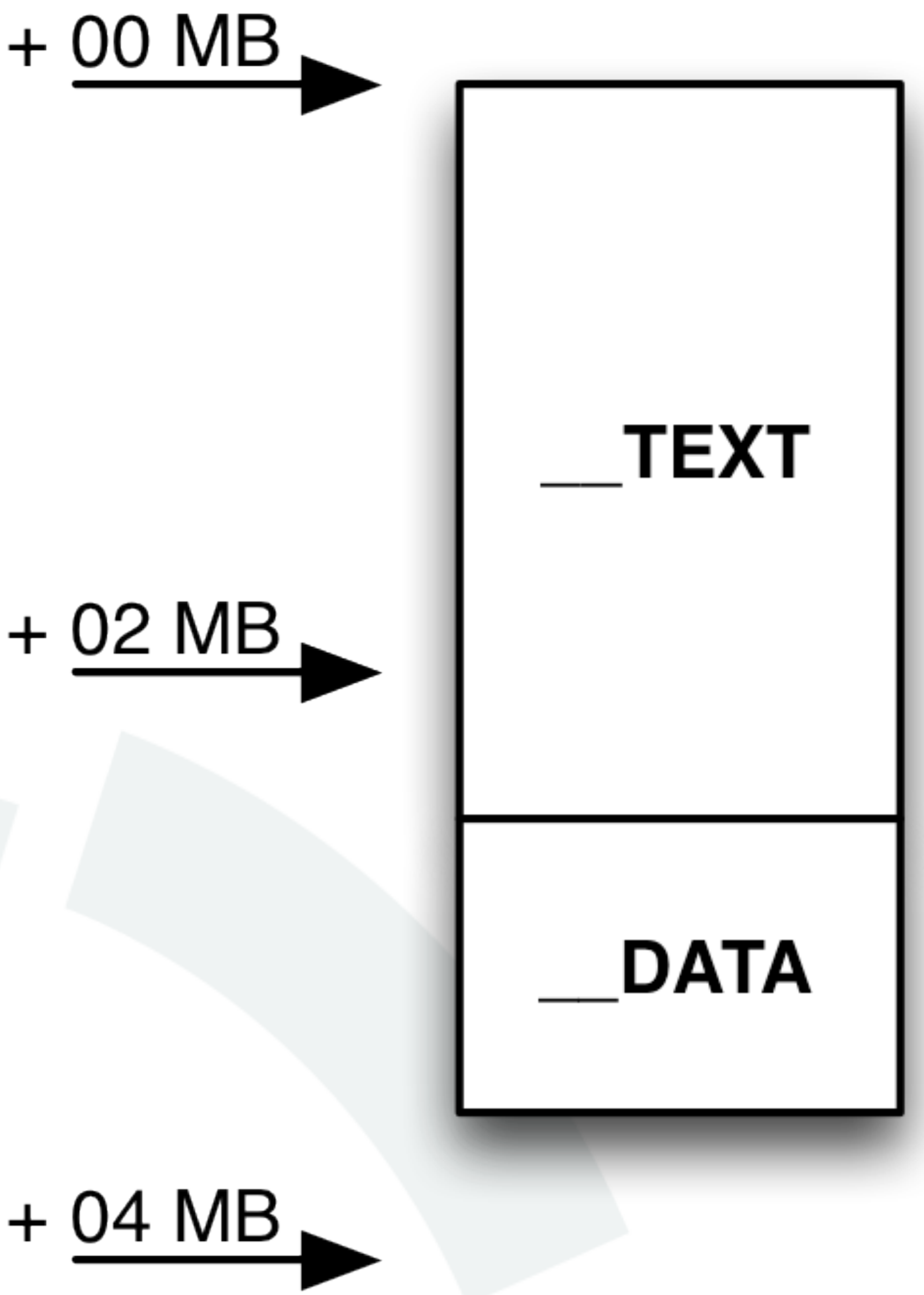
Part II

iOS 6 Kernel Security "Improvements"

KASLR

- iOS 6 introduces KASLR - kernel address space layout randomization
- only 256 possible load addresses
- each 2 MB apart
- starting at **`0x81200000`** ending at **`0xA1000000`**

KASLR: But why 2 MB Aligned?



- 2 MB alignment of KASLR seems arbitrary
- why not smaller alignment?
- big alignment is less secure
- right now:
  - leak any address in __DATA and you know the kernel's base address

    (address - 0x200000) & 0xFFE00000

  - leak any address from first 2 MB of kernel __TEXT and know the kernel's base address

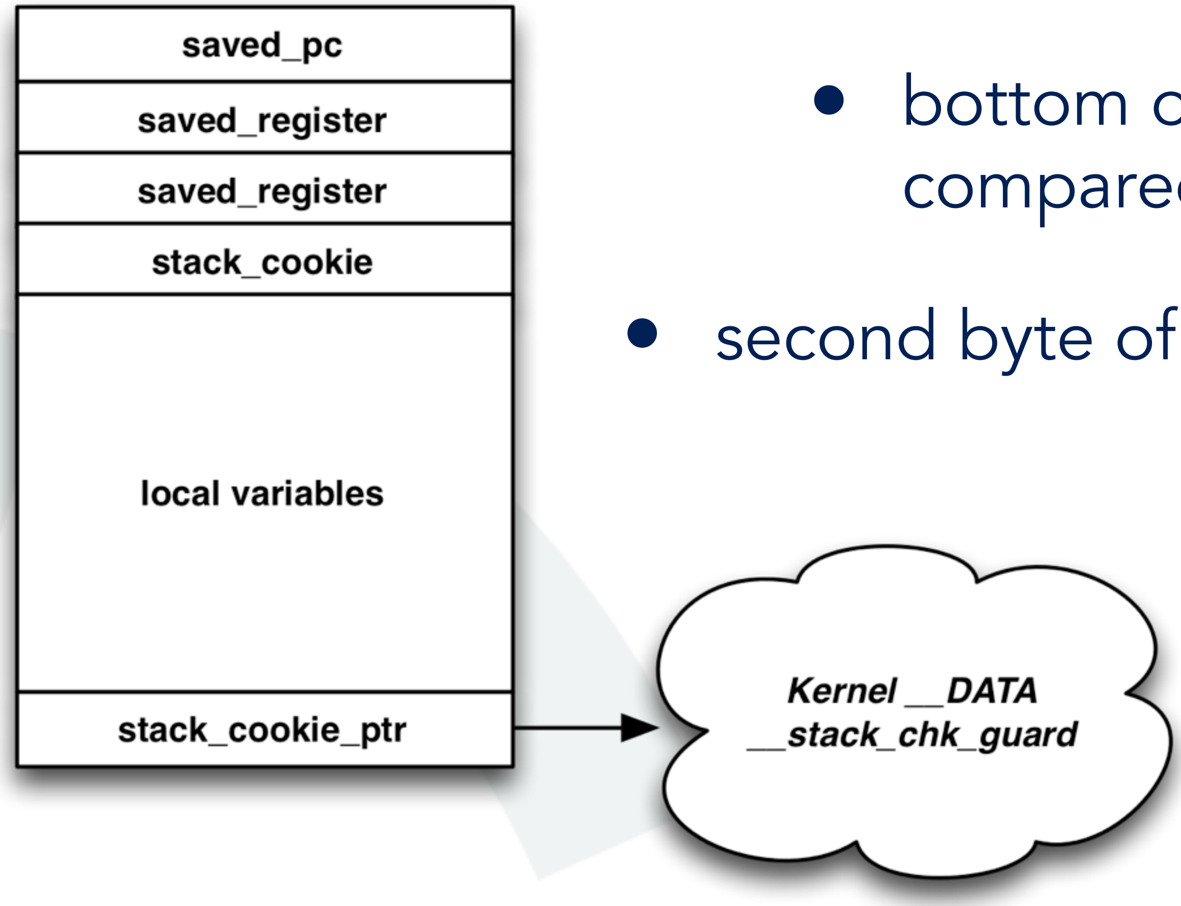
    address & 0xFFE00000

Kernel Address Space Hardening

- kernel __TEXT no longer writable
  - ➡ to stop kernel code hotpatching
- kernel heap no longer executable
  - ➡ to stop just executing kernel data
- kernel address space is separated from user space processes
  - ➡ to stop return into user space code and offset from NULL-deref attacks

Kernel Stack Cookies

- iOS 6 added stack cookies to protect from kernel stack buffer overflows
- implementation is rather unusual
  - stack cookie on top of stack
  - bottom of local stack contains ptr to the value it is compared against
- second byte of stack cookie is forced to **`0x00`**



Kernel Stack Cookie Verification

- stack cookie verification in function epilog
- verification against cookie pointed to
- fact that **`stack_cookie_ptr`** and **`stack_cookie`** are both on stack is a weakness
- wrong cookie value will lead to a kernel panic without message

```
__text:8027AFB0         LDR         R0, [SP,#0x4C+stack_cookie_ptr]
__text:8027AFB2         LDR         R0, [R0]
__text:8027AFB4         LDR         R1, [SP,#0x4C+stack_cookie]
__text:8027AFB6         CMP         R0, R1
__text:8027AFB8         ITTT EQ
__text:8027AFBA         ADDEQ       SP, SP, #0x34
__text:8027AFBC         POPEQ.W     {R8,R10,R11}
__text:8027AFC0         POPEQ       {R4-R7,PC}
__text:8027AFC2         BL          ___stack_chk_fail
```

Kernel Heap Cookies

- iOS 4 and iOS 5 kernel heap exploitation has always attacked the free list
- in iOS 6 Apple introduced heap protection cookies to protect free list
- distinguishes between small poisoned and larger non-poisoned blocks
- two different security cookies are used for this

➡ stops attacks against the free list as used before in public jailbreaks

Kernel Heap Cookies (larger blocks)

- for larger blocks the memory content is kept but end is trashed with cookie
- secret cookie has lowest bit cleared
- if data of freed block leaks this leaks
  - a kernel heap address: **0x87b46500**
  - the secret cookie: **0x6b7769c8 ^ 0x87b46500 = 0xECC30CC8**

next_pointer

```
87b46480: 00 65 b4 87 00 00 00 00 00 00 00 00 00 00 00 00 .e..............
87b46490: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
87b464a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
87b464b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
87b464c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
87b464d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
87b464e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
87b464f0: 00 00 00 00 00 00 00 00 00 00 00 00 c8 69 77 6b .............iwk
```

next_pointer^non_poisoned_cookie

Kernel Heap Cookies (small blocks)

- for small blocks the memory content is overwritten with **`0xdeadbeef`**
- secret cookie has lowest bit set
- if data of freed block leaks this leaks
  - a kernel heap address: **`0x92f1c740`**
  - the secret cookie: **`0x7ec1387b ^ 0x92f1c740 = 0xEC30FF3B`**

next_pointer

```
92f1c700: 40 c7 f1 92 ef be ad de ef be ad de ef be ad de @...............
92f1c710: ef be ad de ef be ad de ef be ad de ef be ad de ................
92f1c720: ef be ad de ef be ad de ef be ad de ef be ad de ................
92f1c730: ef be ad de ef be ad de ef be ad de 7b 38 c1 7e ............{8.~
```

next_pointer^poisoned_cookie

Kernel Heap Cookies after allocation

- on allocation free list pointer and cookie are overwritten with **`0xdeadbeef`**
- most probably as defense in depth against information leaks

```
9072b000: ef be ad de 00 00 00 ff 00 00 00 ff 00 00 00 ff ................
9072b010: 00 00 00 ff 00 00 00 ff 00 00 00 ff 00 00 00 ff ................
9072b020: 00 00 00 ff 00 00 00 ff 00 00 00 ff 00 00 00 ff ................
9072b030: 00 00 00 ff 00 00 00 ff 00 00 00 ff 00 00 00 ff ................
9072b040: 00 00 00 ff 00 00 00 ff 00 00 00 ff 00 00 00 ff ................
9072b050: 00 00 00 ff 00 00 00 ff 00 00 00 ff 00 00 00 ff ................
9072b060: 00 00 00 ff 00 00 00 ff 00 00 00 ff 00 00 00 ff ................
9072b070: 00 00 00 ff 00 00 00 ff 00 00 00 ff ef be ad de ................
```

Kernel Heap Hardening

- previously **mach_zone_info()** and **host_zone_info()** leaked internal state
- both functions now require debugging kernel boot arguments

- previously **OSUnserializeXML()** allowed fine control over kernel heap
- Apple fixed some bugs in it and put some arbitrary limits on it
- only exact methods described at BlackHat / SyScan were killed
- **other ways to abuse this function for kernel heap feng shui still working**

Death to Kernel Info Leaks

- two fold strategy to fight kernel info leaks
  - fix information leak vulnerabilities
  - obfuscate kernel addresses returned to user land

- example of fixed information leaks
  - **BPF** stack data info leak
  - **kern.proc** leak fixed
  - **kern.file** info leak fixed

Kernel Address Obfuscation

- lots of kernel API return kernel addresses to user land processes

  e.g. mach_port_kobject(), mach_port_space_info(), vm_region_recurse(), vm_map_region_recurse(), vm_map_page_info(), proc_info(), fstat(), sysctl()

- protected by adding a random 32 bit cookie *(lowest bit set)*

```
#define VM_KERNEL_ADDRPERM(_v)                          \
        (((vm_offset_t)(_v) == 0) ?                     \
            (vm_offset_t)(0) :                          \
            (vm_offset_t)(_v) + vm_kernel_addrperm)
```

```
iin->iin_urefs = IE_BITS_UREFS(bits);
iin->iin_object = (natural_t)VM_KERNEL_ADDRPERM((uintptr_t)entry->ie_object);
iin->iin_next = entry->ie_next;
iin->iin_hash = entry->ie_index;
```

Kernel Image Address Obfuscation

- some API might even return addresses inside the kernel image
- these addresses are additionally **unslid** to protect against **KASLR** leaks

```
#define VM_KERNEL_UNSLIDE(_v)                                   \
        ((VM_KERNEL_IS_SLID(_v) ||                          \
          VM_KERNEL_IS_KEXT(_v)) ?                          \
             (vm_offset_t)(_v) - vm_kernel_slide :                  \
             (vm_offset_t)(_v))
#define VM_KERNEL_SLIDE(_u)                                 \
        ((vm_offset_t)(_u) + vm_kernel_slide)

#define VM_KERNEL_ADDRPERM(_v)                               \
        (((vm_offset_t)(_v) == 0) ?                     \
             (vm_offset_t)(0) :                     \
             (vm_offset_t)(_v) + vm_kernel_addrperm)
```

```
if (0 != kaddr && is_ipc_kobject(*typep))
    *addrp = VM_KERNEL_ADDRPERM(VM_KERNEL_UNSLIDE(kaddr));
else
    *addrp = 0;
```

Readonly Syscall Table

- previous jailbreaks used partial syscall table overwrites
- Apple moved syscall table into section `__DATA::__const`
- section is made read only at runtime
- controlled by kernel boot argument `dataconstro`
- stops syscall table corruption ...

Just replace Syscall Table completely?

- kernel linking changes in iOS 6 introduced lots of indirect accesses
- syscall table is no longer accessed directly (also true for lots of other stuff)
- instead pointer to syscall table is used from **`__nl_symbol_ptr`** section
- and guess what - this section is writable

```
text:8021F760   LDR     R10, [R0,#0x30]
text:8021F764   CMP     R10, #0
text:8021F768   LDREQ   R10, [R0]
text:8021F76C   MOV     R2, #(_pNsys - 0x8021F77C) ; _pNsys
text:8021F774   LDR     R2, [PC,R2] ; _pNsys
text:8021F778   MOV     R1, #(_pSysent - 0x8021F78C) ; _pSysent
text:8021F780   UXTH    R5, R10
text:8021F784   LDR     R1, [PC,R1] ; _pSysent
text:8021F788   LDR     R2, [R2]
text:8021F78C   CMP     R5, R2
text:8021F790   BLT     loc_8021F7A0
```

```
nl_symbol_ptr:802D2C7C _pNsys      DCD _nsys
nl_symbol_ptr:802D2C7C
nl_symbol_ptr:802D2C80 _pSysent    DCD _sysent
```

Part III

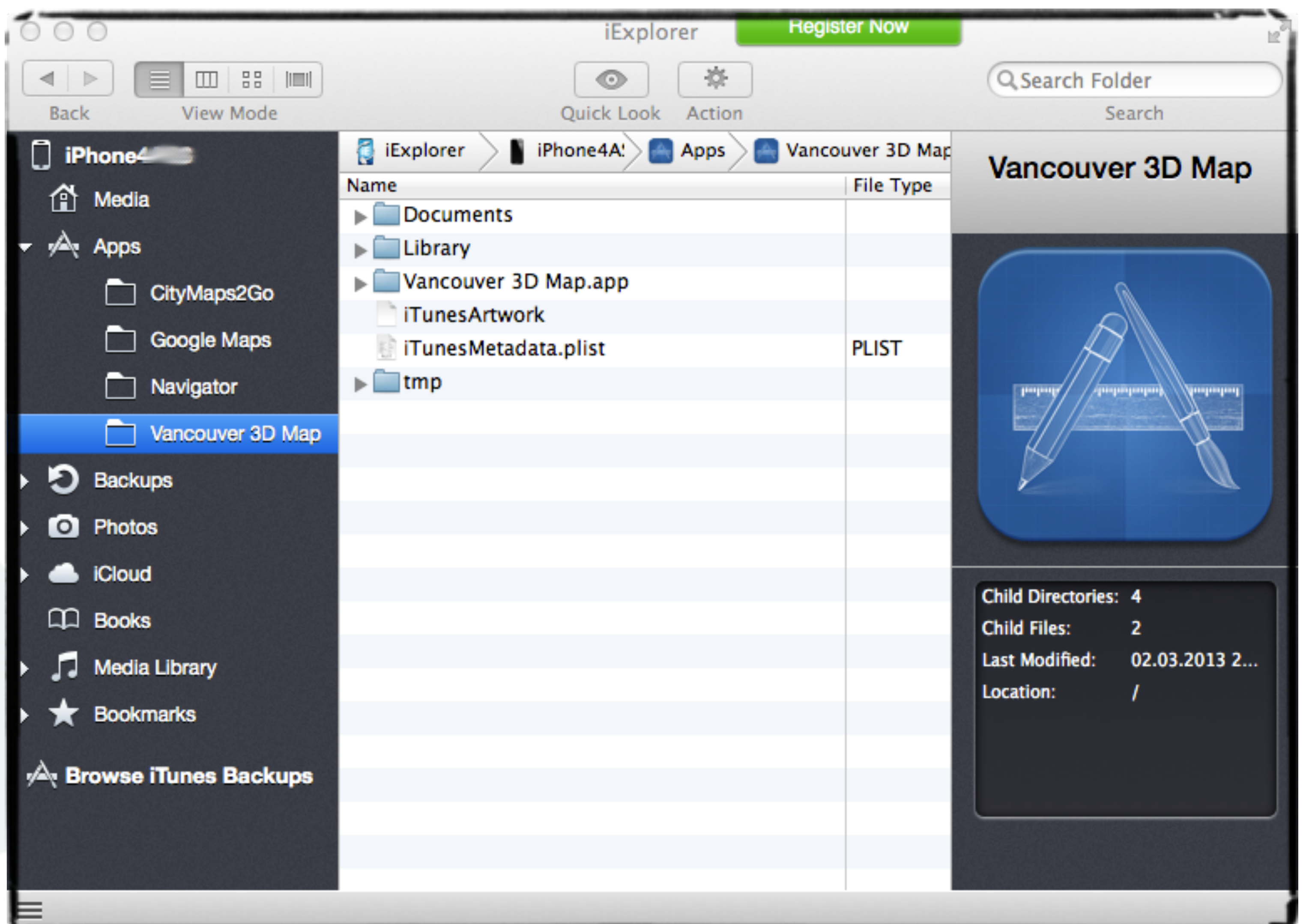
iOS 6 Misc Hardening

BPF not so weird anymore...

- at CSW 2012 BPF was mentioned as weird machine inside the kernel
- in iOS 6.x it is still a machine but not so weird anymore
- Apple added sanity checks inside the function
- access to slack memory is now checked for bounds

mobile_house_arrest - Readonly Code Directory

- lockdown service for reading / writing into app directories
- since iOS 6 application's code directory is no longer writable
- previously it was possible to replace arbitrary application resources



Part IV

User Space ASLR (Address Space Layout Randomization)

ASLR in iOS 4.3-6.x

- randomly slides
    - main binary
    - dyld *(dynamic linker)*
    - dynamic library cache

Position Independent Executables in 2012

```
$ python ipapiescan.py
Adobe Reader                    -       armv7 - PIE    - N/A
Bluefire Reader                 - armv6|armv7 - NO_PIE - 3.0
DiamondDash                     -       armv7 - NO_PIE - 4.2
Ebook Reader                    - armv6|armv7 - NO_PIE - N/A
eBookS Reader                   - armv6|armv7 - NO_PIE - N/A
Facebook                        - armv6|armv7 - NO_PIE - 4.0
Fly With Me                     - armv6|armv7 - NO_PIE - 3.0
FPK Reader                      - armv6|armv7 - NO_PIE - 3.2
Hotels                          - armv6|armv7 - NO_PIE - 3.1
iBooks                          - armv6|armv7 - NO_PIE - 4.2
KakaoTalk                       - armv6|armv7 - NO_PIE - 3.1
Messenger                       - armv6|armv7 - NO_PIE - 4.0
PerfectReader Mini              - armv6|armv7 - NO_PIE - N/A
QR Reader                       - armv6|armv7 - NO_PIE - 4.0
QR Scanner                      - armv6|armv7 - NO_PIE - N/A
QR-Scanner                      -       armv7 - NO_PIE - 4.0
QRCode                          - armv6|armv7 - NO_PIE - N/A
Quick Scan                      - armv6|armv7 - NO_PIE - 4.0
Skype                           - armv6|armv7 - NO_PIE - N/A
Twitter                         - armv6|armv7 - NO_PIE - 4.0
vBookz PDF                      -       armv7 - PIE    - 4.3
VZ-Netzwerke                    - armv6       - NO_PIE - 3.0
Wallpapers                      - armv6|armv7 - NO_PIE - 4.1
WhatsApp                        - armv6|armv7 - NO_PIE - 3.1
Where is                        - armv6|armv7 - NO_PIE - 4.1
```

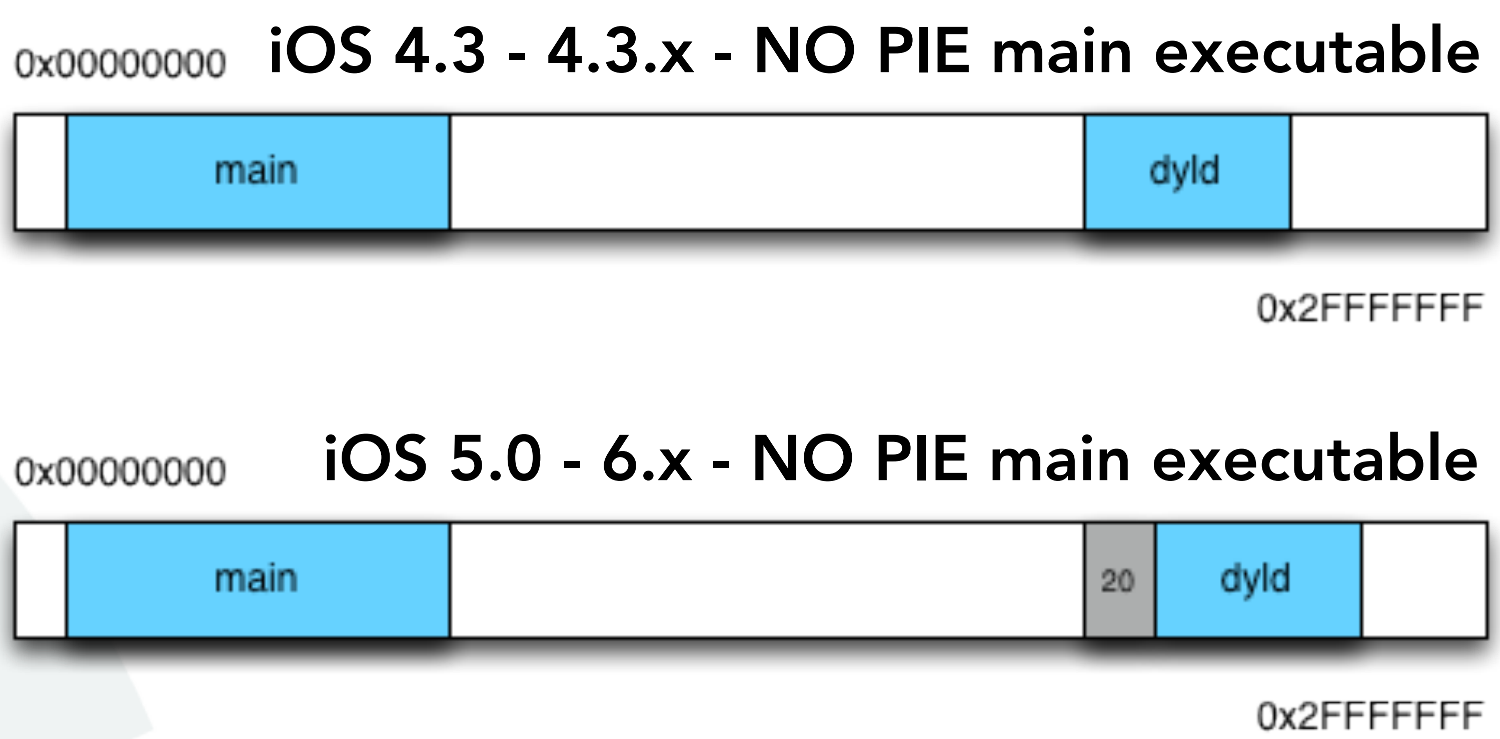
- all system binaries were compiled as PIE
- most 3rd party apps were not compiled as PIE

source code of old **`idapiescan.py`** is available at Github

https://github.com/stefanesser/idapiescan

iOS 4.3-6.x: NO PIE main binary randomization

- dynamic loader is not slid in iOS 4 for NO PIE main executables
- since iOS 5 the dynamic loader is always slid
- randomized by kernel in 256 positions



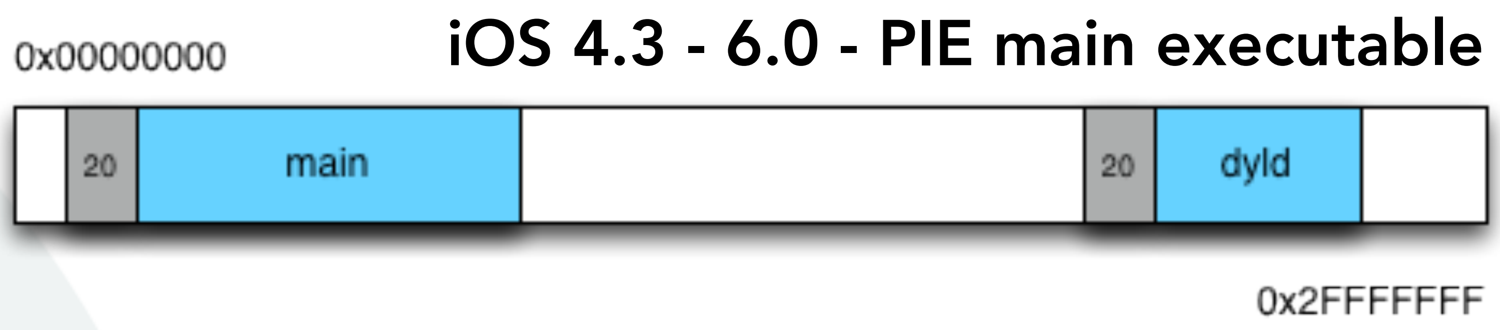
Position Independent Executables in 2013

```
$ python ipapiescan.py
Bluefire Reader      -        armv7(s) - PIE    - 4.3
Calendar Pro         -        armv7(s) - PIE    - 4.3
CalenMob             -        armv7(s) - PIE    - 5.0
Chrome               -        armv7    - PIE    - 4.3
CloudOn              -        armv7    - NO_PIE - 5.0
DiamondDash          -        armv7(s) - PIE    - 4.3
Documents            -        armv7(s) - PIE    - 4.3
Ebook Reader         -        armv7    - PIE    - 4.3
eBookS Reader        - armv6|armv7     - NO_PIE - N/A
Facebook             -        armv7    - PIE    - 4.3
G-Whizz!             - armv6|armv7     - NO_PIE - 4.0
Gmail                -        armv7    - PIE    - 5.0
Google               -        armv7    - PIE    - 4.3
Google Drive         -        armv7    - PIE    - 5.0
Google Earth         -        armv7    - PIE    - 4.3
Google+              -        armv7    - PIE    - 5.0
iBooks               -        armv7    - PIE    - 5.0
IM+                  -        armv7(s) - PIE    - 4.3
Instagram            -        armv7    - PIE    - 4.3
KakaoTalk            -        armv7(s) - PIE    - 4.3
Latitude             - armv6|armv7     - NO_PIE - N/A
Local                - armv6|armv7     - PIE    - 4.3
Lync 2010            -        armv7    - NO_PIE - 4.3
Messenger            -        armv7    - PIE    - 4.3
MSN World            -        armv7(s) - PIE    - 4.3
SkyDrive             - armv6|armv7     - NO_PIE - 4.0
Skype                -        armv7    - NO_PIE - 4.3
SmartGlass           -        armv7    - PIE    - 5.0
SSH Mobile Free      -        armv7(s) - PIE    - 4.3
SystemTools          -        armv7(s) - PIE    - 4.3
Translate            - armv6|armv7     - NO_PIE - N/A
Trillian             -        armv7    - PIE    - 4.3
Twitter              -        armv7    - PIE    - 5.0
Usessh               -        armv7(s) - PIE    - 4.3
```

- all system binaries are compiled as PIE
- most 3rd party apps are now compiled as PIE
- NO_PIE mostly unimportant apps
- some high profile exceptions are: Skype, SkyDrive, Google Translate, ...

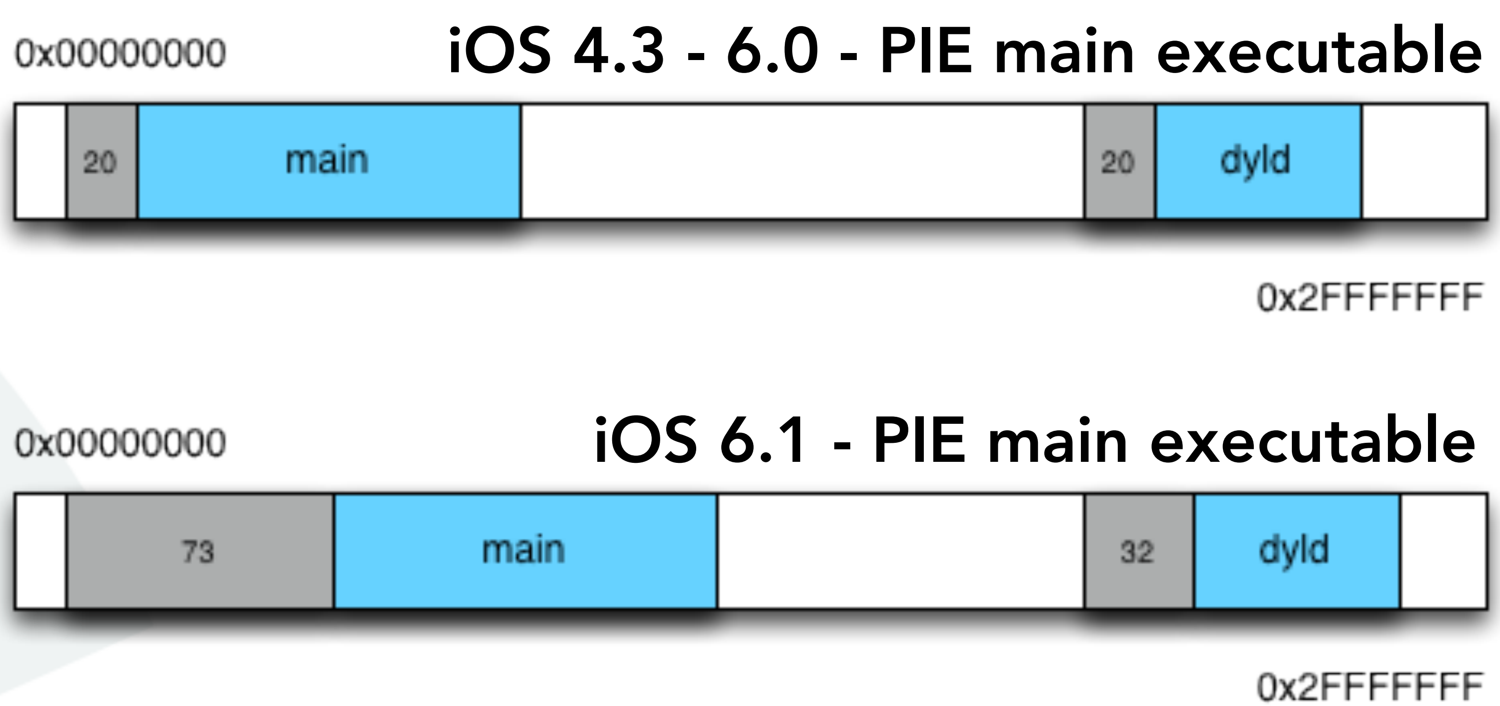
iOS 4.3-6.0: PIE main binary randomization

- for PIE main executables the main binary and dyld are randomized
- main binary and dyld are slid the same amount
- randomized by kernel in 256 positions



iOS 6.1: PIE main binary randomization

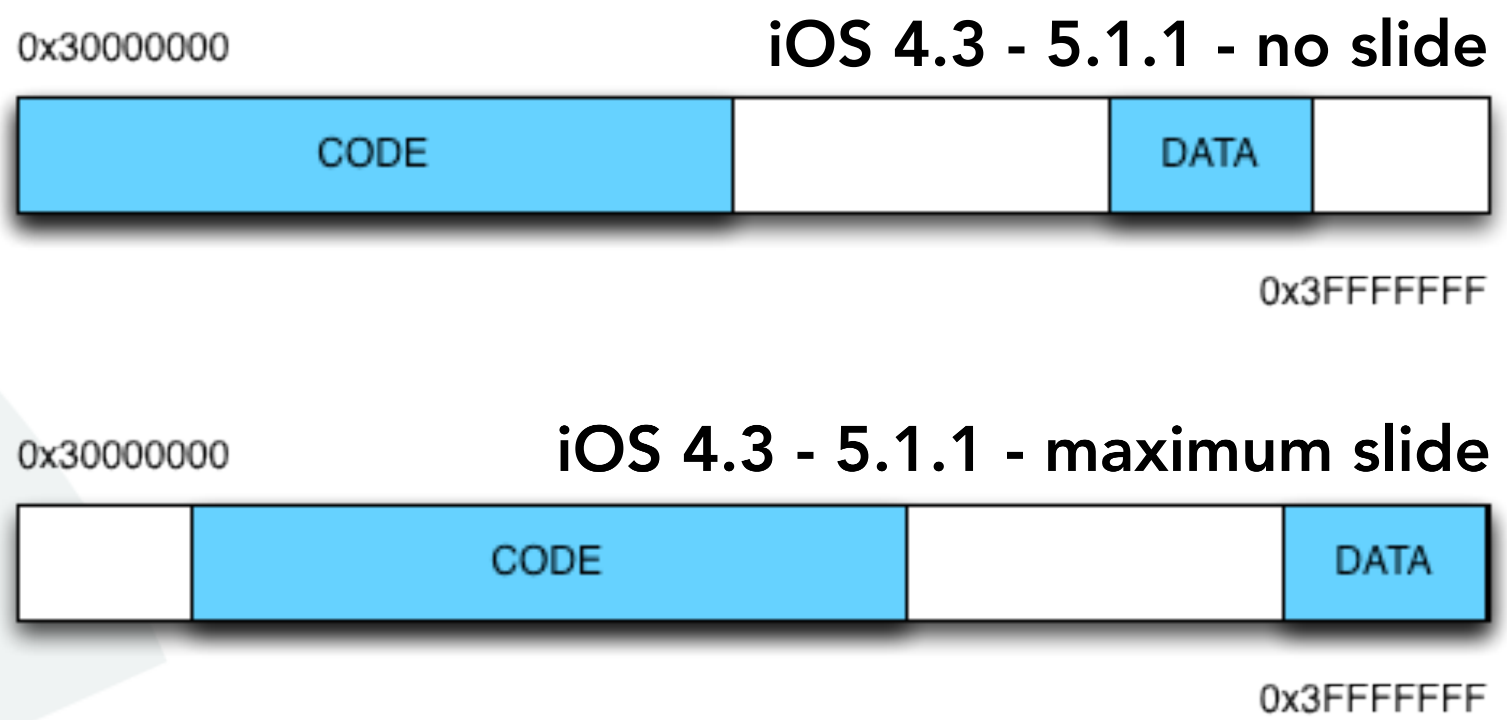
- since iOS 6.1 the kernel finally generates two separate slides
- randomness of both is still limited to 256 positions
- knowing addresses in dyld / main no longer leaks address of other



SektionEins

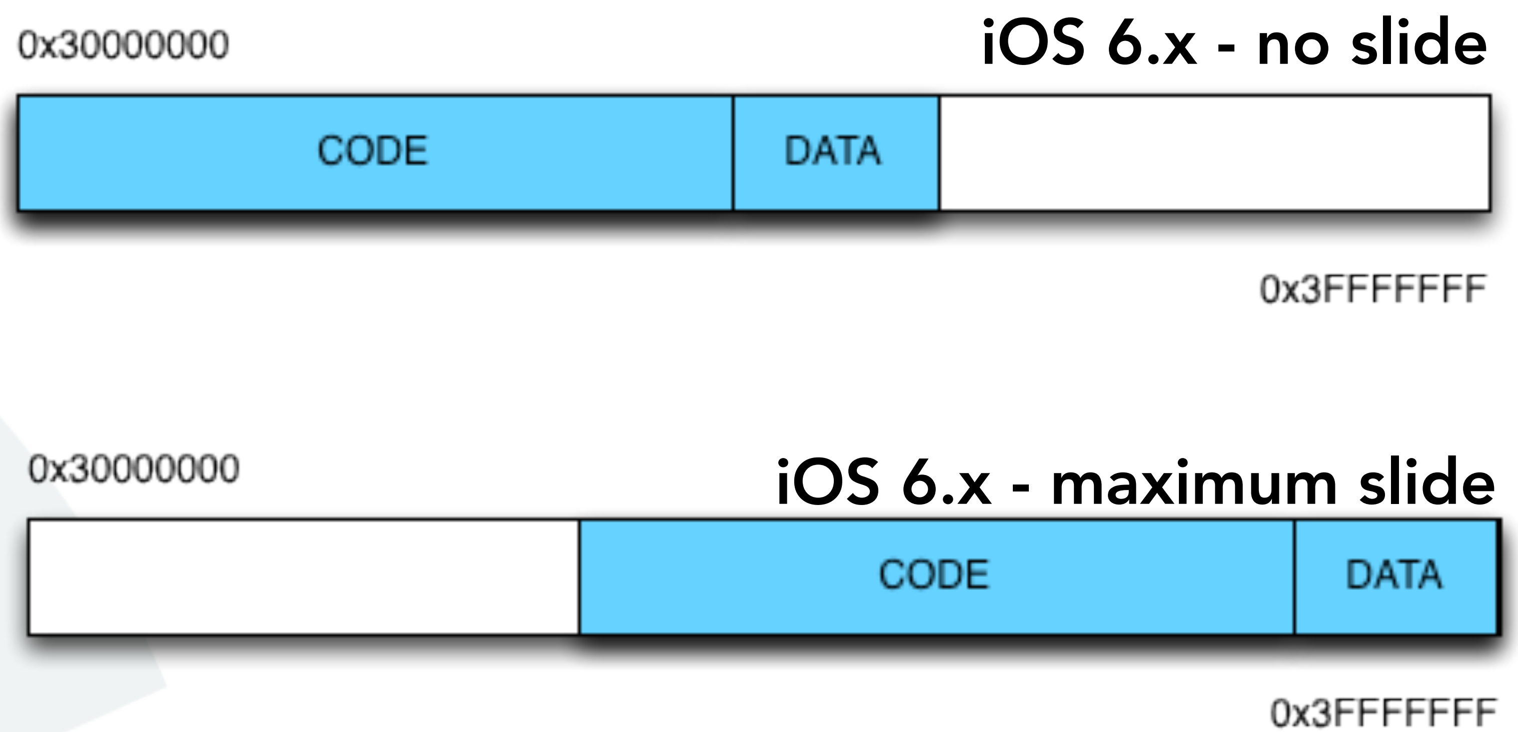
iOS 4.3-5.1.1: dyld_shared_cache randomization

- data and code must slide together *(due to codesigning)*
- hole after code - data usually loaded to 0x3E000000
- max slide determined by difference of end of shared area and end of data
- around 4200 different positions



iOS 6.x: dyld_shared_cache randomization

- code and data loaded right next to each other
- no more hole - no more wasted space
- max slide determined by size of shared area minus size of shared cache
- about 21500 different positions for iPod 4G
  (new devices = more code = less random)



SektionEins

Part V

iOS 6 and the Partial Code-signing Vulnerability

Partial Code-signing Vulnerability (iOS 4)

- in iOS 4.x jailbreaks the method of choice to launch untether exploits
- when a *mach-o* is loaded the kernel will load it as is
- a possible signature will be registered
- missing signature is okay until a not signed executable page is accessed
- dyld was tricked with malformed ***mach-o*** data structures to execute code

sniffLoadCommands (iOS 4.3.4)

- function does pre-handling of mach-o load commands
- iOS 4.3.4 adds protection against partial code signing
  - mach-o load commands must be inside a segment
  - mach-o load commands can only be in R + X segment
  - mach-o load commands may not be partially in a segment

➡ *effect is that once dyld maps the header R+X it can only continue to work on it if there is a valid signature*

Partial Code-signing Vuln (iOS 4.3.4-iOS 5.1.1)

- protection in sniffLoadCommands could be bypassed
    - by having a **`RW-`** ***LC_SEGMENT64*** for ***mach-o*** header
    - and a fake **`R-X`** ***LC_SEGMENT*** for ***mach-o*** header
- disclosed at **CanSecWest 2012** - here on stage
- worked because kernel handles **LC_SEGMENT64** and dyld did not
- magic is that dyld will read mach-o header from from address in memory

sniffLoadCommands (iOS 6.0)

- iOS 6.0 adds protection against CSW 2012 trick to sniffLoadCommands
    - if a LC_SEGMENT64 load command is found an exception is thrown

➡ *CSW 2012 trick was already partially broken after iOS 5.1.1*

- *in iOS 5.1.1 AMFI verified existence of a code signing blob*

Load Command Segment Override (iOS 6.0-6.1.2)

- bug used by evasi0n
- kernel not directly involved in loading dynamic libraries only dyld is
- dyld could be tricked with a malicious dylib
  - contains real R-X segment with load commands in it
  - contains second R-- segment that contains copy of load commands
  - virtual address of both segments is set to same position
  - later segment in mach-o will replace previous in memory
- when dyld accesses header it is in RO memory = no sig needed = bypass

sniffLoadCommands (iOS 6.1.3 beta 2)

- iOS 6.1.3 beta 2 adds additional protections to sniffLoadCommands
  - load commands must now be in one segment only
  - for dynamic libraries a second sniff pass is added
  - all segments must not intersect the R-X segment containing the load commands

➡ evasi0n untether killed

Part VI

iOS 6.1 and Launch-Daemon-Code-Signing

Launch Daemons to launch Untethers

- in iOS 5.x jailbreaks were launched on boot via launch daemons
- launch daemons are plists describing system services

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://
www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Label</key>
    <string>jb</string>
    <key>ProgramArguments</key>
    <array>
        <string>/usr/sbin/corona</string>
        <string>-f</string>
        <string>racoon-exploit.conf</string>
    </array>
    <key>WorkingDirectory</key>
    <string>/usr/share/corona/</string>
    <key>RunAtLoad</key>
    <true/>
    <key>LaunchOnlyOnce</key>
    <true/>
    <key>DisableAslr</key>
    <true/>
</dict>
</plist>
```

DisableAslr was removed from iOS 5.1

Launch-Daemon-Code-Signing (I)

- abuse of launch daemons lead to new iOS 6.1 security feature
- launch daemon loading is now code signed
- implemented in **`/bin/launchctl`**
- can be bypassed by setting kernel boot arguments
  (not possible without low-level exploit)

```
bool launchctl_enforce_codesign()
{
  char buffer[1024];
  char *p, *tmp = NULL;
  size_t len;
  int res = 0;

  len = sizeof(buffer);
  if ( !sysctlbyname("kern.bootargs", buffer, &len, 0, 0) )
  {
    p = strnstr(buffer, "cs_enforcement_disable=", len);
    if ( p )
      res = strtoul(p + 23, 0, 10);
    p = strnstr(buffer, "launchctl_enforce_codesign=", len);
    if ( p )
    {
      if (strtoul(p + 27, &tmp, 10) == 0)
        res = 1;
    }
  }
  return res == 0;
}
```

Launch-Daemon-Code-Signing (II)

- without launch-daemon-code-signing
  `/bin/launchctl` scans **`/System/Library/LaunchDaemons`** for defined launch daemons and loads them

- with activated launch-daemon-code-signing
  a big plist with all defined launch daemons is loaded instead

- launch daemon can only be loaded if it is defined in the plist and exists on disk

```
if ( !LaunchDaemonCachePlist )
{
  length = 0;
  xpcd_cache = dlopen("/System/Library/Caches/com.apple.xpcd/xpcd_cache.dylib", 2);
  if ( !xpcd_cache )
  {
    dlerror_msg = dlerror();
    launchctl_log(3, "cache loading failed: dlopen returned %s.", dlerror_msg);
    goto error1;
  }
  __xpcd_cache = dlsym(xpcd_cache, "__xpcd_cache");
  if ( !__xpcd_cache )
  {
    msg = "cache loading failed: failed to find __xpcd_cache symbol in cache.";
    goto LABEL_6;
  }
  if ( !dladdr(__xpcd_cache, &dl_info) )
```

Launch-Daemon-Code-Signing (III)

- big launch daemon plist is loaded from **`/System/Library/Caches/com.apple.xpcd/xpcd_cache.dylib`**
- this dynamic library is within the **dyld_shared_cache** and therefore **code signed**
- symbol **`__xpcd_cache`** must exist
- but binary plist is take from sectiondata of **`__TEXT::__xpcd_cache`**

```
if ( !LaunchDaemonCachePlist )
{
  length = 0;
  xpcd_cache = dlopen("/System/Library/Caches/com.apple.xpcd/xpcd_cache.dylib", 2);
  if ( !xpcd_cache )
  {
    dlerror_msg = dlerror();
    launchctl_log(3, "cache loading failed: dlopen returned %s.", dlerror_msg);
    goto error1;
  }
  __xpcd_cache = dlsym(xpcd_cache, "__xpcd_cache");
  if ( !__xpcd_cache )
  {
    msg = "cache loading failed: failed to find __xpcd_cache symbol in cache.";
    goto LABEL_6;
  }
  if ( !dladdr(__xpcd_cache, &dl_info) )
```

XPCD_CACHE.PLIST

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.d
<plist version="1.0">
<dict>
  <key>CreationDate</key>
  <date>2013-13-13T13:13:13Z</date>
  <key>LaunchDaemons</key>
  <dict>
    <key>/System/Library/LaunchDaemons/com.apple.AOSNotification.plist</key>
    <dict>
      <key>JetsamProperties</key>
      <dict>
        <key>JetsamMemoryLimit</key>
        <integer>1024</integer>
        <key>JetsamPriority</key>
        <integer>-49</integer>
      </dict>
      <key>KeepAlive</key>
      <dict>
        <key>PathState</key>
        <dict>
          <key>/var/mobile/Library/Preferences/com.apple.AOSNotification.FMFAccounts.plist</key>
          <true/>
          <key>/var/mobile/Library/Preferences/com.apple.AOSNotification.launchd</key>
          <true/>
        </dict>
      </dict>
```

Launch-Daemon-Code-Signing Security

How secure Apple wanted Launch-Daemon-Code-Signing to be...

Launch-Daemon-Code-Signing Security

How secure Launch-Daemon-Code-Signing is right now...

Launch-Daemon-Code-Signing Security

- code signing itself seems to stop loading arbitrary launch daemons
- but Apple forgot / or ignored **`/etc/launchd.conf`**
- **`/etc/launchd.conf`** defines commands **launchctl** executes on start
- attacker can execute arbitrary existing commands

```
bsexec .. /sbin/mount -u -o rw,suid,dev /
setenv DYLD_INSERT_LIBRARIES /private/var/evasi0n/amfi.dylib
load /System/Library/LaunchDaemons/com.apple.MobileFileIntegrity.plist
bsexec .. /private/var/evasi0n/evasi0n
unsetenv DYLD_INSERT_LIBRARIES
bsexec .. /bin/rm -f /private/var/evasi0n/sock
bsexec .. /bin/ln -f /var/tmp/launchd/sock /private/var/evasi0n/sock
```

FAQ: Why not put old launchctl binary on device?

Q: *"If only the newest iOS 6.1* ***launchctl*** *binary implements this code signing. Why not put an iOS 6.0* ***launchctl*** *binary on the device to bypass this protection?"*

A: "System binaries like **launchctl** do not come with a valid code signing signature from Apple. Instead they come only with the table of memory page hashes and entitlements. When the kernel loads such a binary it hashes these tables and checks if the hash is in a whitelist inside the kernel (a.k.a. trust cache). The old **launchctl** binary will not be accepted because it is not in the trust cache of the new kernel."

Final Words

- with iOS 6 Apple has tried to kill all public techniques
- finally kills some stuff that was previously known and ignored for 10 years
- the new mitigations make exploitation a lot harder
- when launch daemon code signing is hardened a bit more, persisting on iDevices will become incredibly hard
- however there are still weaknesses in most of the protections
- ... and tons of kernel information leaks

Questions

?