iOS Security Internals

Dino Dai Zovi
Trail of Bits

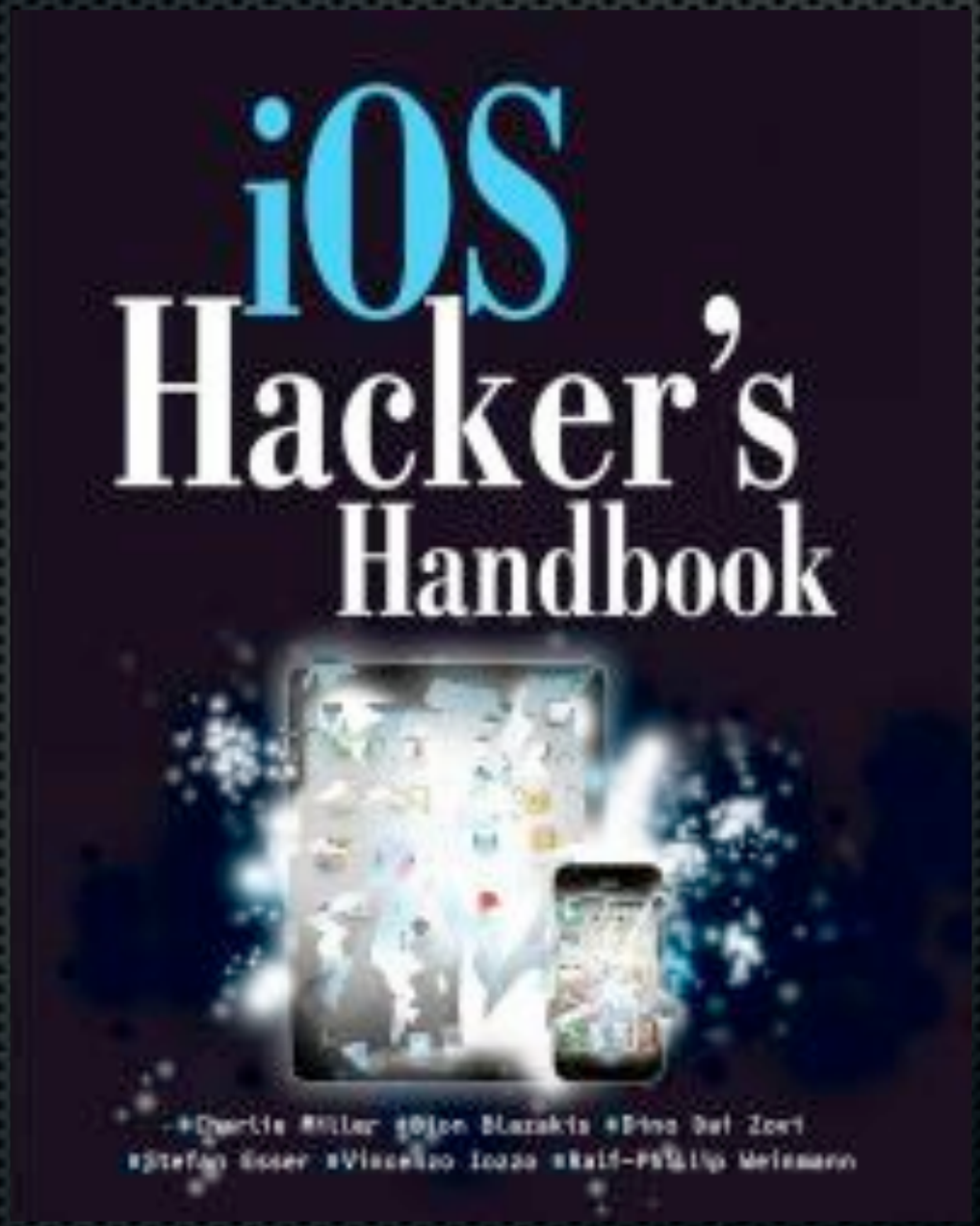
Charlie Miller
Accuvant Labs

Session ID:MBS-402
Session Classification: Advanced

About us

Charlie	Dino
Hacked several Macs, usually in 15 seconds or less	Stayed up all night and hacked a Mac once
Banned from App Store	Not banned from App Store
Bad cop	Good cop
Apple Fanboi	Apple Fanboi

Upcoming book



Agenda

- Mobile Malware vs. iOS
- iOS Security Architecture and Internals
- History of iOS Attacks

Malware Vectors

- User is tricked into downloading and running malicious software via various forms of social engineering
  - Malicious apps in App Store
- Vulnerabilities in software leveraged during normal user behavior (exploits)
  - Malicious e-mail or attachment (“spear phishing”)
  - Malicious web content (“drive by download”)

Malware

- It is hard to design a security model which protects against programs a user downloads and wants to run
- It is typically not the job of the OS to prevent you from running the programs you choose to run
- Anti-Virus is designed to help decide which programs are okay to run and which are not

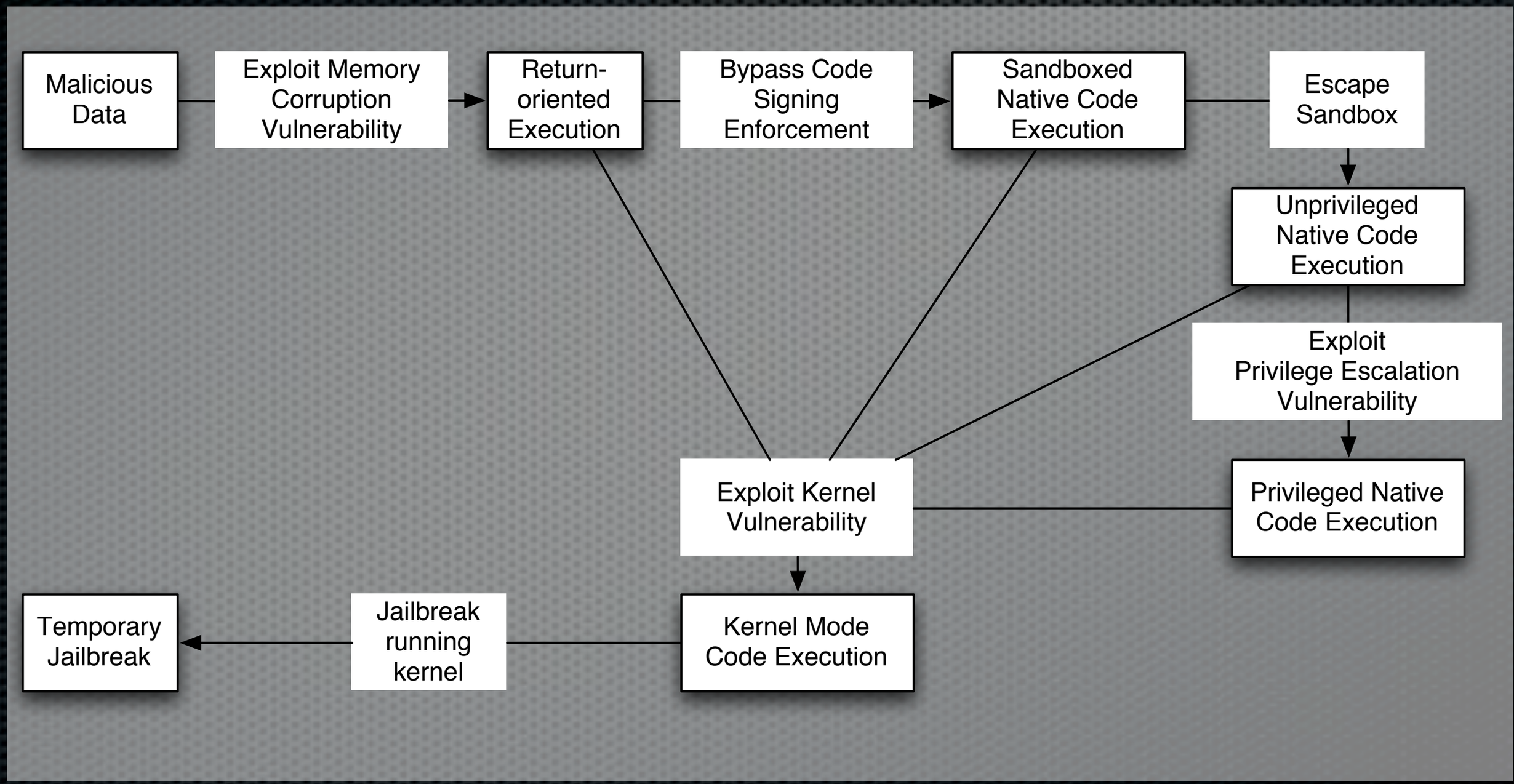
Malware vs. iOS

- Code Signing requires apps to be downloaded from the App Store
- Publishers' real-world identities are verified by Apple
- Apps are reviewed by Apple before they are available in App Store
- Apple acts as an Anti-Virus for iOS

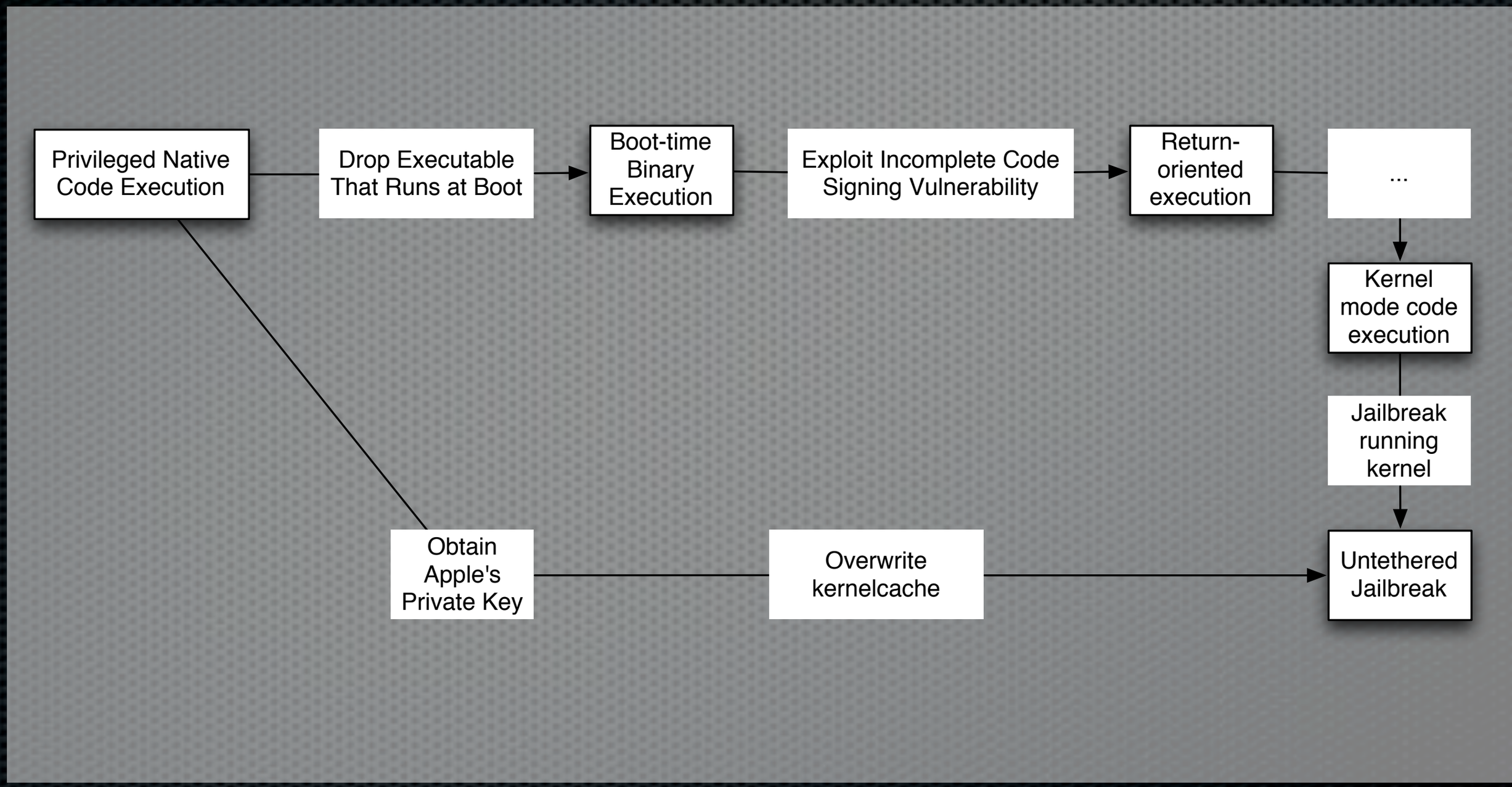
Exploits

- Exploits take advantage of some vulnerability to run code without your permission or knowledge
- Exploits can be stopped by either having no vulnerabilities in the OS/applications or making exploitation impossible even in the presence of vulnerabilities
- Many exploits take advantage of memory corruption
- Memory corruption might be a buffer overflow (stack or heap), or a “wild write”, use after free, etc

Remote Exploit Attack Graph



Persistence Attack Graph



iPhone security architecture

- Reduced attack surface
- Stripped down OS
- Privilege separation
- Code Signing
- Non-Executable Memory
- Address Space Layout Randomization
- Sandboxing

Reduced attack surface

- Not as many application to attack
  - No Flash, Java
- MobileSafari will not render all filetypes that Safari (via QT) will
  - .psd files
  - Even some .mov files won't play
- iPhone doesn't handle all features of PDF's.
  - Only 7% of PDF crashes I find reproduce on iPhone
- Smaller attack surface -> fewer bugs to exploit

Stripped down OS

- Missing lots of useful binaries
- No /bin/sh
- Means no “shell”code
- Even if you had a shell, no ls, rm, ps, etc
- If you get code execution, what do you do?

Privilege separation

- Most processes run as user “mobile”
  - MobileSafari, MobileMail, Springboard, etc
- Many resources require privilege of root (administrative) user
- Sometimes that isn’t even enough...

Entitlements

- Specific applications or executables can be granted special privileges (entitlements)
  - i.e. com.apple.coreaudio.allow-amr-decode
- Entitlements list is protected by code signing and signing certificate must be authorized to grant particular entitlements
- IOKit drivers and Mach RPC servers can verify that callers possess a particular entitlement

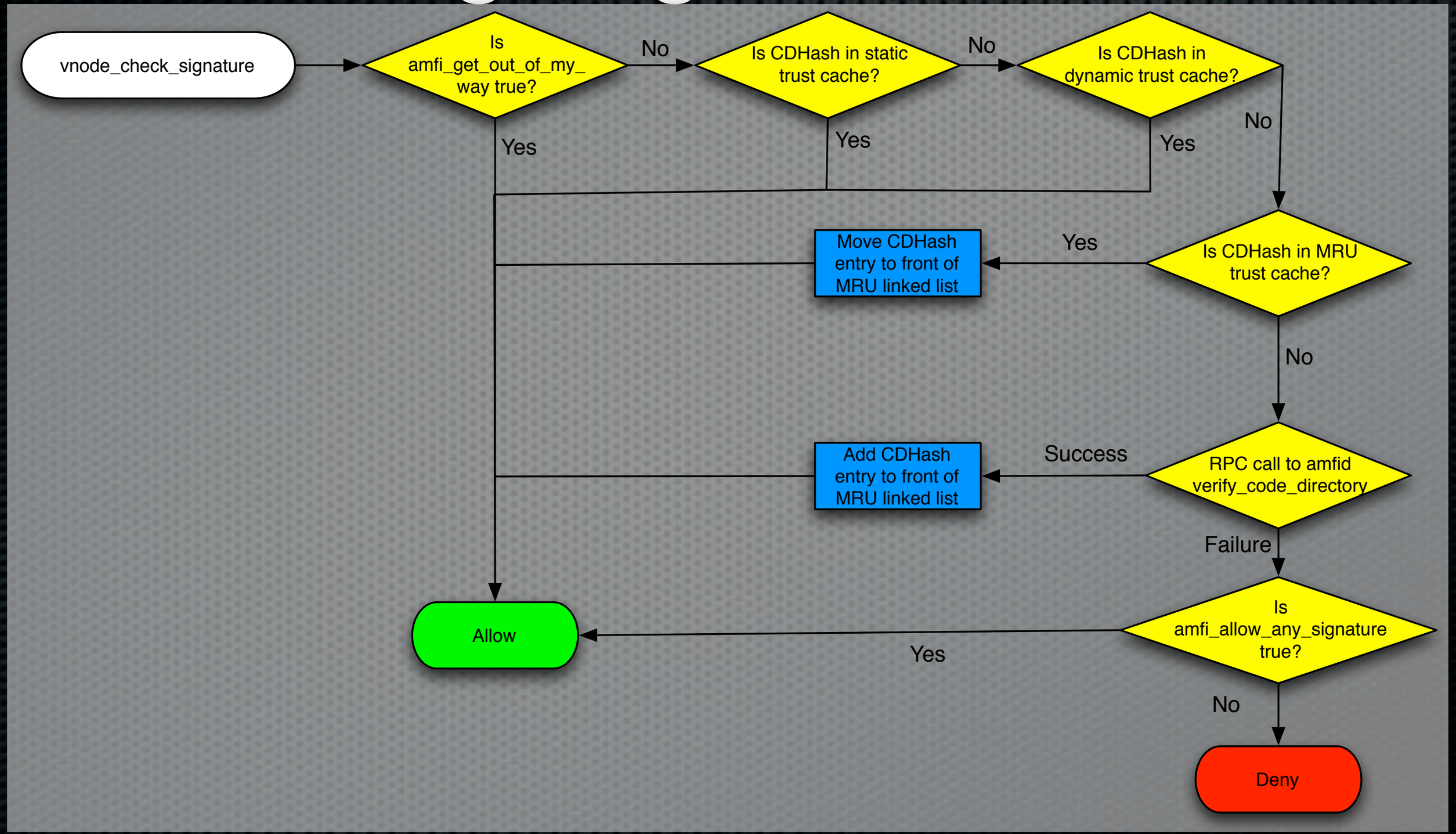
Code Signing

- All binaries and libraries must be signed by Apple
- Or phone needs to be specially provisioned
- This is why applications have to come from the AppStore
- And why people Jailbreak their phones
- Attacker cannot upload a binary and run it

Code Signing

- Mandatory Code Signing
  - Every executable binary or application must have a valid and trusted signature
  - Enforced when an application or binary is executed
- Code Signing Enforcement
  - Processes may only execute code that has been signed with a valid and trusted signature
  - Enforced at runtime

Code Signing Verification



Non-Executable Memory

- XN bit is ARM analogue of the NX/XD bits on Intel/AMD
- Data pages (stack, heap) are marked non-executable
- iOS enforces W^X page protection policy
  - No memory pages may be RWX
  - Pages that are writeable can not become executable
- Injected machine code cannot be immediately executed

Code Signing Enforcement

- Ensures that process stays dynamically valid
  - No introduction of new executable code
    - Existing executable code can’t be changed
- Guarantees that the app code that was reviewed is what runs on the device
  - Also just happens to prevent injecting shellcode

iOS 4.3 Adds JavaScript JIT

Apple Introduces iOS 4.3

Update Includes Faster Safari Performance, iTunes Home Sharing, AirPlay Improvements & New Personal Hotspot

SAN FRANCISCO—March 2, 2011—Apple® today introduced iOS 4.3, the latest version of the world's most advanced mobile operating system. New features in iOS 4.3 include faster Safari® mobile browsing performance with the Nitro JavaScript engine; iTunes® Home Sharing; enhancements to AirPlay®; the choice of using the iPad™ side switch to either lock the screen rotation or mute the audio; and the Personal Hotspot feature for sharing an iPhone® 4 cellular data connection over Wi-Fi.

"With more than 160 million iOS devices worldwide, including over 100 million iPhones, the growth of the iOS platform has been unprecedented," said Steve Jobs, Apple's CEO. "iOS 4.3 adds even more features to the world's most advanced mobile operating system, across three blockbuster devices—iPad, iPhone and iPod touch—providing an ecosystem that offers customers an incredibly rich experience and developers unlimited opportunities."

The Safari mobile browsing experience gets even better with iOS 4.3. The Nitro JavaScript engine that Apple pioneered on the desktop is now built into WebKit, the technology at the heart of Safari, and more than doubles the performance of JavaScript execution using just-in-time compilation. With the Nitro JavaScript engine, Safari provides an even better mobile browser experience working faster to support the interactivity of complex sites you visit on a daily basis.

```
# ldid -e /Applications/MobileSafari.app/MobileSafari
<?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>com.apple.coreaudio.allow-amr-decode</key>
	<true/>
	<key>com.apple.coremedia.allow-protected-content-playback</key>
	<true/>
	<key>com.apple.managedconfiguration.profiled-access</key>
	<true/>
	<key>com.apple.springboard.opensensitiveurl</key>
	<true/>
	<key>dynamic-codesigning</key>
	<true/>
	<key>keychain-access-groups</key>
	<array>
		<string>com.apple.cfnetwork</string>
		<string>com.apple.identities</string>
		<string>com.apple.mobilesafari</string>
	</array>
	<key>platform-application</key>
	<true/>
	<key>seatbelt-profiles</key>
	<array>
		<string>MobileSafari</string>
	</array>
</dict>
</plist>
```

AMFI MAC Hook

MAC Hook	API Description	AMFI Usage
mpo_proc_check_map_anon	Determine whether the subject identified by the credential should be allowed to obtain anonymous memory with the specified flags and protections.	Allows the process to allocate anonymous memory if and only if the process has the dynamic-codesigning entitlement.

Dynamic Code Signing

- The **dynamic-codesigning** entitlement allows the process to map anonymous memory *with any specified protections*.
- Only MobileSafari has this entitlement in iOS 4.3
  - Necessary for JavaScript native JIT (“Nitro”)
  - Previously MobileSafari did bytecode JIT

ASLR

- iPhone randomizes....
  - Binary
  - Libraries
  - Dynamic loader
  - Heap
  - Stack
  - etc...

ASLR without PIE

Executable	Heap	Stack	Libraries	Linker
0x2e88	0x15ea70	0x2fdff2c0	0x36adadd1	0x2fe00000
0x2e88	0x11cc60	0x2fdff2c0	0x36adadd1	0x2fe00000
0x2e88	0x14e190	0x2fdff2c0	0x36adadd1	0x2fe00000
0x2e88	0x145860	0x2fdff2c0	0x36adadd1	0x2fe00000
0x2e88	0x134440	0x2fdff2c0	0x36adadd1	0x2fe00000

Reboot

0x2e88	0x174980	0x2fdff2c0	0x35e3edd1	0x2fe00000
0x2e88	0x13ca60	0x2fdff2c0	0x35e3edd1	0x2fe00000
0x2e88	0x163540	0x2fdff2c0	0x35e3edd1	0x2fe00000
0x2e88	0x136970	0x2fdff2c0	0x35e3edd1	0x2fe00000
0x2e88	0x177e30	0x2fdff2c0	0x35e3edd1	0x2fe00000

ASLR with PIE

Executable	Heap	Stack	Libraries	Linker
0xd2e48	0x1cd76660	0x2fecf2a8	0x35e3edd1	0x2fed0000
0xaae48	0x1ed68950	0x2fea72a8	0x35e3edd1	0x2fea8000
0xbbe48	0x1cd09370	0x2feb82a8	0x35e3edd1	0x2feb9000
0x46e48	0x1fd36b80	0x2fe432a8	0x35e3edd1	0x2fe44000
0xc1e48	0x1dd81970	0x2febe2a8	0x35e3edd1	0x2febf000

Reboot

0x14e48	0x1dd26640	0x2fe112a8	0x36146dd1	0x2fe12000
0x62e48	0x1dd49240	0x2fe112a8	0x36146dd1	0x2fe60000
0x9ee48	0x1d577490	0x2fe9b2a8	0x36146dd1	0x2fe9c000
0xa0e48	0x1e506130	0x2fe9d2a8	0x36146dd1	0x2fe9e000
0xcde48	0x1fd1d130	0x2feca2a8	0x36146dd1	0x2fecb000

Partial vs. Full ASLR

PIE	Main Executable	Heap	Stack	Shared Libraries	Linker
No	Fixed	Randomized per execution	Fixed	Randomized per device boot	Fixed
Yes	Randomized per execution	Randomized per execution (more entropy)	Randomized per execution	Randomized per device boot	Randomized per execution

Identifying PIE support

- otool -hv <executable>

```
$ otool -hv MobileSafari
MobileSafari:
Mach header
      magic cputype cpusubtype  caps    filetype ncmds sizeofcmds      flags
   MH_MAGIC     ARM         V7  0x00     EXECUTE    40       4560   NOUNDEFS DYLDLINK TWOLEVEL PIE
```

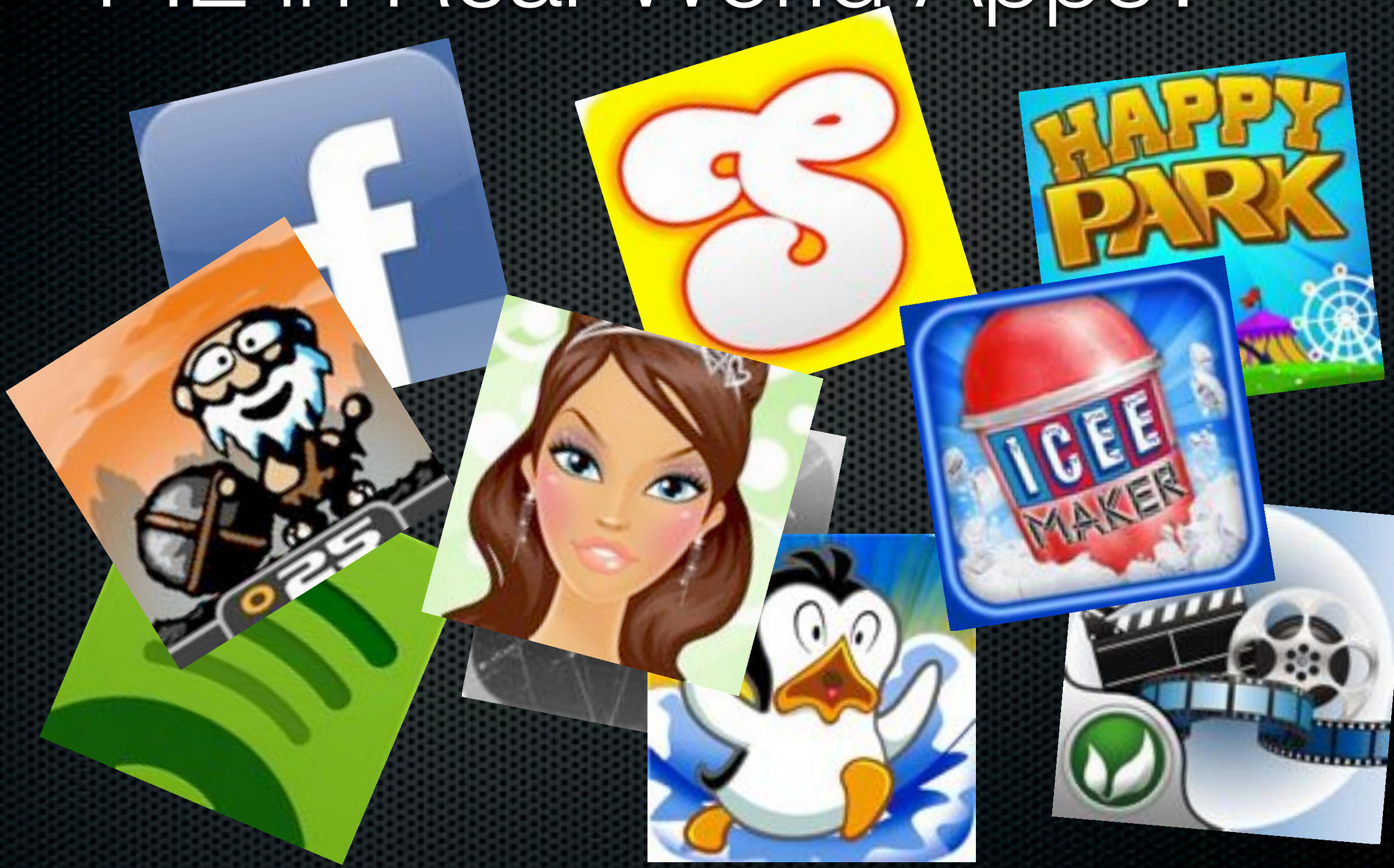
- hexdump

```
$ hexdump -C MobileSafari | head
00000000  ce fa ed fe 0c 00 00 00  09 00 00 00 02 00 00 00  |................|

00000010  28 00 00 00 d0 11 00 00  85 00 20 00 01 00 00 00  |(......... .....|
```

PIE in Real-World Apps?

PIE in Real-World Apps?



Top 10 Free Apps (July 2011)

App	Version	Post Date	PIE
Songify	1.0.1	June 29, 2011	No
Happy Theme Park	1.0	June 29, 2011	No
Cave Bowling	1.10	June 21, 2011	No
Movie-Quiz Lite	1.3.2	May 31, 2011	No
Spotify	0.4.14	July 6, 2011	No
Make-Up Girls	1.0	July 5, 2011	No
Racing Penguin, Flying Free	1.2	July 6, 2011	No
ICEE Maker	1.01	June 28, 2011	No
Cracked Screen	1.0	June 24, 2011	No
Facebook	3.4.3	June 29, 2011	No

Sandboxing

- Applications from the AppStore run in a sandbox
- Sandbox limits what applications can do
- Uses same mechanism as Mac OS X (Seatbelt kext)
- Limits what files can be read/written
- Limits what resources can be accessed

Apple applications

- Many other applications also have a sandbox: MobieSafari, MobileMail, MobileSMS
- Less restrictive
- Compiled into the kernel
- Can:
  - Open SMS database
- Can not:
  - Send SMS messages, fork()

Appstore apps

- More restrictive sandbox
- Cannot access most of filesystem
- Further restrictions on API usage by Apple...

App Home Directory

Subdirectory	Description
<AppName>.app/	The signed bundle containing the application code and static data
Documents/	App-specific user-created data files that may be shared with the user's desktop through iTunes's "File Sharing" features
Library/	Application support files
Library/Preferences/	Application-specific preference files
Library/Caches/	App-specific data that should persist across successive launches of the application but not needed to be backed up
tmp/	Temporary files that do not need to persist across successive launches of the application

App Container Profile

- See BH USA 2011 whitepaper for detailed description and tarball for fully decompiled profile
- Summary:
  - File access is generally restricted to app's home directory
  - Can read user's media: songs, photos, videos
  - Can read and write AddressBook
  - Some IOKit User Clients are allowed
  - All Mach bootstrap servers are allowed

Mach Bootstrap Servers

- All Mach tasks have access to a bootstrap port to lookup service ports for Mach RPC services
  - On iOS, this is handled by launchd
- 141 RPC servers accessible from apps
  - Risk of being exploited over RPC
  - May present risk of allowing apps to perform unauthorized or undesirable actions

Example Servers

- com.apple.UIKit.pasteboardd
- com.apple.springboard
- com.apple.MobileFileIntegrity
- com.apple.fairplayd
- com.apple.mobile.obliteration
- com.apple.iTunesStore.daemon

iPhone security summary

- Hard(er) to find bugs
- No good binaries on system
- No way to write files and execute them
- Shellcode would have to be large and complicated
- Shellcode won’t run anywhere
- Payloads are restricted to compromised process, running as mobile, in a sandbox...

Jailbreaking

- Jailbreaking adds /bin/sh and friends
- Disables code signing (obviously)
- Disables many memory protections
- Increases attack surface (SSHD, etc)
- Many cydia apps run as root with no sandbox
- *Basically breaks security architecture of device*

Attacks against iOS

- Libtiff iPhone exploit (v1.0)
- iPhone SMS exploit (<v3.1)
- Ikee (jailbroken)
- SpyPhone (all)
- Storm 8 (all)
- Pwn2Own 2010 (<v4.0)
- Jailbreakme.com v2 (<v4.0.2)
- Jailbreakme.com v3 (<v4.3.3)
- Code signing (<5.0.1)

Libtiff exploit

- Back in iPhone version 1.0
  - Everything ran as root
  - No ASLR/DEP
  - Shipped with known vulnerable version of libtiff
- Buffer overflow found by Tavis Ormandy
  - Exploit provided by Chris Wade
- Surf with MobileSafari to the wrong site, attacker gets unrestricted root access to your device
- Exploit used to jailbreak iPhones remotely (jailbreakme.com version 1)
- Patched in 1.1.2

Libtiff today

- A MobileSafari exploit today gets the attacker “Apple”-sandboxed permission running as user mobile
- Original exploit used heap spray + shellcode. Now would have to use ROP
- Attacker can do things like read files, keyboard cache, sniff web traffic, read stored SMS, etc.
- Attacker can not tamper with system files, jailbreak, send SMS messages, etc.
- Attacker may try to escape the sandbox, raise privilege level
  - This may require 1 or 2 more vulnerabilities/exploits
  - Not impossible but harder

SMS fun

- In 2009, myself and Collin Mulliner demonstrated a remote exploit over SMS
- Vulnerability in the CommCenter process
  - Runs as root with no sandbox
- Attacker gets full root privilege, doomsday
- SMS is the perfect attack vector
  - No user interaction required
  - User cannot block or turn off SMS
  - Attack is queued by teleco while phone is off
  - Does not (have to) make any indication on the phone attack is occurring
- By way of comparison, comparable Android bug was a Java exception
- Today, CommCenter runs as _wireless

Ikee

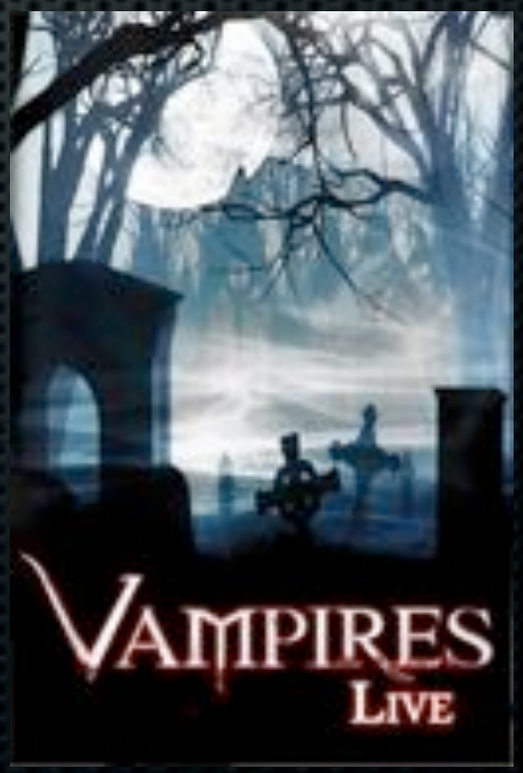
- Aka Dutch ransom, iPhone/Privacy.A, Duh/Ikee.B
- Jailbroken phones with default SSHD passwords
  - Can’t trust users to not mess up the security of their phones
- Gives full unsandboxed, root level access
- Malware does things like
  - change wallpaper
  - Lock phone for ransom
  - Steal content
  - Change password and become a part of a botnet

SpyPhone

- A proof of concept “AppStore” application by Seriot Nicolas
- Completely legit use of API’s, lives in AppStore sandbox
- Still can do the following
  - Get cell number
  - Read/write access to Address Book
  - Safari/YouTube searches
  - Email account information
  - Contacts, keyboard cache
  - Geotagged photos
  - GPS, WiFi AP names

Storm 8

- In 2009, games developed by Storm 8 were collecting cell numbers of devices
- 20 million downloads
- Storm 8 says it was a mistake



Pwn2Own 2010

- Vulnerability exploited from WebKit in Mobile Safari by Vincenzo Iozzo and Ralf-Philipp Weinmann
- Won $15,000 and an iPhone
- Exploit used ROP payload to transmit SMS database to remote server
- Did not escape the sandbox

jailbreakme.com 2 (“Star”)

- Two vulnerabilities
  - CVE-2010-1797: stack overflow in FreeType’s handling of CFF opcodes
  - CVE-2010-2973: Integer overflow in IOSurface property in IOkit
- One to get control, one to get root/bypass sandbox
- ROP Payload
  - Executes IOSurface bug
  - Disables code signing with memory writes
  - Downloads and loads dylib
- Remote root
  - reminiscent of iPhone 1 days
- Patched in less than 2 weeks

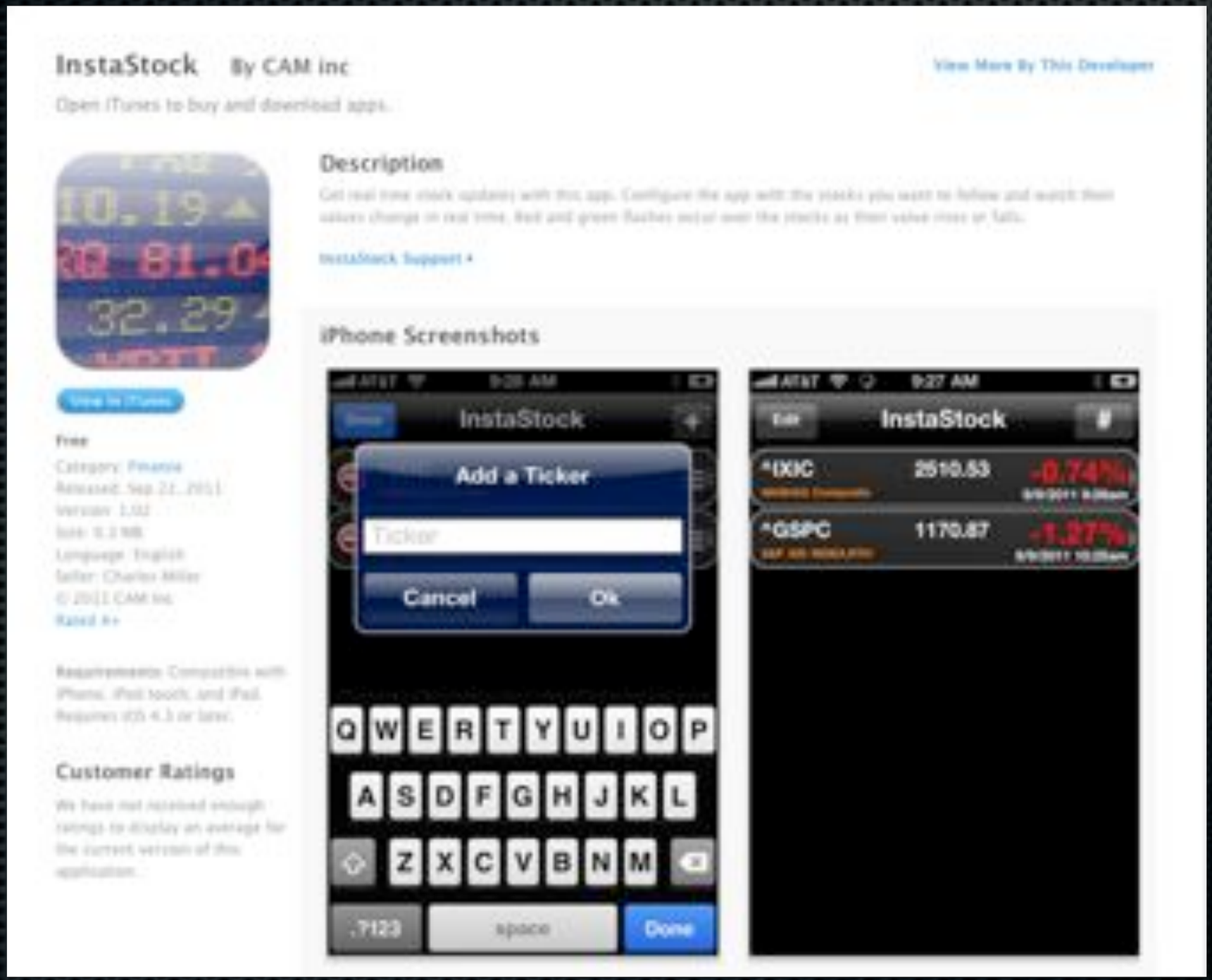
jailbreakme.com 3 (“Saffron”)

- Bypassed all restrictions currently in place in iOS
- Vulnerability in font parsing code allowed reading and writing of memory past buffer
  - Bypass ASLR and corrupt memory
- Second exploit gave full kernel compromise

Code signing bugs

- Found a bug that allowed running (signed) applications to run unsigned code
  - iOS 2 and again in iOS 5
- This would allow “innocent” apps to download unapproved code and run it

Submitted to App Store



App Store Review Process

- Not very close inspection!
- My app was pretty suspicious
  - Tries to download file, does a bunch of pointer manipulation, calls function pointers, etc
  - Both apps had exactly the same code in it
  - Written by ME!
- Suggests they don't actually look at the code for this kind of thing
  - Maybe their review process is not as effective at stopping malware as they contend

Questions?

- Charlie: charlie.miller@accuvant.com
- Dino: ddz@theta44.org