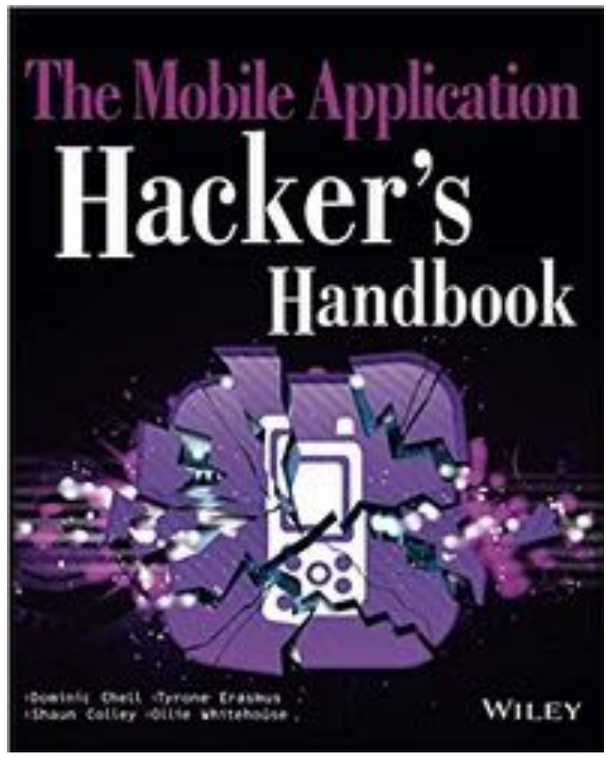
# CNIT 128
# Hacking Mobile Devices




## 2. Analyzing iOS Apps
## Part 1

# Topics: Part 1

- The Security Model
- iOS Apps
- Jailbreaking Explained

# Topics: Part 2

- The Data Protection API
- The iOS Keychain
- TouchID
- Reverse Engineering iOS Binaries

# The Security Model

# Security Features

- Secure boot chain
- Code signing
- Process-level sandboxing
- Data-at-rest encryption
- Generic native language exploit mitigations
  - Address space layout randomization
  - Non-executable memory
  - Stack smashing protection

# Initializing iOS with Secure Boot Chain

- Initializing and loading firmware
- Each step is cryptographically signed and verified


**Figure 2.1** The secure boot chain

# Secure Boot Chain

- Boot ROM
  - Read-only portion of the processor
    - Contains public key for Apple's CA
      - Used to verify next step: the LLB

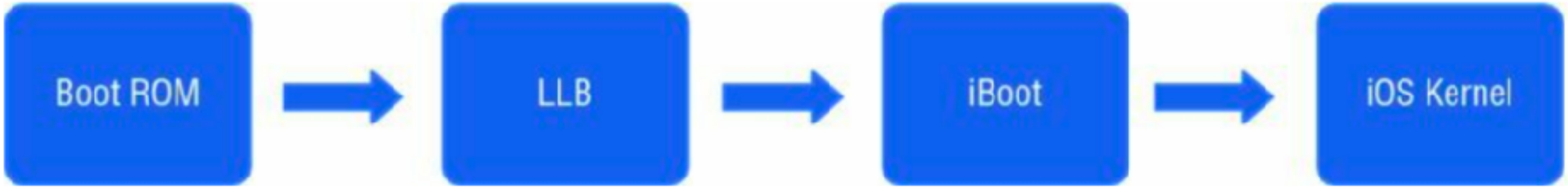



**Figure 2.1** The secure boot chain

# Secure Boot Chain

- LLB (Low-Level Bootloader)
  - Finds iBoot
  - Verifies its signature
  - If signature check fails, boots into recovery mode

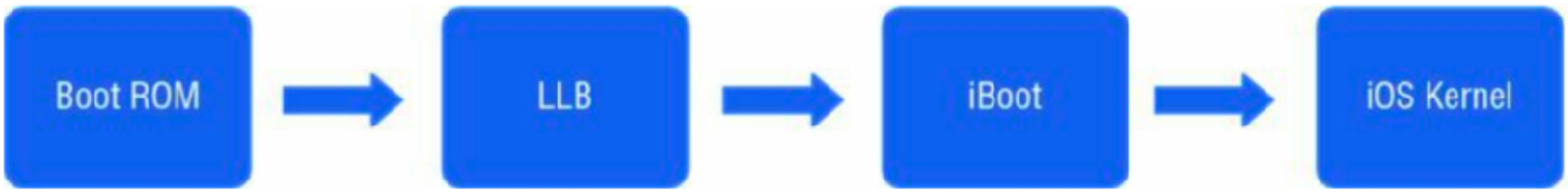



**Figure 2.1** The secure boot chain

# Secure Boot Chain

- iBoot
  - Verifies and loads the iOS kernel
- iOS Kernel
  - Loads usermode environment and OS


**Figure 2.1** The secure boot chain

# Secure Enclave

- Secure coprocessor shipped with all modern iPhones and iPads
  - Since iPhone 5s
- Handles cryptography on device
  - Key management for
    - Data Protection API
    - Touch ID fingerprint data

# Secure Enclave

- A customized version of ARM TrustZone
  - Partitions itself from main processor
    - Provides data integrity even if kernel is compromised
- Even if device is jailbroken
  - Secrets cannot be extracted
  - Such as fingerprint data

# Cachegrab

- Released in Dec. 2017
- Kernel code can deduce some information about contents of the Secure Enclave
- Because it shares cache memory with kernel processes
  - Link Ch 2a

# Restricting Application Processes with Code Signing

- Validates signatures each time an app is executed
    - Only code signed with a trusted signature can run
- Developers can install trusted certificates
    - In a ***provisioning profile*** signed by Apple

# App Store

- Production apps must be signed by Apple
  - After submitting them to the App Store
- Apple bans risky activities
  - Private APIs
  - Apps that download and install executable code

# Isolating Apps with Process-Level Sandboxing

- All third-party apps run in a ***sandbox***
  - Isolated from one another and from OS
- All apps run as the same mobile system user
  - Each app is contained in its unique directory
  - Separation maintained by the XNU Sandbox kernel extension

# The Apple Sandbox

- Introduced way back in Mac OS 10.5 as “Seatbelt”
  - Very naive implementation originally, bypassed and opt-in

- Revamped in Mac OS 10.7 as “The App Sandbox”
  - Stronger implementation, introducing containers
  - Opt-in for Apple’s own binaries and apps
  - Mandatory for Mac App Store apps (but not for DMG based)

- Far stronger still in iOS
  - Mandatory for all third party applications
  - Evolved beyond MacOS implementation

- From Jonathan Levin (link Ch 2b)

# Permissions

- Since iOS 7, app needs permission from user to access
  - Media
  - Microphone
  - Address book

# Protecting Information with Data-at-Rest Encryption

- All file system data encrypted with AES
  - Filesystem key generated on first boot
    - Stored on block 1 of NAND flash storage
- Device decrypts filesystem on bootup
- Filesystem is only encrypted at rest
- Remote wipe erases the key

# Data Protection API

- Can encrypt individual files and keychain items
- Key derived from passcode
- Encrypted items are inaccessible when device is locked

# Exploit Mitigations

# Write or Execute (W^X)

- Memory pages cannot be both writable and executable at the same time
  - Implemented with ARM's Execute Never (XN) feature
  - Pages marked as writable cannot be later reverted to executable
- Similar to Data Execution Protection (DEP) in Windows, Linux, and Mac OS X

# Return-Oriented Programming

- Bypasses non-executable memory
- Injection contains only addresses
- Pointing to fragments of code
- Exploit is built from these fragments

# ASLR

- Address Space Layout Randomization
  - Code location randomized
  - Attacker cannot find the injected code to run it
  - Makes ROP chains more difficult to use

# ASLR Weaknesses

- Before iOS 5, dynamic linker was not relocated
  - Memory disclosure bugs
    - Can be used to improve exploits
- Apps compiled with PIE (Position Independent Execution) can use full ASLR
  - All memory regions randomized
- Other apps put base binary and dynamic linker at a fixed address

# Stack Smashing

- Uses ***stack canaries***
  - Pseudorandom values on stack
  - Buffer overflow attacks overwrite the canary values
    - And terminate the app

# iOS Apps

# Three Groups of Apps

- Standard native apps
- Browser-based apps
- Hybrid apps

# Standard Native Apps

- Most common type
- Written in Objective-C or Swift
- Compiled to native code
- Linked against iOS SDK and Cocoa Touch frameworks

# Browser-Based Apps

- Render in iOS web views
- Loaded via mobileSafari
- Use HTML, JavaScript, and CSS
- Secure them like Web apps

# Hybrid Apps

- Deployed with a native wrapper
  - Used to display browser-based apps
- Mobile Enterprise App Platform deployment

# Distribution of iOS Apps

- App Store
  - Need an Apple Developer account
  - Apps signed with a developer certificate can run on up to 100 iOS devices for testing
  - App store approval has manual and automated tests
  - Blocks malicious apps

# Distribution of iOS Apps

- Enterprise Distribution
  - Organizations can develop and distribute custom apps in-house
  - Apps signed with enterprise developer certificate can run on any number of devices
  - Apple screens developers entering this program
  - Must have a legitimate business and a Dun and Bradsheet number

# Abuse of Certificates

- An expired enterprise developer certificate was used
- By changing system date back to the past
  - To run a Game Boy advanced emulator
  - For Pangu jailbreak

# App Structure

- IPA archive is a Zip archive containing
  - Payload
  - Payload/Application.app
    - Data, compiled code, and resources
- iTunesArtwork
  - Icon
- iTunesMetadata.plist
  - Developer's name, copyright info

# App Permissions

- Before iOS 6, every app in the App Store had access to
  - Contact, photos, other sensitive data

# Data Classes in iOS 6

- Location services
- Contacts
- Calendar
- Photos
- Reminders
- Microphone access
- Motion activity
- Bluetooth access
- Social media data

# Privacy Prompt

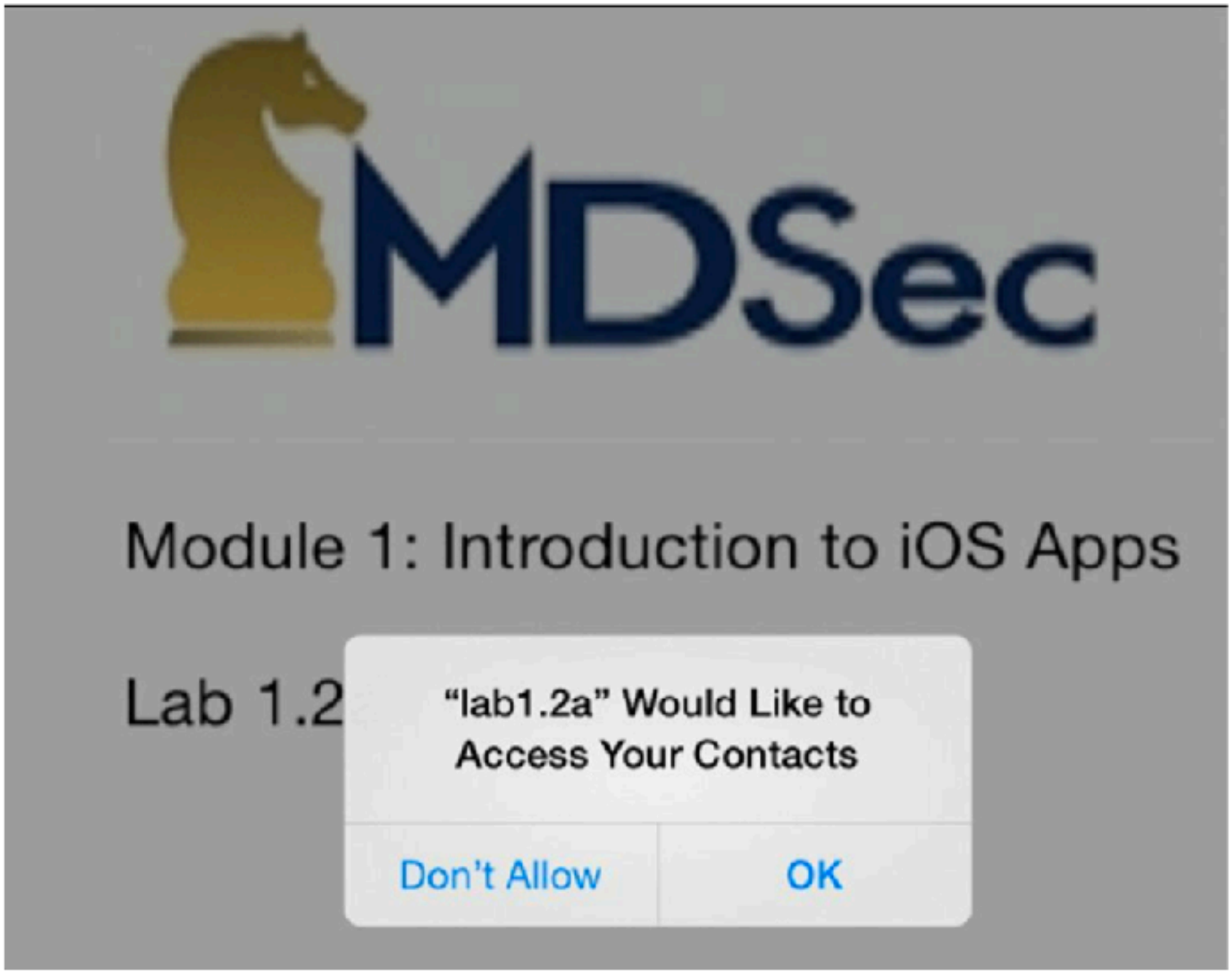
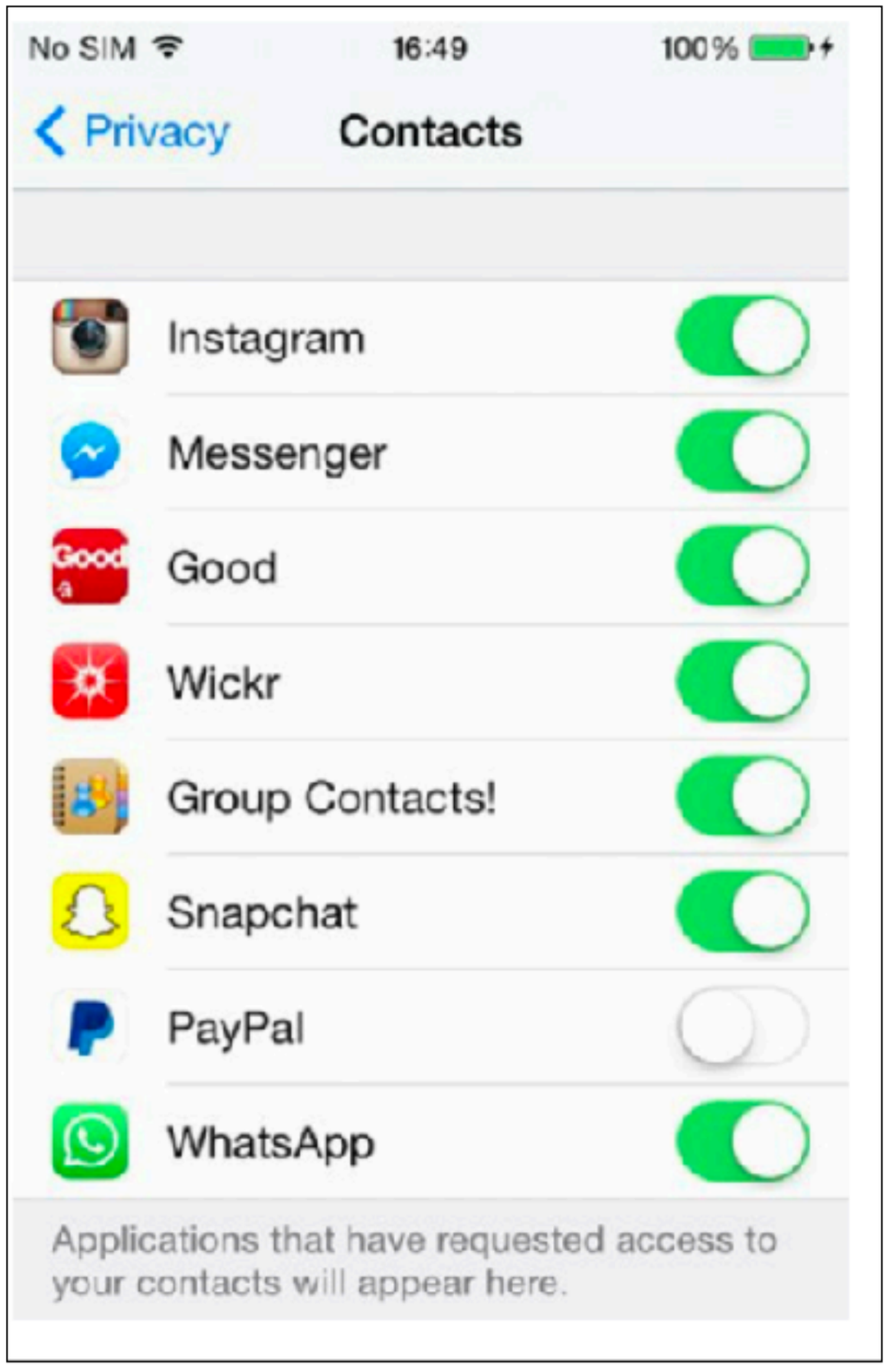



**Figure 2.2** The user sees this privacy prompt when an application tries to access the address book.


No SIM
16:49
100%
Privacy
Contacts
Instagram
Messenger
Good
Wickr
Group Contacts!
Snapchat
PayPal
WhatsApp
Applications that have requested access to your contacts will appear here.

# iOS 8 Location Information

- Three possible values: app is
  - Never allowed access to location information
  - Allowed access only while the app is in the foreground and in use
  - Always allowed access to location information

# Jailbreaking Explained

# Reasons for Jailbreaking

- Get apps from unauthorized marketplaces like Cydia
- Piracy
- Access to restricted functions like tethering

# Risks

- Weakens security of OS
  - Allow unsigned code to run
- Most iOS malware only runs on jailbroken phones
  - **iKee** - first iPhone worm, rickrolled phones using default password
  - **iKee.B** - Botted phones, phished Dutch users

# iOS Malware Campaign "Unflod Baby Panda"

*SektionEins GmbH* — 2014-04-18 10:00

SektionEins did a quick and dirty analysis of Unfold.dylib which is part of an iOS malware campaign targetting jailbroken iPhones.

- Chinese origin
- Only on jailbroken phones
- Hooked functions to steal AppleID and password

# Types of Jailbreaks

- **Untethered** - persists across reboots
- **Tethered** - requires a computer to start the phone; otherwise you get Recovery Mode
- **Semi-tethered** - requires a computer to start into jailbroken state, booting without the computer ends up in non-jailbroken state

# Jailbreakme v3 Saffron

- Simply visit a Web site hosting a PDF file
- Works for iOS before 4.3.4
- Uses:
  - Integer signedness issue to gain code execution
  - ROP payload
  - Type confusion vulnerability

# evasi0n Jailbreak

- Worked for iOS 6.0 - 6.1.2
- No memory corruption
- Used bypasses and logic bugs
  - Lockdownd service allowed file permissions to be changed
  - USB driver allowed arbitrary functions to be called

# Building a Test Environment

# Accessing the Phone

- After jailbreaking
- install OpenSSH in Cydia
- Connect via Wi-Fi or USB
- Default credentials are
  - **mobile** / **alpine**
  - **root** / **alpine**

# Toolkit

- **Cydia** -- open app store
- **BigBoss Recommended Tools**
  - Command-line UNIX tools
  - Including **apt**

# Apple's CC Tools

- Tools to parse, assemble, and link Mach-O binaries
    - File format for iOS and OSX apps
- Part of the iOS and OS X development toolchain
- Run on OS X or Linux

# Apple's CC Tools

- **otool**
  - Object file-displaying tool
  - All-purpose tool for Mach-O binary analysis
  - Reveals class and method names
  - Lists libraries, symbols
  - Shows header information and load commands

# Apple's CC Tools

- **nm**
  - Displays symbol table of a binary or object file
- **lipo**
  - Can combine or remove architecture types from an app

# Debuggers

- **gdb**
  - Cydia's version doesn't work well on modern iOS versions
  - Radare's version is better

# Code Signing

- **codesign**
  - Apple's binary-signing tool
  - Can also display signatures
- **ldid**
  - Saurik's code-signer

# Installipa

- Normal app installation uses **installd** service
  - Verifies code signature
- **ipainstaller**
  - Can install unsigned apps on jailbroken devices

# Exploring the Filesystem

- Jailbroken devices allow full access
- Unjailbroken devices allow access to portions of the filesystem, including
  - Sandboxed area where apps are installed
  - Must pair to a computer over USB first
  - Use apps like **iExplorer** or **iFunBox**

# On a Jailbroken Device

**Table 2.1** Interesting Filesystem Locations

| DIRECTORY | DESCRIPTION |
|---|---|
| `/Applications` | **System applications** |
| `/var/mobile/Applications` | **Third-party applications** |
| `/private/var/mobile/Library/Voicemail` | **Voicemails** |
| `/private/var/mobile/Library/SMS` | **SMS data** |
| `/private/var/mobile/Media/DCIM` | **Photos** |
| `/private/var/mobile/Media/Videos` | **Videos** |
| `/var/mobile/Library/AddressBook/AddressBook .sqlitedb` | **Contacts database** |

# Property Lists

- **.plist** files
  - Binary format similar to XML
  - Stores serialized objects and key-value pairs
- **plutil** can convert **plists** to XML

# Other Files

- Binary Cookies
  - From Web pages
  - Can be opened with **BinaryCookieReader.py**
- SQLite Databases


Kahoot!