Faux Disk Encryption
Realities of Secure Storage on Mobile Devices

Daniel A. Mayer  @DanlAMayer
Drew Suarez  @utkan0s
Who we are

Daniel Mayer
Principal Security Consultant with NCC Group
Developer of idbtool.com, iOS pentesting tool

Drew Suarez
Senior Security Consultant, Research Director with NCC Group
CyanogenMod (OSS) Device bringup / Wiki

NCC Group
UK Headquarters, Worldwide Offices
Software Escrow, Testing, Domain Services
Outline

1. Introduction

2. Secure Storage on iOS

3. Secure Storage on Android

4. Where does this leave us?
Apps Dominate Mobile

**Traditional**
- All data stored on server
- Tight controls

**Mobile**
- Data stored on device
- Difficult to control

---

Apps Continue to Dominate the Mobile Web

<table>
<thead>
<tr>
<th>Year</th>
<th>Mobile Web</th>
<th>Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013</td>
<td>20%</td>
<td>80%</td>
</tr>
<tr>
<td>2014</td>
<td>14%</td>
<td>86%</td>
</tr>
</tbody>
</table>

Percentage of time spent

Source: Flurry Analytics
Challenge: Device Mobility

Data is being carried around

Devices prone to loss/theft [1]

1.4 million phones lost
3.1 million stolen
(US, 2013)
Challenge: Data Accessibility

Local Data
- Data cached and stored on the device

Credentials
- Usernames / passwords
- Access tokens
Challenge: Usability

Known security controls reduce usability

MORDAC, THE PREVENTER OF INFORMATION SERVICES.

SECURITY IS MORE IMPORTANT THAN USABILITY.

IN A PERFECT WORLD, NO ONE WOULD BE ABLE TO USE ANYTHING.

To complete the log-in procedure, stare directly at the sun.

www.dilbert.com

© 2007 Scott Adams, Inc. Dist. by UFS, Inc.
There is no absolute security

Remote Attacker
There is no absolute security

Coffee Shop Attacker
There is no absolute security

Casual Thief
There is no absolute security

Targeted Attacks
There is no absolute security
There is no absolute security
Mobile Data Security
A Word on Full-Disk Encryption

Encrypts files stored on the file-system
  - Transparently decrypted when read
  - Transparently encrypted when written

Protection only when device is turned off
  - In combination with strong passcode!

Need more fine-grained control
Secure Data Storage
...on iOS
iOS Boot/App signing

Apple Hardware + Apple Software

Boot Chain Completely Signed
  Hardware root of trust (ROM) contains Apple CA

iOS Updates
  Signed by Apple
  Downgrades not allowed

App Signing
  All code running on iOS must be signed by Apple
Bootstrapping Encryption

Device Passcode
- Not stored on device
- Derive encryption key when entered
- Wipe key when device is locked

Problems
- Users choose weak passcodes [1]
- Prone to offline brute-force attacks
Hardware Root of Trust

Tie Encryption to a Device
- Unique encryption key per device
- Cannot be read by operating system
- Can “ask” Secure Enclave to decrypt

Hardware Controls
- Enforce brute-force controls
- Enforce device-wipe
iOS Encryption Hierarchy

- File System Key
- Class Keys
- File Meta Data
- File Key

PBKDF2
Passcode Key
Hardware Key
iOS Encryption Hierarchy

- Hardware Key
- Passcode Key (PBKDF2)
- Class Keys
- File System Key
- File Meta Data
- File Key

Erase this iPad?
All your content and settings will be erased. An erased iPad cannot be located or tracked.

Cancel | Erase
iOS Encryption Hierarchy

- **Class Keys**
  - **Passcode Key**
  - **Hardware Key**

PBKDF2

- **File Meta Data**
  - **File Key**

Erase this iPad?
All your content and settings will be erased. An erased iPad cannot be located or tracked.

Cancel | Erase
iOS Encryption Hierarchy

- Hardware Key
- Passcode Key
- Class Keys
iOS Encryption Hierarchy

- Hardware Key
- Passcode Key

- `NSFileProtectionNone`
- `NSFileProtectionComplete UntilFirstUserAuthentication`
- `NSFileProtectionComplete`
iOS Encryption Hierarchy

- **Hardware Key**
  - **Passcode Key**

- **NSFileProtectionNone**
  - **NSFileProtectionComplete UntilFirstUserAuthentication**
  - **NSFileProtectionComplete**

[Image of iOS Encryption Hierarchy diagram with hardware key and passcode key, showing different protection levels.]
iOS Keychain

Structured Data Store
- Lives in SQLite database
- Entries individually encrypted

Main Criticism
- Data not deleted when app is uninstalled!
# Keychain

<table>
<thead>
<tr>
<th>File Protection (NSFileProtection)</th>
<th>Keychain Class (kSecAttrAccessible)</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Always</td>
<td>No protection.</td>
</tr>
<tr>
<td>UntilFirstUserAuthentication</td>
<td>AfterFirstUnlock</td>
<td>Protected from boot until user unlocks.</td>
</tr>
<tr>
<td>Complete</td>
<td>WhenUnlocked</td>
<td>Protected when device is locked.</td>
</tr>
<tr>
<td>N/A</td>
<td>WhenPasscodeSet</td>
<td>Only store if passcode is set.</td>
</tr>
</tbody>
</table>

Daniel A. Mayer, Drew Suarez  -  Realities of Secure Storage on Mobile Devices
Usability vs. Security

Data Accessibility
Some data must be accessible when device is in use
Tackling Usability

**TouchID**

Usability feature
Controlled by Secure Enclave
Encourages users to set passcode
Simply protects passcode-based key

https://www.youtube.com/watch?v=vl3OvT4b-sA
Advanced Controls

User Presence for Keychain
- Requires users to enter Passcode (or TouchID)

Local Authentication
- OS-level API
- Not tied-in with crypto
- Bypassable when jailbroken [5]
- Use Keychain User Presence instead
Security Threats - Jailbroken

Jailbreaks Do

- Allow execution of unsigned code
- Disable some OS-level protections

Jailbreaks Don’t

- Disable Sandboxing for App Store apps

What About Secure Storage?

- Passcode may prevent public jailbreaks
- Access to all non-protected data

http://idbtool.com
Security Threats - Non-Jailbroken

Malicious Applications
- Asking for access to personal data
- Apps attacking other apps via IPC mechanisms

Evil Maid-Style Attacks
- Jailbreak device
- Backdoor OS / App
Secure Data Storage
...on Android
## Evolution of Android Security

<table>
<thead>
<tr>
<th>Feature</th>
<th>4.0</th>
<th>4.1</th>
<th>4.2</th>
<th>4.3</th>
<th>4.4</th>
<th>5.x</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASLR</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>DEP/PIE</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restricted logcat</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restricted adb</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manifest Export Security</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Secure Random from OpenSSL</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Untrusted Application Malware Scanning</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>SELinux (Permissive)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SELinux (Enforcing)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>KeyStore Hidden Keys*</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>No setuid/getuid, nosuid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text Relocation Protection</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dm-verity</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>TEE signing of KEK</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>forceencrypt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X*</td>
</tr>
</tbody>
</table>
Adoption of Android Security

Android OS versions

[Image: Bar chart showing the adoption rates of different Android OS versions.]

[6]
Flash back to iOS Adoption..
Impact on Application Devs

Developers face different platform versions and security APIs
- Code complexity and inconsistent behavior
- Access to more secure functionality is not available for all users

Security improvements available via latest version
- Complicated problem of an OTA update process
How Android Encryption Works

- **AES CBC Mode**
- **ESSIV: SHA256**
- **Disk Sectors**

[8,9]
How Android Encryption Works

- AES CBC Mode
- KEK+IV
- DEK
- ESSIV: SHA256
- Disk Sectors
- Stored on Partition

Encrypted DEK

[8,9]
How Android Encryption Works

1. Password Key
   - PBKDF2 or scrypt
2. KEK+IV
3. DEK
   - AES CBC Mode
   - ESSIV: SHA256
4. Disk Sectors
   - Stored on Partition

AES CBC Mode

[8,9]
How Android Encryption Works

- PBKDF2 or scrypt
- Password Key
- Signed Password Key
- RSA 2048 Signature
- KEK+IV
- Encrypted DEK
- Stored on Partition
- DEK
- AES CBC Mode
- ESSIV: SHA256
- Disk Sectors

Discarded Text (not part of the diagram): RSA 2048 Signature

[8,9]
How Android Encryption Works

This protection only covers the userdata partition

Crypto footer
Carved out of end of userdata partition (-16kB)
Sometimes there is a dedicated partition
Master key stored here encrypted by the KEK

LUKS-ish but not quite.
Footer can only hold one decryption key
Android Credential Storage

System Credential Store allows for storage of
  VPN Keys
  WiFi
  Asymmetric keys

Encrypted by key derived from user's passcode

Can be hardware backed
  Private keys non-extractable, even as root
  Requires use of device in attack

Issues with KeyStore
  Inconsistent protections available to developers
  Unclear documentation and erratic behavior causes keys to be wiped (fixed in 5.0)
  Improved with Marshmallow
A look at Marshmallow changes

**scrypt hashing of unlock passcode values**
- Replaces weaker SHA-1/MD5 hash concatenation

**Additional KeyStore improvements**
- Added support to store symmetrical keys (without private API)
- Documented and refined KeyStore wipe behavior
- Additional properties for keys
  - Prevent unsafe modes (fixed IV’s, ECB mode, etc)
  - Explicitly define a key type
Nexus Imprint, etc

Allows for a more complex passwords
Secure payments, unlock capabilities
Stored securely in TEE
Sets defined standards for other OEMs
Google & OEMs

Wild inconsistencies among devices
- Boot loader security
- Hardware backed crypto storage
- TEE / TrustZone
- Boot image type

Different OEMs offer different protection schemes
- eMMC write protection
- Boot image signature verification
- Locked, locked but unlockable, permissive by default

Difficult problem to solve
- Challenging for Google to enforce consistent protections on the OEMs
- Apple has a distinct advantage in controlling the whole stack
Importance of Boot Security

A typical vulnerable boot chain of trust

bootloader → boot (kernel) → /system → /data

Vulnerable! (without signing)
Vulnerable! (without dm-verity)
Download Mode

Samsung specific boot loader interface for their Android devices

Internally, Samsung uses a tool called ODIN
  Interacts with the device and flash firmware images
  Check out heimdall if you want a cross-platform, open source version

Overly permissive!

Most devices allow direct write access!
  Except for a few US carrier protected models
  (Boot image signature verification)

[11, 12]
lk (little kernel) Bootloader

Issues with lk used on many devices

“Fastboot boot command bypasses signature verification (CVE-2014-4325)” [13]

“Incomplete signature parsing during boot image authentication leads to signature forgery (CVE-2014-0973)” [14]

Bootable partition named laf found on many LG devices

Communication via Send_Command binary (Windows)
  Also available as python script for all platforms
  Drops into a root shell
  Flash new images from shell

Fixed? Not quite.
  /dev/block/mmcblk0p1 - protected
  /dev/block/mmcblk0 - not protected
  dd + seek :)

[16]
Let’s revisit:
“FDE protects data when device is turned off”
Mobile “Evil Maid” Attacks

Exploit permissive bootloader
- Flash custom boot image
- Backdoor in kernel in image
- < 2 minutes (including reboots!)

Give device back to user

Profit!
- Get encryption key…
- …or data exfiltration
- …or shells

ROSIE!
Dev Step 1: Flash Recovery

For more info on recovery… [19]
Dev Step 2: Backdoor the Kernel

Arrow keys navigate the menu. <Enter> selects submenus -->. Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes, <M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search.

Legend: [*] built-in [ ] excluded <M> module < > module capable

Backdoor kernel module to bypass FDE
<> Analog Devices Digital Potentiometers
<> Atmel AT32/AT91 PWM support
<> Dummy IRQ handler
<> Sensable PHANToM (PCI)
<> Parallel Trace Interface for MIPI P1149.7 cJTAG standard
<> SGI IOC4 Base I0 support

<Select> < Exit > < Help > < Save > < Load >
Dev Step 3: Test Exploit

1. Compile backdoored kernel
2. Create boot image
3. Flash boot image via recovery
4. Reboot and test
The Attack: Review

Possible on a number of OEM devices

This is not a new problem

Google provides mechanisms to prevent this

Similar attack possible in iOS, but requires jailbreak
A penny for your thoughts…?

- Secure configurations by default!
- Responsible bootloader unlock capabilities
- Clearly documented security guarantees
- Consistency among OEM partners
“Alternatives” to Platform Security
No Password? No Problem!? | Daniel A. Mayer, Drew Suarez - Realities of Secure Storage on Mobile Devices

What if users may not have set passcodes?

**Custom App Sandboxes**
- Add passcode to app
- Derive encryption key
- Encrypt data
- Wipe key!

**Challenges**
- Crypto is hard! [20]
- Not hardware backed, no brute-force protection
Online Apps

No Offline Storage
- Does data need to be offline?
- Consider storing server-side

Usability
- Login each time
- Long-lived token, back to storage problem
Where does this leave us?
Best Practices for Users

**General**
- Set a (strong) passcode!
- Use the latest OS available for your hardware

**iOS**
- Enable (remote) wipe

**Android**
- Choose your phone wisely
- Encrypt your device
Best Practices for Developers

General
Determine if data has to be stored locally
Case by case situation…

Android
Relying on platform security is challenging
Discussion: supporting old versions of Android

iOS
Use protection class that requires passcode
Warn user when no passcode is set
The Road Ahead
Usability

For Users
  Beyond Passwords
  Biometrics

For Developers
  Consistency in platform
  With sane, documented defaults
1. Security controls should be balanced with data sensitivity and threat model.

2. Protect data until access is actually needed.

3. Secure storage relies on the entire stack being secured.
References

[1] Consumer Reports. Smart phone thefts rose to 3.1 million last year, Consumer Reports finds, May 2014
[18] https://twrp.me/
¡Gracias!

Questions?

Daniel A. Mayer  @DanlAMayer
Drew Suarez  @utkan0s

Slidedeck: https://speakerdeck.com/utkanosos