The State of Mobile Security

Manuel Egele
maeg@cs.ucsb.edu
Computer Security Group at University of California Santa Barbara

DIMVA 2012, Thu. July 26th 2012
Smartphones are Ubiquitous

If not yet they are certainly on the way to get there:

365 million iDevices in total

400 million Android devices in total
Mobile Platforms

- Smartphones and tablets are ubiquitous
- Have access to (sensitive) user/company data
  - Addressbook
  - Calendar
  - etc.
- Sensors (e.g., GPS, camera, microphone) provide data that is available to third-party application developers
- App Stores drive the economy behind mobile applications
App Stores - Economy

Apple AppStore
- 650,000 Apps
- 400 million store accounts
- 30 billion downloads
- $5 billion to developers / $2.5 billion in 2011 alone

Google Play
- 490,000 Apps
- Top 200 Apps in Google Play $679,000 per day in Jan. 2012
- ~ $247 million / year
Where There's Money, ...
Malware Reached Mobile Platforms

- Smartphones are targets b/c they store personal/sensitive information
  - Address books, GPS coordinates
  - Weakly protected online banking credentials
  - Zitmo intercepts mobile transaction numbers (mTAN)
  - Monetization through premium SMS and calls

- Different ecosystem than commodity computers
  (e.g., different level of control of the user over the device)
  - Root exploits against iOS and Android
Current State of Affairs

- Millions of users (potential victims)
- Easy access to data stored on their devices
- Existing mechanisms to monetize this data
- Bad guys that combine the above
- We need better security solutions
Overview

• Ubiquity of mobile systems
• Why mobile systems security
• Different security/protection approaches
  – iOS – Apple AppStore
  – Android – Google Play Store
• Static vs. Dynamic analysis for protection
• Challenges on Android systems
• Static analysis of iOS apps to detect privacy violations
• Recent developments / Lessons learned
• Summary
Security Models (iOS – AppStore)

- Developer pays $99/year for iPhone developer program
- Apple App Store – 650k apps available, 30bn downloaded
- Non-public vetting process for each submitted application
  - Probably a combination of static and dynamic analysis techniques
- Code signing and encryption
- Once the app is approved it is available on the AppStore
Security Models (iOS – Device)

• On app start-up OS loader decrypts the application and places the decrypted contents in memory (only!)
• Mandatory code signing:
  – No unsigned code will execute
  – Prevents self modifying code
  – Safari et al. have dynamic-codesigning entitlement
• Applications have unfettered access to most information on the device (noteworthy exceptions: GPS, SMS, Phone)
• Since iOS 4.3 ASLR
• All apps run as one user (i.e., mobile)
• Application sandboxing through MAC policy hooks
Security Models (Android – Google Play)

- 25$ registration fee – keeps spammers away

- Google Bouncer
  - Screens submitted applications for malicious code
  - Was circumvented (e.g., malicious apps on the Play store for weeks, download & execute of additional malicious code)
  - Was hacked (e.g., Oberheide and Miller got a shell on Bouncer, could inspect the analysis enviroment)

- Mandatory application signing
  - Self signed certificates accepted
Security Models (Android – Device)

- Each application is run as separate Linux user
- Install-time permissions for applications
  - Access address book or GPS location
  - Open network connections
  - Send & receive/intercept SMS
  - Permissions enforced at the kernel level → to circumvent them, the attacker needs to exploit the kernel
- Common defensive techniques:
  - ProPolice (stack buffer overrun protection), Android 1.5+
  - Format string vulnerability protection, Android 2.3+
  - Address space layout randomization, Android 4.0+
  - Position independent executables, Android 4.1+
Security Models (Android – Device)

- CyanogenMod – aftermarket Android firmware
  - Revocation of install-time permissions
  - Apps might crash if they do not handle the changes gracefully
  - Faking data patch gives apps fake data from address book, location, etc.
Known Malicious Apps (iOS)

Apps that have been retracted from Apple AppStore
- Torch – Flashlight app that enables tethering (good for user, bad for network operators)
- Path, Gowalla et al. upload address book to remote servers
- Storm8 apps leaked phone numbers
- Find & Call – steals address book, sends text messages to contacts with spoofed sender number
- POC by Charlie Miller to circumvent mandatory code signing ➔ arbitrary code execution
- jailbreakme.com performs root exploit ➔ Drive-by download
Known Malicious Apps (Android)

Malicious apps on Google Play and third party markets

- Repackaged popular titles including malicious functionality often appear in third party markets
- If User needs to enable installation from untrusted sources and agree to the permissions at install time ➔ not a drive-by download
- Find & Call – steals address book, sends text messages to contacts (yes this is truly multi-platform!)
Good day!

System is in process of beta-testing. In result of failure of one of the components there is a spontaneous sending of inviting SMS messages. This bug is in process of fixing. SMS are sent by the system, that is why it won't affect your mobile account.
When is Data Transmission Legitimate?

• Find & Call: “The Find and Call app has been removed from the App Store due to its unauthorized use of users’ address book data, a violation of App Store guidelines,” Apple spokesperson Trudy Muller told Wired.
• Text messages sent from backend server (iOS does not expose APIs to send text messages)
• Find & Call only caught b/c the app was advertising itself
• Path et al. do not use the address book information in similar obviously nefarious ways

Q: How can we tell such cases apart?
Malware Growth on Commodity Systems

Let's take a step back and take a look at the past development of malicious software on commodity systems.
Malware Growth on Commodity Systems

Number of new signatures

Period

2002 2003 2004 2005 2006 2007 2008 2009

20,254 19,159 74,981 113,081 167,069 708,742 1,691,323 2,895,802
Motivation for Mobile Malware Analysis

- Why don't we implement virus scanners on mobile devices?
  - Sandboxing prevents access to other applications
- Potentially huge number of malware samples
  - Generating signatures manually is slow and does not scale
  - Sophisticated targeted attacks → no sample to create a signature
- Efficient and scalable way to answer:

  Q: *Is an unknown piece of code malicious or benign?*
Static vs. Dynamic Security Measures

• Static measures can be applied pre-launch
  – One time effort
  – All users benefit immediately
  – No performance overhead at runtime
  – Challenges for static analysis (obfuscation, dynamically loaded code, etc.)
Static vs. Dynamic Security Measures

• Dynamic security measures
  – Can be more precise than static measures
  – Incomplete path coverage \(\Rightarrow\) pre-launch analysis is incomplete
  – Mobile apps are inherently user driven \(\Rightarrow\) to increase coverage interaction with the application is required
  – Often each device has to perform its own analysis during execution
  – Performance overhead during execution
Why Bother with Static Analysis

- Apps on smartphones run in restricted environments
  - Sandboxed in Android and iOS
  - Permission model in Android
  - No side loading on iOS
  - No self modifying or dynamically loaded code in iOS
  - No self modifying code in Dalvik

→ less freedom for attackers
Static Analysis Prerequisites

Many static analysis approaches require access to control flow graphs (CFG) and call graphs (CG):

• How can we extract CGs and CFGs for Android apps?
• How can we extract CGs and CFGs for iOS apps?
Overview

- Ubiquity of mobile systems
- Why mobile systems security
- Different security/protection approaches
  - iOS – Apple AppStore
  - Android – Google Play Store
- Static vs. Dynamic analysis for protection
- Challenges on Android systems
- Static analysis of iOS apps to detect privacy violations
- Recent developments / Lessons learned
- Summary
CFG for Android

Easy case:

.class final Lcom/admob/android/ads/c;
.method public constructor <init>(Ljava/lang/String;...)V
  new-instance v0, Ljava/net/URL;
  invoke-direct {v0, p1}, Ljava/net/URL;-><init>(Ljava/lang/String;)V
Challenges for CFG (Android)

Interfaces and inheritance:

```java
.class public interface abstract Lcom/admob/android/ads/n;
.method public abstract d()Ljava/lang/String;
...
.class public final Lcom/admob/android/ads/m;
.method public final a(Lcom/admob/android/ads/n;)V
   invoke-interface {p1}, Lcom/admob/android/ads/n;d()Ljava/lang/String;
```

To determine what methods might be called by the invoke, we need to understand the possible types of `a`'s argument. To determine these types, we have to find all call-sites to `m.a()`. Want more challenges? Reflective calls
More Fun Analyzing Android Apps

Junk byte injection

Original Bytecode

entry point of the method

address space

Obfuscated Bytecode

entry point of the method

address space

ins ins ins ins ins ins ins ins
goto +2

fill-array-data-payload

ins ins ins ins ins ins ins ins

© dexlabs.org
More Fun Analyzing Android Apps cont.

```java
Method 3131 (0xc3b):
private java.lang.String
org.dexlabs.poc.dexdropper.DropActivity.execute(
    java.lang.String p0)

15  fill-array-data
    v0, v0, loc_35B1E
    v0, arraydata_35B16

# CODE XREF: DropActivity_down_exec@V@L+6C1]

# DATA XREF: DropActivity_exec@L@L+41]

# Array definition; 198 elements, each 1 bytes

new-instance
v5, <t: StringBuilder> # Array Contents
{v5}, <void StringBuilder.init()> (this), <ref DropActivity.getApplicationContext()> @_def_DropActivity_getApplicationContext@L>

v6, <ref Context.getFilesDir()> @_def_Context_getFilesDir@L>

{v5, v6}, <ref StringBuilder.append(ref)> @_def_SringBuilder_append@L>

v5
v6, aTemp "temp"

{v5, v6}, <ref StringBuilder.append(ref)> @unk_D93C>

v5
v5
{v5}, <ref StringBuilder.toString()> @_def_StringBuilder_toString@L>

v5
v5
v5
v6, 0
{p0, v5, v6}, <ref DexFile.loadDex(ref, ref, int)> @_def_DexFile_loadDex@L>

v4
v5, aBad "$bad"
{v5}, <ref DropActivity.getClassLoader()> @_def_DropActivity_getClassLoader@L>

v6
{v4, v5, v6}, <ref DexFile.loadClass(ref, ref)> @_def_DexFile_loadClass@L>
```
“An intent is an abstract description of an operation to be performed.”

“An activity is a single, focused thing that the user can do. Almost all activities interact with the user,...” often a screen/view

Use `startActivity(Intent)` upon a click event to switch to a new activity
Android Intents & Activities cont.
Enough of Android (For Now)

Let's look under the hood of iOS

- Signatures do not scale
- Behavior-based detection of apps that access privacy sensitive information and transmit this information over the Internet without user intervention or consent
- Model this functionality as a data-flow problem

Challenges

- Apps are binary only → binary analysis
- Object-oriented concepts of Obj-C
iOS – App Analysis

1. Extract control flow graph (CFG) from binary application
2. Identify sources of sensitive information and network communication sinks
   - Perform reachability analysis between sources and sinks
3. Data flow analysis on detected paths
Static Analysis – IDA Pro

The file is encrypted. The disassembly of it will likely be useless. Do you want to continue?

No  Yes
Background (iOS & DRM)

- App Store apps are encrypted and digitally signed by Apple
- Loader verifies signature and performs decryption in memory
- Decrypting App Store apps:
  - Attach with debugger while app is running
  - Dump decrypted memory regions
  - Reassemble binary, toggle encrypted flag
Static Analysis (Call Graph)

IDA Pro state of the art disassembler for binary analysis
call graph for “Bomb”

objc_msgSend
iOS – App Analysis (CFG)

• Most iOS apps are written in Objective-C
• Cornerstone: `objc_msgSend` dispatch function
• Task: Resolve type of receiver and value of selector for `objc_msgSend` calls
  – Backwards slicing
  – Forward propagation of constants and types
• Result: Inter and intra procedural CFG is constructed from successfully resolved `objc_msgSend` calls
Background (objc_msgSend)

• `objc_msgSend` dynamic dispatch function

• Arguments:
  – Receiver (Object)
  – Selector (Name of method, string)
  – Arguments (vararg)

• Method look-up:
  – Dynamically traverses class hierarchy
  – Calls the method denoted by selector
  → All information readily available at runtime, but challenging to do statically

• Similar to reflection in Java, Obj-C uses only reflection
iOS – App Analysis (Class Hierarchy)

• Problem: Multiple candidate types for receiver
• Class hierarchy is extracted from application and libraries
• All possible candidate types are inspected whether they implement a method
• If only one candidate implements the method that type is chosen for the receiver
iOS – App Analysis (CFG)

Novel object-oriented analysis approach for Obj-C binaries based on two key techniques:

(1) Resolve *type* of receiver and *value* of selector for `objc_msgSend` calls
   (a) Backwards slicing
   (b) Forward propagation of constants and types

(2) Multiple candidate types for receiver  ➔ class hierarchy

Result: Inter and intra procedural CFG is constructed from successfully resolved `objc_msgSend` calls
Example ObjC to ASM

1  LDR    R0, =off_24C58
2  LDR    R1, =off_247F4
3  LDR    R0, [R0]
4  LDR    R1, [R1]
5  BLX    _objc_msgSend
6  LDR    R1, =off_247F0
7  LDR    R1, [R1]
8  BLX    _objc_msgSend
9  ADDS   R4, R0, #0
...   this.score
10 LDR    R3, [R6,R3]
11 STR    R3, [SP,#0x40+var_40]
12 ADDS   R3, R4, #0
...   ... _objc_msgSend
13 BLX    _objc_msgSend
(fmt: "uniqueid=%@&scores=%d")
14 BLX    _objc_msgSend
...   ... _objc_msgSend

iOS Apps – Finding Privacy Leaks

• Based on inter and intra procedural CFG
• Reachability Analysis (find paths)
  – From interesting sources
  – To network sinks
• Data-flow analysis from source to sink
iOS Apps – Evaluation

• 1,407 Applications (825 from App Store, 582 from Cydia)

• Pervasive ad and statistic libraries:
  – 772 Apps (55%) contain at least one such library
  – Leak UDIDs, GPS coordinates, etc.
Ad and Statistic Libraries

• 82% use AdMob (Google)
• Transmit UDID and AppID on start-up and ad request
• Ad company can build detailed usage profiles
  – Gets info from all Apps using the ad library
• Problem: Location-based Apps
  – Access to GPS is granted per App
  – Libraries linked into location based apps have access to GPS too
• UDIDs cannot be linked to a person directly, but...
Is Leaking UDIDs a Problem?

• UDIDs cannot be linked to a person directly

• But: Combine UDID with additional information e.g.,
  – Google App can link UDID to a Google account
  – Social networking app get user's profile (often name)

• Linking ICC-ID with UDID is trivial
  – 114,000 iPad 3G users
Is Leaking UDIDs a Problem?

June 2010

AT&T Inc. acknowledged Wednesday that a security hole in its website had exposed its iPad customers' email addresses, a breach that highlights how corporations still have problems protecting private information.

A small group of computer experts that calls itself Goatse Security claimed responsibility for the intrusion, saying the group had exploited an opening in AT&T’s website to find numbers that identify iPads connected to AT&T’s mobile network.

Those numbers allowed the group to uncover 114,000 email addresses of thousands of iPad customers, including prominent officials in companies, politics and the military, the group said. Gawker Media LLC reported the breach Wednesday. It doesn’t appear any financial or billing information was made public.

89014104243220@nytimes.com
89014104243219@time.com
89014104243221@newscorp.com
89014104243315@hearst.com
89014104243315@dowjones.com
89014104243221@weinsteinco.com
89014104243315@bloomberg.net

Janet Robinson, CEO of NY Times
Ann Moore, CEO of Time Inc.
Chase Carey, President/COO of News Corp.
Cathie Black, President of Hearst Magazines
Les Hinton, CEO of Dow Jones
Harvey Weinstein, Co-Founder of Weinstein Co.
Michael Bloomberg, Founder of Bloomberg LP
## PiOS – Evaluation: Leaked Data

### Table

<table>
<thead>
<tr>
<th>Source</th>
<th>#App Store 825</th>
<th>#Cydia 582</th>
<th>Total 1407</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceID</td>
<td>170 (21%)</td>
<td>25(4%)</td>
<td>195(14%)</td>
</tr>
<tr>
<td>Location</td>
<td>35(4%)</td>
<td>1(0.2%)</td>
<td>36(3%)</td>
</tr>
<tr>
<td>Address book</td>
<td>4(0.5%)</td>
<td>1(0.2%)</td>
<td>5(0.4%)</td>
</tr>
<tr>
<td>Phone number</td>
<td>1(0.1%)</td>
<td>0(0%)</td>
<td>1(0.1%)</td>
</tr>
<tr>
<td>Safari history</td>
<td>0(0%)</td>
<td>1(0.2%)</td>
<td>1(0.1%)</td>
</tr>
<tr>
<td>Photos</td>
<td>0(0%)</td>
<td>1(0.2%)</td>
<td>1(0.1%)</td>
</tr>
</tbody>
</table>
PiOS – Evaluation: Case Studies

Address book contents:
- Apps have unrestricted access to the address book
- Facebook and Gowalla transmit the complete AB
- Feb. 2012: Mainstream media picks up this and similar cases
  ➔ Apple is changing policies and implements restrictions

Phone numbers:
- Nov. 2009 Apple removed all Storm8 titles from App Store
- Apps transmitted phone numbers (SBFormattedPhoneNumber)
- New versions don't have that code anymore
- Old version of “Vampires“ PiOS detected the privacy leak
- MogoRoad – Free version users get calls from telemarketers
Unauthorized iPhone And iPad Apps Leak Private Data Less Often Than Approved Ones

Users have learned over the last few years that Apple’s “walled garden” approach to third party apps isn’t quite as protective of their sensitive data as it might sound. More surprising, perhaps, is another revelation: that the popular unauthorized apps outside those walls tend to respect privacy better than the approved ones inside.

"Path" Would Like To Access Your Contacts
Not all apps recover successfully from having their Contacts access revoked.

Don't Allow  OK

A screenshot of the Contact privacy feature in the unofficial Cydia iOS app platform.

"In the wake of news that the iPhone app Path uploads users' entire contact lists without permission, Forbes di Systems Lab that aimed to analyze how and where iPhone apps transmit users' private data. Not only did the researchers potentially identify users and allow profiles to be built of their activities; they also discovered that program far less frequently than Apple's approved apps. The researchers ran their analysis on 1,407 free apps (PDF) on for instance, compared with only four percent of unauthorized apps."

100 of 179 comments loaded
PiOS – Evaluation: Case Studies

Address book contents:
- Apps have unrestricted access to the address book
- Facebook and Gowalla transmit the complete AB
- Feb. 2012: Mainstream media picks up this and similar cases
  ➔ Apple is changing policies and implements restrictions

Phone numbers:
- Nov. 2009 Apple removed all Storm8 titles from App Store
- Apps transmitted phone numbers ($SBFormattedPhoneNumber$)
- New versions don't have that code anymore
- Old version of "Vampires" PiOS detected the privacy leak
- MogoRoad – Free version users get calls from telemarketers
Overview

• Ubiquity of mobile systems
• Why mobile systems security
• Different security/protection approaches
  – iOS – Apple AppStore
  – Android – Google Play Store
• Static vs. Dynamic analysis for protection
• Challenges on Android systems
• Static analysis of iOS apps to detect privacy violations
• Recent developments / Lessons learned
• Summary
Lessons Learned

• Communicating privacy issues and raising awareness can be challenging

• Jailbroken iPhones are not necessarily less secure:
  – See PrivaCy application to opt out of ad tracking
  – Security patches for legacy systems
  – Experimental or research apps almost always require Jailbreak

• Would more fine grained permissions be helpful?
  – Users get tired of reading permission screens
Recent Developments

• Apple announced permissions for address book access for iOS 6
  – Will Apple come up with a working solution for permissions?

• Obfuscated Dalvik applications
  – Google advocates the use of ProGuard to protect IP – renaming class and method names to a.a.b() etc.
  – Recently: applications storing bytecode in arrays and jumping there – throws off linear sweep disassembler
Recent Developments cont.

Get in-app purchase for free on iOS and Apple Mac Store
- No Jailbreak required
- Install a trusted CA
- Install a trusted certificate for Apple's AppStore server
- Make DNS resolve the AppStore name to the fake AppStore
- Done
- Similar attack exists for the Mac Store on OS X

Interestingly, Apple immediately reacted and promised a fix with the next iOS update.

Developers + revenue vs. Privacy
Summary

- Mobile systems are ubiquitous
- Mobile systems implement new paradigms and security mechanisms
  - AppStores
  - Mandatory code signing
  - Permissions
- Static and dynamic analysis methods can be used to detect malware in mobile applications
Thanks for your attention

QUESTIONS?