Cellular Networks and Mobile Computing
COMS 6998-8, Spring 2012

Instructor: Li Erran Li
(lel2139@columbia.edu)

http://www.cs.columbia.edu/~coms6998-8/

Lecture 13: Mobile Privacy
Mobile Privacy

Data privacy
- Detecting and preventing privacy leaks
  - PiOS for iOS
  - TaintDroid for Android
- Stealthy information leaks through covert channels and prevention
  - Soundcomber
- Auditing to determine which files accessed after device loss
  - Keypad

Location privacy [Presented by Sameer Choudhary]
- Quantifying location privacy
PiOS:
Detecting Privacy Leaks in iOS Applications

Manuel EGELE, Christopher KRUEGEL, Engin KIRDA, Giovanni VIGNA
{maeg,chris,vigna}@cs.ucsb.edu, ek@ccs.neu.edu
Int. Secure Systems Lab, UCSB & TU Vienna & Northeastern University
Motivation

• App Store: 300k apps available, 10 billion apps downloaded
• iOS apps are created by third party developers
• “iPhone Developer License Agreement”
• States guidelines (e.g., user's privacy)
• Submitted binaries are scrutinized by Apple through a secret vetting process
• Apps passing the vetting process → App Store
• Cydia: repository for jailbroken devices
Motivation (cont.)

• Vetting process is not flawless
• ”Bad apps” make it to the App Store
• Apple removed several apps after they were available e.g.,
  – Flashlight (enables tethering w/o the network operator's consent)
  – Storm8 games harvested device phone numbers
  – MogoRoad collect phone numbers of free app users
→ Telemarketers called and offered the paid full version
Motivation (cont.)

• How big is the problem?
Overview

• Motivation

• Goals & Challenges

• PiOS Analysis
  – Extract CFG
  – Reach ability analysis
  – Data Flow analysis

• Evaluation

• Summary
Goals & Challenges

• Goals
  – Identify Apps that access privacy sensitive information and transmit this information over the Internet without user intervention or consent
  – Perform this analysis on a large body of Apps
  – Gain insight in how Apps handle privacy sensitive data

• Challenges
  – Apps are only available as binary executable
  – App Store Apps are encrypted
Approach

1. Extract control flow graph (CFG)
2. Identify sources of sensitive information and network communication sinks
   - Perform reachability analysis between sources and sinks
3. Data flow analysis on detected paths
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
- Cydia Apps are not encrypted
Analysis (CFG)

• IDA Pro generated CFG for “Bomberman”

objc_msgSend
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

Non-trivial to do statically
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
Example ObjC to ASM

1. `LDR R0, =off_24C58`  
   - `UIDevice`
2. `LDR R1, =off_247F4`  
   - `currentDevice`
3. `LDR R0, [R0]`
4. `LDR R1, [R1]`
5. `BLX _objc_msgSend`  
   - `r0? UIDevice r1? ::currentDevice`
6. `LDR R1, =off_247F0`
7. `LDR R1, [R1]`
8. `BLX _objc_msgSend`  
   - `r1? uniqueldentifier`
9. `STR R0, [SP,#0x60+var_34]`
10. `LDR R3, [SP,#0x60+var_34]`
   ... 
11. `BLX _objc_msgSend`  
    - `NSString ::initWithFormat:(fmt: "uniqueid= %@&username=%@&country=%@&email=%@")`
   ... 
12. `BLX _objc_msgSend`  
    - `POSTScore ::startPostingData:toURL: (0x1b478)`
Finding Privacy Leaks

- Inter and intra procedural Control Flow Graph
- Reachability Analysis (find paths)
  - From interesting sources
  - To network sinks
- Implicit interruption of CFG for user-input (e.g., dialog boxes, etc.)
  - Touch events are generated by the OS not in the developer's code
Sources and Sinks

• Sources:
  – Address book
  – GPS coordinates
  – Unique device ID
  – Photos
  – Email account settings
  – WiFi connection information
  – Phone information (phone #, call lists, etc.)
  – YouTube application Settings
  – MobileSafari settings and history
  – Keyboard Cache (every word typed w/o passwords)

Sinks:
  – NSURLConnection, NSString::initWithContentsOfURL, etc.
Data Flow Analysis

- For each source/sink pair perform reachability analysis
  - Is there a path in the CFG that connects the source to the sink?

- Along paths that result from reachability analysis
  - Taint flow analysis
  - Conservatively taint results of methods without implementation if at least one input parameter is tainted
Evaluation

• 1,407 Applications (825 from App Store, 582 from Cydia)
  – Resolving calls to objc_msgSend
  – 4,156,612 calls
  – 3,408,421 identified (82%)
    i.e., class and selector exist and match

• 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
- UDIDs cannot be linked to a person directly
- Problem: Location based Apps
  - Access to GPS is granted per App libraries linked into location based apps have access to GPS too
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
Evaluation Data Flow Analysis

- Reachability analysis: 205 apps
- Enumerate all paths from source to sink with length < 100 basic blocks
- Perform data flow analysis along these paths
- PiOS detected flows of sensitive data for 172 apps (TP)
- 6 true negatives, 27 false negatives
  - FN e.g., aliased pointers, format string from config file, JSON library (i.e., invoking JSONRepresentation on each object in a dictionary, PiOS does not track types in aggregates)
Evaluation: Leaked Data

<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>
Evaluation: Case Studies (1)

• Address book contents:
  – Apps have unrestricted access to the address book
  – Facebook and Gowalla transmit the complete AB
  – Facebook: detailed warning that data will be sent
  – Gowalla (Social networking app):
    • User can send Invitations to contacts
    • Complete AB is sent on load (i.e., before the user chooses a contact)
      → “We couldn’t find any friends from your Address Book who use Gowalla. Why don’t you invite some below?”
Evaluation: Case Studies (2)

- Phone number

- Nov. 2009 Apple removed all Storm8 titles (social games) from App Store
  - because apps transmitted phone numbers (SBFormattedPhoneNumber)
  - New versions don't have that code anymore

- Old version of “Vampires“ PiOS detected the privacy leak
  → Improvement over Apple vetting process
Summary

- PiOS is able to create a CFG from ObjC binaries
- 82% of the calls to objc_msgSend could be resolved
- Data flow analysis is used to identify privacy leaks
- PiOS showed how pervasive ad and statistics libraries are used in apps
- PiOS identified unknown and known privacy leaks that lead to App Store removal in the past
DETECTING PRIVACY LEAKS IN SMARTPHONE APPLICATIONS AT RUN TIME

Byung-Gon Chun
Yahoo! Research

Joint work with William Enck, Peter Gilbert, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, Anmol N. Sheth
Roadmap

• Approach
• TaintDroid design
• Performance study
• Application study
TaintDroid Goal

Monitor app behavior to determine when privacy sensitive information leaves the phone in real time
Current “Best” Practice

- Trust-or-cancel
- Coarse-grained access control
- No visibility into the actual behavior

Hardware controls
- take pictures

Phone calls
- read phone state and identity

System tools
- restart other applications
TaintDroid Approach

• Look inside of applications to watch how they use privacy sensitive data

• Trust-or-cancel → Trust-but-verify
Challenges

• Smartphones are resource constrained
• Third-party applications are entrusted with several types of privacy sensitive information
• Context-based privacy information is dynamic and can be difficult to identify when sent
• Applications can share information
Dynamic Taint Analysis

• A technique that tracks information dependencies from an origin

• Taint
  – Source
  – Propagation
  – Sink

\[
C = \text{Taint\_source}() \\
... \\
A = B + C \\
... \\
\text{Network\_send}(A)
\]
Dynamic Taint Analysis in Action

- Expensive! 2-20x slowdown.
- Whole-system tracking!

code snippet from iexplore.exe

4/23/12

Cellular Networks and Mobile Computing
(COMS 6998-8)

Courtesy: Byung-Gon et. al
TaintDroid
Leverage Android Platform Virtualization

Message-level tracking

Application code

Virtual machine

native system libraries

Network interface

Secondary storage

Variable-level tracking

Method-level tracking

File-level tracking

msg

Application code

Virtual machine
TaintDroid Android Architecture in Detail

Interpreted Code:
- Trusted Application
  - Taint Source
  - (1)
- Untrusted Application
  - (8)

Userspace:
- Dalvik VM Interpreter
  - Virtual Taint Map
  - (2)
  - (3)
- Binder IPC Library
  - Binder Hook
  - (4)
  - (5)
- Trusted Library
  - Taint Sink
  - (6)
  - (7)
  - (8)

Kernel:
- Binder Kernel Module
  - (4)
  - (5)
VM Variable-level Tracking

• Modified the Dalvik VM interpreter to store and propagate taint tags (a taint bitvector) on variables
  – Local variables and method args: taint tags stored adjacent to variables on the internal execution stack.
  – Class fields: similar to locals, but inside static field heap objects
  – Arrays: one taint tag per array to minimize overhead
Modified Stack Format Example

Low Addresses (0x00000000)

- stack pointer (top)
- frame pointer (current)

High Addresses (0xffffffff)

Interpreted Targets

- out0
- VM goop
- v0 == local0
- v0 taint tag
- v1 == in0
- v1 taint tag
- v2 == in1
- v2 taint tag

Native Targets

- arg0
- arg1
- return taint
- arg0 taint tag
- arg1 taint tag

(frame pointer (previous))

(variable)

(variable taint tag)
# DEX Taint Propagation Logic

<table>
<thead>
<tr>
<th>Op Format</th>
<th>Op Semantics</th>
<th>Taint Propagation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>const-op vA C</td>
<td>vA ← C</td>
<td>T(vA) ← 0</td>
<td>Clear vA taint</td>
</tr>
<tr>
<td>move-op vA vB</td>
<td>vA ← vB</td>
<td>T(vA) ← T(vB)</td>
<td>Set vA taint to vB taint</td>
</tr>
<tr>
<td>move-op-R vA</td>
<td>vA ← R</td>
<td>T(vA) ← T(R)</td>
<td>Set vA taint to return taint</td>
</tr>
<tr>
<td>return-op vA</td>
<td>R ← vA</td>
<td>T(R) ← T(vA)</td>
<td>Set return taint (0 if void)</td>
</tr>
<tr>
<td>move-op-E vA</td>
<td>vA ← E</td>
<td>T(vA) ← T(E)</td>
<td>Set vA taint to exception taint</td>
</tr>
<tr>
<td>throw-op vA</td>
<td>E ← vA</td>
<td>T(E) ← T(vA)</td>
<td>Set exception taint</td>
</tr>
<tr>
<td>unary-op vA vB</td>
<td>vA ← op vB</td>
<td>T(vA) ← T(vB)</td>
<td>Set vA taint to vB taint</td>
</tr>
<tr>
<td>binary-op vA vB vC</td>
<td>vA ← vB op vC</td>
<td>T(vA) ← T(vB) U T(vC)</td>
<td>Set vA taint to vA taint U vB taint</td>
</tr>
<tr>
<td>binary-op vA vB</td>
<td>vA ← vA op vB</td>
<td>T(vA) ← T(vA)UT(vB)</td>
<td>Set vA taint to vA taint U vB taint</td>
</tr>
<tr>
<td>binary-op vA vB C</td>
<td>vA ← vB op C</td>
<td>T(vA) ← T(vB)</td>
<td>Set vA taint to vB taint</td>
</tr>
<tr>
<td>aput-op vA vB vC</td>
<td>vB[vC] ← vA</td>
<td>T(vB[]) ← T(vB[]) UT(vA)</td>
<td>Update array vB taint with vA taint</td>
</tr>
</tbody>
</table>

...
Native Methods

• Applications execute native methods through the Java Native Interface (JNI)

• TaintDroid uses a combination of heuristics and method profiles to patch VM tracking state
IPC and File Taint Propagation

• Message-level tracking for IPC
  – Marshall data items
  – Unmarshall data items

• Persistent storage tracked at the file level
  – Single taint tag stored in the file system XATTR
Roadmap

• Approach
• TaintDroid design
• Performance study
• Application study
Performance Study: Microbenchmark

CaffeineMark 3.0 benchmark

- Android
- TaintDroid

14% overhead
Performance Study

• Memory overhead: 4.4%
• IPC overhead: 27%
• Macro-benchmark
  – App load: 3% (2ms)
  – Address book: (<20ms) 5.5% create, 18% read
  – Phone call: 10% (10ms)
  – Take picture: 29% (0.5s)
Taint Adaptors

• Sources
  – Low-bandwidth sensors: location, accelerometer
  – High-bandwidth sensors: microphone, camera
  – Information databases: address book, SMS storage
  – Device identifiers: IMEI, IMSI, ICC-ID, Phone number

• Sink: network
# Application Study

<table>
<thead>
<tr>
<th>Applications (with the Internet permission)</th>
<th>#</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Weather Channel, Cetos, Solitarie, Movies, Babble, Manga Browser</td>
<td>6</td>
<td>![Image1]</td>
</tr>
<tr>
<td>Bump, Wertago, Antivirus, ABC --- Animals, Traffic Jam, Hearts, Blackjack, Horoscope, 3001 Wisdom Quotes Lite, Yellow Pages, Datefonbuch, Astrid, BBC News Live Stream, Ringtones</td>
<td>14</td>
<td>![Image2]</td>
</tr>
<tr>
<td>Layar, Knocking, Coupons, Trapster, Spongebot Slide, ProBasketBall</td>
<td>6</td>
<td>![Image3]</td>
</tr>
<tr>
<td>MySpace, ixMAT, Barcode Scanner</td>
<td>3</td>
<td>![Image4]</td>
</tr>
<tr>
<td>Evernote</td>
<td>1</td>
<td>![Image5]</td>
</tr>
</tbody>
</table>
Findings: Location

• 15 of the 30 apps shared physical location with at least an ad server (admob.com, ad.qwapi.com, ads.mobclix.com, data.flurry.com)
  e.g., received data with tag 0x411 data=
  [GET /servernameA1?
  hello=1&time=1&bumpid=354957030504982&
  locale=en_US&gpslong=-122.316&gpslat=47.662&gpsaccuracy=32.000&timezone=0…

• In no case was sharing obvious to user or in EULA
  – In some cases, periodic and occurred without app use
Findings: Phone Identifiers

• 7 apps sent IMEI and 2 apps sent phone #, IMSI, ICC-ID to remote servers without informing the user

• Frequency was app-specific, e.g., one app sent phone information every time the phone booted
Demo

• http://appanalysis.org/demo/index.html
Conclusion

• Efficient, system-wide, dynamic taint tracking for mobile platforms.
  – 14% overhead for computing-intensive work

• Private data leak is prevalent
  – 20 of the 30 studied applications share information in a way that was not expected
Soundcomber
A Stealthy and Context-Aware Sound Trojan for Smartphones

• Roman Schlegel
  • City University of Hong Kong

• Kehuan Zhang, Xiaoyong Zhou, Mehool Intwala,
• Apu Kapadia, XiaoFeng Wang
  • Indiana University Bloomington
The smartphone in your pocket is really a computer

- 1 GHz Processor
- 512MB / 16GB
- Android OS (Linux)
No surprise malware targets smartphones

- Android malware steals info from 1,000,000 users
- Trojan sends premium-rate text messages
- Security experts release Android root-kit

But “sensory malware” can do much more

What can malware overhear?

Do you think anybody will ever figure out that I keep a spare door key in the flower pot on my front porch?

Nah, how would anybody ever find out?
Some situations are easy to recognize

“Please enter or speak your credit card number now.”
Naive approach: record and upload

1,000,000 phones ≈ 1TB/day

≈ 500KB/day

Internet

Malware Master

Data
t

Valuable Information

4/23/12

Cellular Networks and Mobile Computing (COMS 6998-8)

Courtesy: Roman et. al
Certain combinations of permissions are suspicious

Can easily be recognized and disallowed

([5] W. Enck et al.)
Our contributions over the naive approach

– *targeted* and *local* extraction of valuable data

• inconspicuous permissions
• stealthiness
Naive approach: record and upload
Soundcomber approach: process and upload
Two trojans are stealthier than one
Soundcomber minimizes the necessary permissions
Hotline greetings can be fingerprinted easily

“Thank you for calling...”

“Welcome to ...”

Bank 1

Bank 2
Tricking the user into installing two apps

- pop-up ad
- packaged app
Soundcomber extracts sensitive information locally

Microphone

Soundcomber App

Profile Database

Extract Data

Process Audio

Record Audio

0111010100110101011....

9820305180812615025219029103892...

5180 8126 1502 5219

To Deliverer

Cellular Networks and Mobile Computing
(COMS 6998-8)

4/23/12

Courtesy: Roman et. al
Profiles allow for context aware extraction

- Initial Menu Options
  - Option 1: Prompt for Account Number
    - Enter Account # → Acquire PIN
      - Enter PIN → Sensitive Information Acquired
      - Enter CC #
  - Option 2: Loan & Credit Card
    - Enter Account # → Acquire PIN
      - Enter PIN → Sensitive Information Acquired
  - Other input

Termination

Cellular Networks and Mobile Computing (COMS 6998-8)
DTMF tones are “dual tones”

- 8 frequencies
- 2 simultaneous frequencies for each digit
- used to navigate hotline menus

<table>
<thead>
<tr>
<th></th>
<th>1209 Hz</th>
<th>1336 Hz</th>
<th>1477 Hz</th>
<th>1633 Hz</th>
</tr>
</thead>
<tbody>
<tr>
<td>697 Hz</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>A</td>
</tr>
<tr>
<td>770 Hz</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>B</td>
</tr>
<tr>
<td>852 Hz</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>C</td>
</tr>
<tr>
<td>941 Hz</td>
<td>*</td>
<td>0</td>
<td>#</td>
<td>D</td>
</tr>
</tbody>
</table>
Soundcomber dynamically adjusts thresholds to detect faint tones
Android introduces new covert channels

- vibration settings (87 bps)
- volume settings (150 bps)
- screen (5.3 bps)
- file locks (685 bps)
Vibration settings are broadcast to interested apps

<table>
<thead>
<tr>
<th>Setting</th>
<th>Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>VIBRATE_SETTING_ON</td>
<td>0</td>
</tr>
<tr>
<td>VIBRATE_SETTING_ONLY_SILENT</td>
<td>1</td>
</tr>
</tbody>
</table>

Audio Manager

Set vibration setting

Audio Manager

Vibration setting changed notification

Register broadcast receiver

Deliverer App

Soundcomber App

Setting

VIBRATE_SETTING_ON 0

VIBRATE_SETTING_ONLY_SILENT 1
Volume settings can be modified and accessed by any app

Soundcomber App

Audio Manager

set volume setting

small delay

Audio Manager

get volume setting

Deliverer App

<table>
<thead>
<tr>
<th>Volume</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
</tr>
</tbody>
</table>
Soundcomber is fast and accurate

<table>
<thead>
<tr>
<th></th>
<th>No Error</th>
<th>1 Error</th>
<th>&gt;= 2 Errors</th>
<th>1 missing</th>
<th>&gt;= 2 missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speech</td>
<td>55 %</td>
<td>12.5 %</td>
<td>15 %</td>
<td>7.5 %</td>
<td>10 %</td>
</tr>
<tr>
<td>Tone</td>
<td>85 %</td>
<td>5 %</td>
<td>0</td>
<td>10 %</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Recording Length</th>
<th>Processing Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speech</td>
<td>20 s</td>
<td>7 s</td>
</tr>
<tr>
<td>Tone</td>
<td>45 s</td>
<td>8 s</td>
</tr>
</tbody>
</table>
Hotlines can be fingerprinted with reasonable accuracy

- 20 recorded samples of 5 different hotlines (4 each)
- 20 samples of normal conversation

<table>
<thead>
<tr>
<th>Correct</th>
<th>Missed</th>
<th>Wrong</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hotline</td>
<td>55 %</td>
<td>40 %</td>
</tr>
</tbody>
</table>

Correct

Conversation 100 %

Cellular Networks and Mobile Computing
(COMS 6998-8)

Courtesy: Roman et. al
Keeping Soundcomber hidden and undetectable

• defer/throttle processing
• track user presence
• performance enhancements
Defense: disable recording when a sensitive number is called
Demo

• Demo video
  – http://www.youtube.com/watch?v=Z8ASb-tQVpU
Conclusion

- stealthy, sensory malware is a real threat
- need to explore other such threats
- develop generalized defenses to such attacks
Keypad: Auditing Encrypted File system for Theft-prone Devices

Roxana Geambasu
John P. John
Steve Gribble
Yoshi Kohno
Hank Levy

University of Washington
Slides and Video Presentation

• Slides

• Video presentation