Understanding the Threat Profile of Mobile Apps

Session 11
Wednesday, November 20, 2013
11:15 AM - 12:30 PM
Ming Chow
Tufts University
Key Points

1. Common vulnerabilities and flaws in mobile applications
2. The security features/defenses major platforms provide
3. What attackers of mobile app flaws actually exploit
4. The impact of mobile app attacks
5. App security recommendations/solutions
6. To confirm most of what Charlie Miller said in his keynote yesterday in this room, November 19, 2013
Preliminaries: Why Mobile Is Different

- High value
- Device can be easily lost or broken
- Has good computational power
- It is “always on” (including networking), even when you are not interacting directly with device
- Constraints: battery, screen size, input
- Features: GPS, accelerometer, compass, NFC
- New security model including app distribution
- No authorized method for gaining administrative access by default
- Hence, some old security models (e.g., anti-virus) do not work
What Will Not Be Discussed

- Mobile malware
- Mobile web apps and the mobile browser (that was my talk yesterday)
- Jailbreaking
- Exploit development
- Forensics
It Isn’t Pretty for Mobile Apps

- See Veracode’s **State of Software Security Report: Volume 5**...
- ...or **Chris Wysopal’s presentation** *We See the Future and It’s Not Pretty: Predicting the future using vulnerability data* (SOURCE Boston Conference 2013)
- Compelling numbers
  - One of the key findings: “Cryptographic issues affect a sizeable portion of Android (64%) and iOS (58%) applications.”
Android Vulnerability Prevalence (Percentage of Applications Affected)

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Quality</td>
<td>76%</td>
</tr>
<tr>
<td>Cryptographic Issues</td>
<td>64%</td>
</tr>
<tr>
<td>CRLF Injection</td>
<td>62%</td>
</tr>
<tr>
<td>SQL Injection</td>
<td>31%</td>
</tr>
<tr>
<td>Information Leakage</td>
<td>26%</td>
</tr>
<tr>
<td>Directory Traversal</td>
<td>14%</td>
</tr>
<tr>
<td>Time and State</td>
<td>11%</td>
</tr>
<tr>
<td>Cross-Site Scripting (XSS)</td>
<td>8%</td>
</tr>
<tr>
<td>Authorization Issues</td>
<td>7%</td>
</tr>
<tr>
<td>Credentials Management</td>
<td>7%</td>
</tr>
<tr>
<td>Encapsulation</td>
<td>4%</td>
</tr>
<tr>
<td>Session Fixation</td>
<td>4%</td>
</tr>
<tr>
<td>Dangerous Functions</td>
<td>1%</td>
</tr>
<tr>
<td>API Abuse</td>
<td>1%</td>
</tr>
<tr>
<td>Insufficient Input Validation</td>
<td>1%</td>
</tr>
</tbody>
</table>
iOS (ObjectiveC) Vulnerability Prevalence (Percentage of Applications Affected)

- Error Handling: 76%
- Cryptographic Issues: 58%
- Information Leakage: 42%
- Credentials Management: 17%
- Numeric Errors: 13%
- Buffer Management Errors: 11%
- Code Quality: 8%
- Buffer Overflow: 6%
- Directory Traversal: 4%
- Time and State: 2%
- Format String: 1%
- Untrusted Search Path: 1%
- Race Conditions: 1%
## Vulnerability Distribution for Mobile Platforms (Share of Total Vulnerabilities Found)

<table>
<thead>
<tr>
<th>Android</th>
<th>iOS</th>
<th>Java ME</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRLF Injection</td>
<td>Information Leakage</td>
<td>Cryptographic Issues</td>
</tr>
<tr>
<td>Cryptographic Issues</td>
<td>Error Handling</td>
<td>Information Leakage</td>
</tr>
<tr>
<td>Information Leakage</td>
<td>Cryptographic Issues</td>
<td>Directory Traversal</td>
</tr>
<tr>
<td>SQL Injection</td>
<td>Directory Traversal</td>
<td>Insufficient Input Validation</td>
</tr>
<tr>
<td>Time and State</td>
<td>Buffer Management Errors</td>
<td>Credentials Management</td>
</tr>
</tbody>
</table>
## Android Vulnerability Prevalence

<table>
<thead>
<tr>
<th>Vulnerability Type</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Quality</td>
<td>94%</td>
</tr>
<tr>
<td>Cryptographic Issues</td>
<td>78%</td>
</tr>
<tr>
<td>CRLF Injection</td>
<td>76%</td>
</tr>
<tr>
<td>Information Leakage</td>
<td>39%</td>
</tr>
<tr>
<td>SQL Injection</td>
<td>37%</td>
</tr>
<tr>
<td>Time and State</td>
<td>31%</td>
</tr>
<tr>
<td>Directory Traversal</td>
<td>29%</td>
</tr>
<tr>
<td>Credentials Management</td>
<td>20%</td>
</tr>
<tr>
<td>Insufficient Input Validation</td>
<td>17%</td>
</tr>
<tr>
<td>Authorization Issues</td>
<td>17%</td>
</tr>
<tr>
<td>Potential Backdoor</td>
<td>8%</td>
</tr>
<tr>
<td>Cross-Site Scripting (XSS)</td>
<td>3%</td>
</tr>
<tr>
<td>Dangerous Functions</td>
<td>1%</td>
</tr>
</tbody>
</table>
## iOS Vulnerability Prevalence

<table>
<thead>
<tr>
<th>Issue</th>
<th>Prevalence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Handling</td>
<td>92%</td>
</tr>
<tr>
<td>Cryptographic Issues</td>
<td>85%</td>
</tr>
<tr>
<td>Credentials Management</td>
<td>51%</td>
</tr>
<tr>
<td>Information Leakage</td>
<td>48%</td>
</tr>
<tr>
<td>Code Quality</td>
<td>14%</td>
</tr>
<tr>
<td>Buffer Management Errors</td>
<td>11%</td>
</tr>
<tr>
<td>API Abuse</td>
<td>11%</td>
</tr>
<tr>
<td>Numeric Errors</td>
<td>10%</td>
</tr>
<tr>
<td>Directory Traversal</td>
<td>9%</td>
</tr>
<tr>
<td>Buffer Overflow</td>
<td>3%</td>
</tr>
<tr>
<td>CRLF Injection</td>
<td>1%</td>
</tr>
<tr>
<td>SQL Injection</td>
<td>1%</td>
</tr>
</tbody>
</table>

**UPDATED from Veracode (November 15, 2013; thanks to Chris Wysopal)**
Android Security Model

- The application sandbox (by default)
  - “System assigns a unique user ID (UID) to each Android application and runs it as that user in a separate process.”
  - “Applications cannot interact with each other and applications have limited access to the operating system.”
Android Security Model (continued)

- Least privilege-based permissions to protected APIs, system resources (e.g., camera, GPS, Internet)
  - Apps can only access a limited range of system resources by default
  - Permissions must be stated in `AndroidManifest.xml` file for app
- Permissions check on sensitive data and inputs including calendar and personal information (user grants access)
- No Security Manager (legacy code), no Java Sandbox
- Can make system calls (via NDK)
- Apps can access the kernel, drivers, syscalls, etc. --if device is rooted
- Address Space Layout Randomization (ASLR) supported in Android 4.0
Example of an AndroidManifest.xml file (my app)

```xml
<manifest>
    <uses-sdk
        android:minSdkVersion="11"
        android:targetSdkVersion="17" />

    <uses-feature
        android:glEsVersion="0x00020000"
        android:required="true"/>

    <permission
        android:name="com.example.mapdemo.permission.MAPS_RECEIVE"
        android:protectionLevel="signature"/>

    <uses-permission android:name="com.example.mapdemo.permission.MAPS_RECEIVE"/>
    <uses-permission android:name="android.permissionINTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity
            android:name="edu.tufts.cs.mchow.matransit.MainActivity"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```
iOS Security Model

- Address Space Layer Randomization (ASLR) since iOS 4.3
- Apps run as the same user, but...
  - Apps must be signed by Apple (code signing)
    - App ID, development certificate (public/private keys), production certificate (public/private keys) all required
  - Apps are given a unique ID and directory by Apple
- Sandboxed apps: apps cannot access data from other apps
- To use Apple services including Game Center, iCloud, Passbook, Push Notification Service, development and production certificates (public/private keys) also required
iOS Developer Program Registration

● $99.00 (not including taxes) per year
● “Provide basic personal information, including your legal name and address. If you’re enrolling as a company/organization, we’ll need a few more things, like your legal entity name and D-U-N-S® Number, as part of our verification process.”
● You can revoke and change development and production app certificates
● Devices must be whitelisted for testing
● The moral of the story: “Apple knows who you are”
iTunes App Store App Submission Process

- Must pass manual content review by Apple
- Static analysis used in app review process
- Developer(s) will receive either approved or reject email
- There is App Review Board to appeal rejections
- You can be banned
The status for your app, MATtransit (653572142), is now Ready for Sale.

Dear Ming Chow,

The following app has been approved and the app status has changed to Ready for Sale:

App Name: MATtransit
App Version Number: 1.0
App Type: iOS App
App SKU: MATtransit-iOS
App Apple ID:653572142

If your contracts are not in effect at this time, your app status will be Pending Contract. You may track the progress of your contracts in the Contracts, Tax, and Banking module in iTunes Connect.

To make changes to this app, sign in to iTunes Connect and open the Manage Your Applications module.

It can take up to 24 hours before your app is available on the App Store. This delay is dependent on any app availability issues.

Before you market your app, read the App Store Marketing and Advertising Guidelines for Developers. The guidelines include information on using the App Store badge, best practices to market apps on the App Store, and details on the use of Apple product images.

If you have any questions regarding your app, use the Contact Us module on iTunes Connect.

Regards,

The App Store team
Google Play App Store Submission Process

- Registration: a one time $25 registration fee
- No identity verification necessary.
- Dynamic analysis is used in app review
- No app review board
- No manual content review
- No device whitelisting for testing; just take USB cable, transfer APK to device, and run APK to install app onto phone assuming developer mode is enabled
- Lost the private key for signing Android APK? You’re out of luck, cannot release updates to app.
Malicious App Campaigns

- From **Mobile Exploit Intelligence Project by Guido and Arpaia (SOURCE Boston Conference 2012)**
- iTunes App Store: 0
- Google Play: 30
- The moral of the story: “Say what you will about police states, but they have very little crime.”
- Could be worse: third-party app stores
  - For jailbroken iOS devices
  - For Android (e.g., Amazon, many in Asia)
  - Many more malicious app campaigns
Crypto and Transport Layer Issues

- Hard-coding cryptographic keys directly into mobile app (e.g., to a *.plist file in iOS, to AndroidManifest.xml)
- Not using HTTPS (e.g., for GET and POST API calls)
- Not hashing passwords; stored in a plaintext or transported in the clear
- Recent examples:
  - Tube Map Live Underground (BlackBerry and Android App)
  - Super Backup (Android App)
- Rolling your own crypto, snake oil
More Information Leakage From Apps

- Your entire address book was sent to Path (iOS App)
  - http://mclov.in/2012/02/08/path-uploads-your-entire-address-book-to-their-servers.html
  - Path was fined $800K by the FCC for collecting information without users’ (especially children’s) consent
  - Now an iOS app must ask for permission to use your contacts
- Path was not the only app that took your address book
Veracode’s AdiOS

http://www.veracode.com/blog/2012/02/adios-say-goodbye-to-nosy-iphone-apps/
What Are Pentesters Doing and Finding

- Wi-Fi attacks
- Phishing (email and SMS) and social engineering
- Running remote attacks
- Running mitmproxy
- Using Android (default or jailbroken) devices
- Decompiling Android APKs
- Fuzz testing
- Findings: information leakage, privilege escalation
Advanced Research

- Executing special Unstructured Supplementary Service Data (USSD) codes to remote wipe Android devices (Borgaonkar, 2012)
- Twitter SMS Spoofing via phone number, assuming no lock (Rudenberg, 2012)
- Android NFC vulnerabilities (Miller, 2012; Wall of Sheep 2013)
- Facebook, LinkedIn, and Dropbox app authentication keys stored in plain text on iOS (2012)
- Juice-jacking (Wall of Sheep 2011; Lau, Jang, and Song, 2013)
- Android master key vulnerability (Forristal, 2013)
- Rooting SIM cards (Nojl, 2013)
- Google Chrome and iOS Safari vulnerabilities (Mobile Pwn2Own 2013)
Impact, The Loss (as of now)

- Credentials (Facebook, banking)
- Photos
- Address books, contacts
- Your location
- Installation of malware to steal SMS and call logs, etc.
Recommendations and Tools

- **Tools**
  - mitmproxy (http://mitmproxy.org/)
  - Crashlytics (for developing Android and iOS apps; https://crashlytics.com/)
  - Binary static analysis tools, dynamic analysis (e.g., Veracode)

- **Readings:**
Crashlytics

- **Issues**
  - 9 Issues
  - 366 Non-Fatal Issues
  - 40 Users Affected
  - 15 Crashes
  - 11 Users Affected

- **Issues Details**
  - **MyLocation.java** line 42
    - edu.tufts.cs.mchow.matransit.MyLocation.onLocation...  
    - 2.0 (2) Non-Fatal
    - 352 Non-Fatal
    - 39 Users

  - **HttpConnection.java** line 70
    - libcore.net.http.HttpConnection.<init>
    - 2.0 (2) Non-Fatal
    - 8 Non-Fatal
    - 1 Users

  - **MainActivity.java** line 51
    - edu.tufts.cs.mchow.matransit.MainActivity.initMain...  
    - 2.0 (2) Non-Fatal
    - 8 Crashes
    - 4 Users

  - **GetClosestStations.java** line 66
    - edu.tufts.cs.mchow.matransit.GetClosestStations....  
    - 2.0 (2) Crashes
    - 6 Users

  - **GetClosestStations.java** line 38
    - edu.tufts.cs.mchow.matransit.GetClosestStations....  
    - 2.0 (2) Non-Fatal
    - 2 Users
The Third-Party Server

- Even if your app passes static and dynamic analysis, and have no crashes, there may be serious problems on the server end
- Servers will always be targeted
- Example: my app messagehub for Android and iOS (not on app stores) and the corresponding messagehub web application has GET and POST JSON APIs
- January 2013: ESPN ScoreCenter app (iOS) cross-site scripting (XSS) and cleartext password vulnerabilities
The Future

- Mobile is here to stay
- The good news: vendors (e.g., Apple, Google, Samsung) respond to concerns
- The bad news:
  - Fragmentation of devices; some devices may not be able to update apps or the OS (think carriers)
  - No end in sight to the mobile malware problem on Android; 718,000 malicious apps and counting
  - Online banking moving to mobile
  - Sophistication necessary to find mobile exploits. Look at what pentesters are finding vs what is actually getting exploited
- Problems and cost of impact bound to be greater later
References

- http://bluebox.com/corporate-blog/bluebox-uncovers-android-master-key/