Secure Android Applications
The OWASP Way

Jack Mannino
CEO/Chief “Breaker”

June 21, 2011
Overview

- Who I am/ What we do
- OWASP Mobile Security Project
- Mobile World Meets Security World
- Android Crash Course
- Threat Modeling Android Apps
- Risks and Controls
- Where Do We Go From here?
- Q&A, Resources
Who I Am/ What We Do/ Where We Are

➢ Who I am
  • Jack Mannino
  • Company co-founder
  • Co-leader of the OWASP Mobile Security Project
  • Has a lot of phones.....

➢ What we do:
  • Mobile Application Security
  • Web Application Security
  • Penetration Testing
  • Secure Development Training

➢ Where we are:
  • Northern Virginia
Began in 2010

Current state of mobile application security: bad

We are aiming to make it: good

How do we plan to achieve this?
OWASP Mobile Security Project

- Threat Modeling
- Top 10 Risks
- Top 10 Controls
- Cheat Sheets
- Testing Methodologies
- Tools
- Secure Development Guide
- Training Resources
Disclaimer

- We support OWASP by contributing expertise to the security community
- OWASP does not support or endorse our business and services
- Why am I mentioning this?

https://www.owasp.org/index.php/OWASP_brand_usage_rules
Mobile World Meets Security World
Mobile World Meets Security World

- Once upon a time, all phones could do was make phone calls....

- And then, the world changed

- Today’s mobile devices do things like
  - Make phone calls
  - Send SMS messages
  - Browse the web
  - VPN into corporate assets
  - Video conferencing
  - Track our location
  - Tap our phones to pay for things (soon)

- Is anyone making money?

- Do people use these things and their “apps”?


Top Five Smartphone Vendors, Shipments, and Market Share, Q4 2010 (Units in Millions)

<table>
<thead>
<tr>
<th>Vendor</th>
<th>4Q10 Units Shipped</th>
<th>4Q10 Market Share</th>
<th>4Q09 Units Shipped</th>
<th>4Q09 Market Share</th>
<th>Year-over-year growth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nokia</td>
<td>26.3</td>
<td>28.0%</td>
<td>26.8</td>
<td>36.6%</td>
<td>36.1%</td>
</tr>
<tr>
<td>Apple</td>
<td>16.2</td>
<td>16.1%</td>
<td>8.7</td>
<td>16.1%</td>
<td>86.2%</td>
</tr>
<tr>
<td>Research In Motion</td>
<td>14.6</td>
<td>14.5%</td>
<td>16.7</td>
<td>19.9%</td>
<td>36.4%</td>
</tr>
<tr>
<td>Samsung</td>
<td>9.7</td>
<td>9.6%</td>
<td>1.8</td>
<td>3.3%</td>
<td>438.9%</td>
</tr>
<tr>
<td>HTC</td>
<td>8.6</td>
<td>8.5%</td>
<td>2.4</td>
<td>4.5%</td>
<td>258.3%</td>
</tr>
<tr>
<td>Others</td>
<td>23.5</td>
<td>23.3%</td>
<td>9.5</td>
<td>17.6%</td>
<td>147.4%</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>100.9</strong></td>
<td><strong>100.0%</strong></td>
<td><strong>53.9</strong></td>
<td><strong>100.0%</strong></td>
<td><strong>87.2%</strong></td>
</tr>
</tbody>
</table>

Source: IDC Worldwide Quarterly PC Tracker, January 12, 2011
Android Crash Course
And Now…Android!

- Debuted in 2008
- Most popular mobile platform around

**Smartphone market share**

March '11, Nielsen Mobile Insights, National

- Android OS 37%
- Apple iOS 27%
- RIM BlackBerry OS 22%
- Microsoft Windows Mobile / WP 10%
- Palm / WebOS 3%
- Symbian OS 2%

Source: The Nielsen Company.
People Use Android….Now What?

- Huge market share + attack monetization = target
- Android Market is OPEN (in a bad way)
- In the past 2 months, 4 times as much Android malware as all of 2010
  (Source: Friend @ Lookout Mobile Security)
  - GGTracker- Toll fraud
  - DroidDream- Trojan in over 50 Android apps
  - Plankton- Steals browsing history, credentials, device logs, and more
    - 12 apps undetected in the Android Market for over 2 months!
    - Masqueraded with titles like “Angry Birds Rio Unlock”
It Gets Worse

- Mobile developers are partying like it’s 1999
- Android platform is highly fragmented
- Apps are self-signed
- Old vulnerabilities are new vulnerabilities
- New developers, new companies
- Have we learned anything?!
Android Crash Course- Overview

- Linux-based operating system
- Optimized for ARM architecture
- Android runtime and libraries run on top of the OS
- Applications run within the Dalvik Virtual Machine
- Dalvik = optimization, not security
- Each application runs in its own process (with exceptions)
- Permissions model dictates what apps can/can’t do (sometimes)
Android Crash Course - Architecture
Android Crash Course- Essentials

- **AndroidManifest.xml**
  - Main configuration file
  - Where most components are declared
    - Permissions
    - Activities
    - Intents
    - Content Providers

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.nvisium.tapjacking" android:versionCode="2"
  android:versionName="2.0">
  <uses-sdk android:minSdkVersion="7" />
  <application android:icon="@drawable/h" android:label="@string/app_name">
    <activity android:name="Main" android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <service android:name=".DialerService">
      <intent-filter>
        <action android:name="com.nvisium.tapjacking.DialerService" />
      </intent-filter>
    </service>
    <service android:name=".BackgroundInstallerService">
      <intent-filter>
        <action android:name="com.nvisium.tapjacking.BackgroundInstallerService" />
      </intent-filter>
    </service>
  </application>
</manifest>
```
Permissions

- Applications are granted permissions for various actions
- Declared within AndroidManifest.xml
- “All or nothing” basis
  - ACCESS_FINE_LOCATION
  - CALL_PHONE
  - WRITE_SETTINGS
  - WRITE_SMS
  - READ_LOGS
  - And many, many more
  - Custom permissions too
Permissions

Some developers go overboard

Questionable apps often request ridiculous permissions too

Example: Justin Bieber Wallpaper

- android.permission.PROCESS_OUTGOING_CALLS
- android.permission.WAKE_LOCK,
- android.permission.READ_PHONE_STATE
- android.permission INTERNET
- android.permission.RECEIVE_BOOT_COMPLETED
- android.permission.ACCESS_NETWORK_STATE
- android.permission.ACCESS_COARSE_LOCATION
- android.permission.ACCESS_FINE_LOCATION
- com.google.android.googleapps.permission.GOOGLE_AUTH
- com.google.android.googleapps.permission.GOOGLE_AUTH.OTHER_SERVICES
- android.permission.GET_ACCOUNTS
Activity

Single, focused thing a user can do (simple definition)

Intent

- Used to launch Activities and communicate with other components
- Primary way of passing around data within Android

```java
Intent intent = new Intent(Intent.ACTION_DIAL);
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
// showing Google some love
intent.setData(Uri.parse("tel:650-253-0000"));
getApplication().startActivity(intent);
```
Content Provider

- Used to expose and access data across applications
- Permissions are declared by provider attribute in AndroidManifest.xml
- Exposes data using a URI format

content://com.somepackage.topsecret/piidata/3

Prefix
Authority (class name)
Path
Record ID
Threat Modeling Android Apps
Threat Modeling Android Apps

- Threat modeling is used to better understand an application’s surface for attack
- Don’t assume the sky is falling…..
- Assume that it already fell
- Users:
  - Root their phones
  - Lose their phones
  - Install things they shouldn’t
  - Use public wifi
  - Never listen to security people (ever)...
- Now we can see the bigger picture
Threat Modeling Android Apps

- Corporate Assets
- Android Market Remote Install/Remove/Wipe/Backup
- Website/Web Service
- Good and Bad Apps
- Facebook Phishing, Malware, Mark Zuckerberg, etc.

"The Web"

- Wireless Interfaces (NFC, Bluetooth, Wifi)
- Theft or Loss

OWASP
The Open Web Application Security Project

nVisium SECURITY
Threat Modeling Android Apps - Remote Market Attack

Corporate Assets

Android Market
Remote Install/Remove/Wipe/Backup

Website/Web Service

Good and Bad Apps

Facebook

Phishing, Malware, Mark Zuckerberg, etc.

Wireless Interfaces (NFC, Bluetooth, Wifi)

The Web

Theft or Loss
Threat Modeling Android Apps - Legacy Architectures
Risks And Controls
OWASP Mobile Top 10 Risks and Controls

Top 10 Risks

1. Insecure or unnecessary client-side data storage
2. Lack of data protection in transit
3. Personal data leakage
4. Failure to protect resources with strong authentication
5. Failure to implement least privilege authorization policy
6. Client-side injection
7. Client-side Denial Of Service (DoS)
8. Malicious third-party code
9. Client-side buffer overflow
10. Failure to apply server-side controls
#1 Insecure or Unnecessary Client-Side Data Storage

- Do I really have to store it?

```bash
# cd shared_prefs
# ls
com.evernote_preferences.xml
# cat
com.evernote_preferences.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
  <string name="serviceHost">happy_gilmore</string>
  <string name="username">MyPassword123</string>
  <boolean name="ACCOUNT_CHECKED" value="true" />
  <int name="servicePort" value="0" />
  <boolean name="NotifyUploadStatus" value="true" />
</map>
#
#2 Lack of Data Protection in Transit

- No SSL/TLS

- Broken SSL/TLS
  - Ignoring certificate errors to “make apps work”
  - Facilitates Man In The Middle (MITM) attacks

- Near Field Communications (NFC) leaves transport encryption up to the developers to implement correctly

**Controls:**

- Use strong transport encryption when transmitting sensitive information
  - Even over 3G/4G...assume the carrier is compromised too
- Detect errors and properly handle them
  - Unrecognized CA
  - Certificate name mismatches
#3 Personal Data Leakage

- Logging sensitive information
- Caching sensitive information
  - Browser
  - Search history
  - Location information

Controls:
- Use only protected storage areas
  - Never external media!
  - Don’t use the global log file
- Understand the implications of what you are storing and caching
  - Do you really need 3 years of GPS info on the device?
#4 Failure To Protect Resources With Strong Authentication

- This risk presents itself in multiple ways:
  - App-to-app
    - Single Sign On (Google Auth, Facebook)
    - Exposing Content Providers, Broadcasts
  - Client/Server
    - Has overlap with #10- Failure To Apply Server Side Controls

- Controls:
  - Keep small session timeout windows when possible
  - Require re-authentication for sensitive actions
  - Never authenticate based on:
    - Device ID
    - Location
#5 Failure To Implement Least Privilege Authorization Policy

- Overly permissive permissions granted to apps
  - Does an application really need to modify system settings?
  - Does the permission even get used?

- File access
  - MODE_WORLD_READABLE, MODE_WORLD_WRITEABLE

- Overexposing Android components
  - Activities
  - Intents
  - Content Providers

- Controls:
  - Only grant what is needed
  - K.I.S.S.
  - Common sense usually prevails
#6 Client-Side Injection

- Lots of familiar faces
  - Cross Site Scripting (XSS)
  - Client-side SQL Injection

- Multiple entry points
  - Browser
  - App-to-app
  - Server-side initiated attacks

- Controls:
  - Encode data as close to parser boundary as possible
  - Validate input, validate output
  - Database calls should use prepared statements
    - String concatenation = still bad
#7 Client-Side DoS

- Scenarios that cause an application to stop working
  - Application crashes
  - Denies system resources to other apps
    - Dialing 911

- May be triggered
  - Server side
  - Client-side

- Controls:
  - Handle exceptions gracefully
  - Perform load testing to ensure resources are released as intended
#8 Malicious Third-Party Code

- Lots of free to use code

- Do your due diligence before using it

- Trustworthy sources only

- Perform code review before using third party libraries
#9 Client-Side Buffer Overflow

- On Android, applies to native apps
  - If your application uses native libraries, this applies
  - If you are using the standard SDK, less to worry about

- Controls:
  - As insurance, always validate input and output
  - Perform bounds checking on native code you develop
#10 Failure To Apply Server-Side Controls

- This should be familiar territory
  - Anything originating from the client = untrusted

- Parameter manipulation
  - Prices
  - User ID (potential privilege escalation)

- Injection Attacks
  - SQL Injection (against server)
  - Attacking web services

- Controls:
  - Many...
  - OWASP Top 10 for web covers these issues
Where Do We Go From Here?
What Happens Next?

- We haven’t seen ANYTHING yet
- Ton of education and awareness needed
- Things will get worse before they get better
- Technology is outpacing security
- Can’t fix the hard stuff without fixing easy stuff
Questions?

➢ Got them? Ask them

➢ I hope this was useful

➢ Thank you for attending!

➢ Contact Information:
  • Jack@nvisiumsecurity.com
  • http://twitter.com/jack_mannino
  • http://www.linkedin.com/pub/jack-mannino/7/2b7/562
Resources

- **OWASP Mobile Security Project**

- **Android Developer Resources**

- **DroidDream**

- **Plankton**
  - [http://www.csc.ncsu.edu/faculty/jiang/Plankton/](http://www.csc.ncsu.edu/faculty/jiang/Plankton/)

- **OWASP iGoat Project**
  - [https://www.owasp.org/index.php/OWASP_iGoat_Project](https://www.owasp.org/index.php/OWASP_iGoat_Project)