Happy May the 4th!!!!1!

“USE THE FORCE, HARRY”
—Gandalf
Android Security

CS 642
Drew Davidson
Some Slides taken from John Mitchell
Lecture Roadmap

• What is Android?
  – History
  – Design

• Exploits
  – System Defenses

• Other Attacks
  – Threats
  – Defenses
What is Android?

• A lot of things to different people
  – The fabled gPhone
    • Invites comparison to the iPhone
  – An internet of things (IoT) platform
  – An operating system for your car?
• Too big to explain in this lecture
  – We’ll introduce some security features as needed
  – More to learn
(Ancient) History of Android

• 2003: Andy Rubin cofounds Android Inc to build a web-connected smartphone
• 2005: Google acquires Android Inc
• 2007: iPhone Gen I released
• 2008: HTC Dream (G1) released
Android Design

• More than an Operating System
  – A specialized Linux distro, at the lowest level
  – A framework for running Android “apps”
  – An entire ecosystem for smartphone users

Android Open Source Project

Apps
App Store (Google Play)
Development tools
Closed-Source Components
From Google to You

Google → OEM

Users → Service Provider

Google

OEM

verizon

Users
Android Exploits
What is an Android Exploit?

• Working definition:
  An action that occurs in contravention of the security model of an Architecture

• Examples:
  – Privilege Escalation: User code runs as root
  – Data Exfiltration: App steals another’s data
  – DOS: App renders device unusable
Multi-Layered Architecture
Application Design

• Each app runs within an independent instance of the Dalvik Virtual Machine (DVM)
  – Apps largely run bytecode
  – Each app runs as its own user, i.e. there is a separate UID for each app
App Deployment

Diagram showing the process of app deployment, including steps like application resources, Java compiler, .class files, dex, .dex files, and Android package (.apk).
Intra-Application Security

• Signed code
  – Prevents out-of-band rewrites

• Java-style Sandbox protections
  – Bytecode verifier prevents ill-formed programs
  – Runtime checks against buffer overflows, etc.
  – Could use the security manager for policies

• Android Lifecycle, App Killer
  – System may pause an app
  – System may kill an app with too many resources
Inter-Application Security

• OS level protections
  – Separate UIDs give apps distinct privileges
  – Minimizes privilege escalation

• Binder IPC
  – Kernel mediates communication between apps
  – Receiving app must register for incoming messages
OS Protection

• ASLR
  – Makes it statistically impossible/improbable to know if you’re smashing the stack effectively

• Dlmalloc
  – Makes it much harder to spray the heap
Google Play (Store)

• Largest distribution channel for apps
  – Kill switch
  – Google Bouncer
  – “Wisdom” of the crowds
Exploits Still Happen

• Confused deputy
  – Stagefright

• Data exfiltration
  – Sensor side-channels
    • Microphone, Gyroscope
  – App misconfiguration
    • Facebook Debug log

• Denial of Service
  – Exception loops
  – Battery drain
Other Threats
Shady Code

• The previous definition of exploit was somewhat weak
  – What happens when the security model is insufficient?

• Enable “PII attacks”
  – Broadly, attacks that leverage your personally identifiable information
Shady Code Defenses

- Android Permissions
  - Install-time permissions
Shady Code Defenses

• Android Permissions
  – Runtime
  – Update-Time
<table>
<thead>
<tr>
<th>Category</th>
<th>Permission</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Your Accounts</td>
<td>AUTHENTICATE_ACCOUNTS</td>
<td>Act as an account authenticator</td>
</tr>
<tr>
<td></td>
<td>MANAGE_ACCOUNTS</td>
<td>Manage accounts list</td>
</tr>
<tr>
<td></td>
<td>USE_CREDENTIALS</td>
<td>Use authentication credentials</td>
</tr>
<tr>
<td>Network Communication</td>
<td>INTERNET</td>
<td>Full Internet access</td>
</tr>
<tr>
<td></td>
<td>ACCESS_NETWORK_STATE</td>
<td>View network state</td>
</tr>
<tr>
<td>Your Personal Information</td>
<td>READ_CONTACTS</td>
<td>Read contact data</td>
</tr>
<tr>
<td></td>
<td>WRITE_CONTACTS</td>
<td>Write contact data</td>
</tr>
<tr>
<td>System Tools</td>
<td>WRITE_SETTINGS</td>
<td>Modify global system settings</td>
</tr>
<tr>
<td></td>
<td>WRITE_SYNC_SETTINGS</td>
<td>Write sync settings (e.g. Contact sync)</td>
</tr>
<tr>
<td></td>
<td>READ_SYNC_SETTINGS</td>
<td>Read whether sync is enabled</td>
</tr>
<tr>
<td></td>
<td>READ_SYNC_STATS</td>
<td>Read history of syncs</td>
</tr>
<tr>
<td>Your Accounts</td>
<td>GET_ACCOUNTS</td>
<td>Discover known accounts</td>
</tr>
<tr>
<td>Extra/Custom</td>
<td>WRITE_SECURE_SETTINGS</td>
<td>Modify secure system settings</td>
</tr>
</tbody>
</table>
What’s the Problem with Permissions?

• Admittedly, a step up over the Desktop
  – Arguably, table stakes for such a personal device

• “Permission entanglement”
  – You may control when a permission is used, but not how
    • Permissions are per-app thus shared with libraries
    • A single permission may be used in various ways
    • Composite effect of permissions exceed sum
Fixing Shady Code

• Fewer easy answers
  – One person’s privacy violation is another’s feature
    • Location-aware advertising?
Now Entering the Realm of Research

- What follows is a discussion of research prototypes
  - Unlike above, there are occasionally obvious reasons NOT to do these things
Data flow analysis

• Label the uses of permissions in the program
  – Sources: produce sensitive information
  – Sinks: interact with untrusted entities

• We’d like to know how these endpoints interact

• Tools
  – FlowDroid
  – Stamp
Example Endpoint permissions

Sources
• Account data
• Audio
• Calendar
• Call log
• Camera
• Contacts
• Device Id
• Location
• Photos (Geotags)
• SD card data
• SMS

Sinks
• Internet (socket)
• SMS
• Email
• System Logs
• Webview/Browser
• File System
• Broadcast Message
Possible Flows

Sources

- READ_CONTACTS
- READ_SYNC_SETTINGS
- READ_SYNC_STATS
- GET_ACCOUNTS
- INTERNET

Sinks

- INTERNET
- WRITE_SETTINGS
- WRITE_CONTACTS
- WRITE_SECURE_SETTINGS
- WRITE_SETTINGS
Implementing Dataflow Analysis

• Identify what methods use which permissions
  – No canonical map!

• Identify what permissions actually do
  – Is it a source? Sink? BOTH?

• View the program as a Program Dependence Graph
  – Edges represent flows of control or data
  – Nodes represent abstract regions of code
  – Requires a program semantics / abstraction
Dataflow Analysis Example

FB API

Source: FB_Data

Write Contacts

Sink: Contact_Book

Read Contacts

Source: Contacts

Send Internet

Sink: Internet
Limitations of Dataflow Analysis

• Technical
  – Over-approximate
  – Requires deep knowledge of the system
    • Impractical without some manual modelling, at least on Android

• Practical
  ...ideas?
(Dynamic) Taint Tracking

• Not the most media-savvy name
• Extend the system to record the provenance of data
  – Is it *tainted* by an input source?
• Tools
  – TaintDroid
Limitations of Dynamic Taint Tracking

- Technical limitations
  - Misses control dependencies
- Practical limitations
  - Slows execution
  - Could use it solely as an offline analysis
App Rewriting

• Change the behavior of the app
  – Reverse engineer it
  – Make some changes
  – Recompile it
DroidWeave

• To the board!
Conclusion

• Good luck on Finals!

• If you’re graduating, good luck in life!