# An Overview of Android Security

Department of Computer Science
University of Illinois at Chicago

Rigel Gjomemo
rgjome1@uic.edu

## Agenda

- Android adoption and reasons for its success
- Overview of existing security mechanisms in Android and of related research
  - Android permissions
  - Android application components
- The problem of Input Validation
- Parameter Tampering
- Rooting
- App Provenance
- Sensitive data leakage
- Android Malware Survey

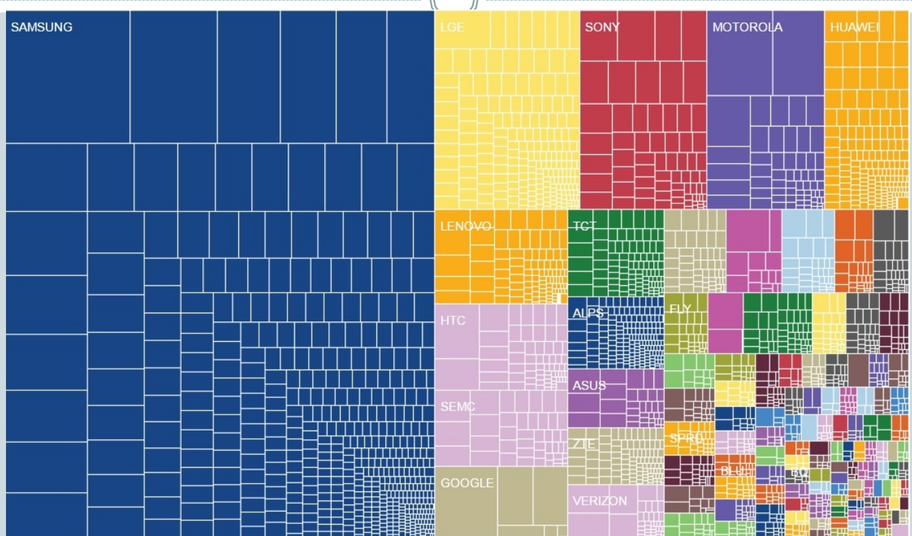
## Android Adoption in Numbers (2019)

- 2.0 billion users worldwide
- 75% market share
- 2.1 million apps on Google Play market alone
- 24K+ different customizations (2016)

## Reasons for Success

- Android Open Source Project
  - Distributed under Apache License 2.0
  - Carriers , device manufacturers, end users can customize it easily
  - Encourages testing, innovation, distribution
- Open Handset Alliance
  - 84+ mobile and technology companies invested heavily in Android
- Availability to mobile carriers (US only)
  - iPhone: AT&T (2007-2011), Verizon, Sprint (2011), T-Mobile (recently)
  - Android: T-Mobile (2008), Sprint, Verizon (2009), AT&T (2010)

## Fragmentation by Manufacturer (2014)

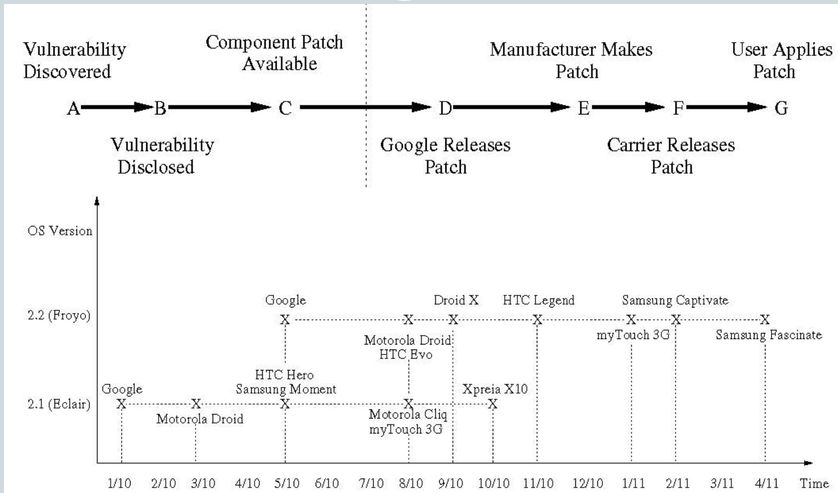



Source: http://opensignal.com/reports/2014/android-fragmentation/

## Security Implications

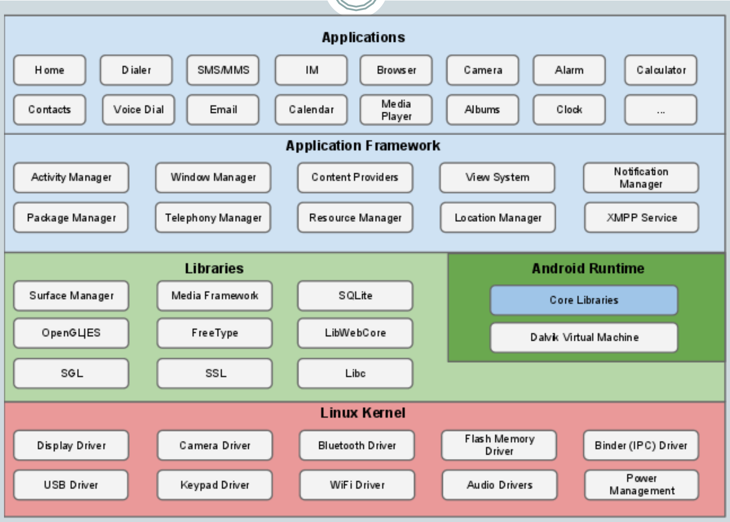
- Widely popular, therefore preferred target
  - Malware can reach more users
  - More private data can be collected
- Open source, therefore easier to study
  - For both attackers and defenders
- Fragmentation makes patches very hard to reach end users
  - Things have been changing since Lollipop (Android 5.0)
    - More streamlined update processes
    - Shaming manufacturers (?)

# Fragmentation and Patch Cycles




Source: [Vidas]

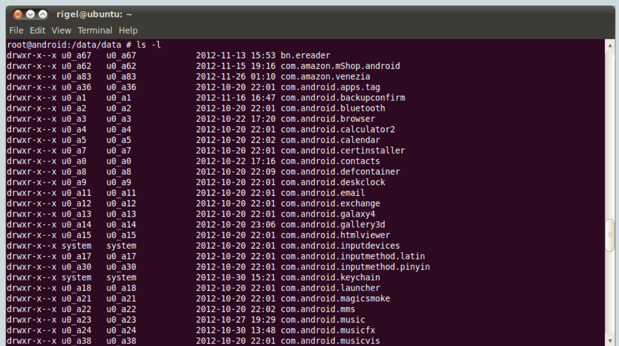
# Android Software Stack




Source: http://source.android.com/tech/security/index.html

## Android Security Features: Sandboxing

- Every application has a distinct normal user ID
- Directories used by applications are by default readable and writable only by the application themselves
- At runtime, every application has its own memory space




## Android Security Features: Filesystem Encryption

- Full application data encryption (since Android 3.0, on by default since Android 5)
- Not available with common file system of previous Android devices
- Encryption key is encrypted by user’s unlock password
  - Is it long enough?
- Encryption is irreversible

## Android Security Features: Memory Management et al.

- New mechanisms against memory attacks introduced with every Android version
- ProPolice against stack overflows (Android 1.5)
- Format string protections, hardware-based No eXecute (Android 2.3)
- Address Space Layout Randomization (Android 4.0)

Source: https://source.android.com/devices/tech/security/enhancements/index.html

## Android Security Features: Memory Management et al.

- Adb authentication (Android 4.2.2)
- No setuid/setgid programs.
- Smart Lock, unlock when paired with devices, face recognition (Android 5)
- SELinux on by default (Android 5)
- Boot Verification (Android 6)
- Update patches for several components (e.g., WebView) connected to Google Apps rather than the rest of the system
- ……

Source: https://source.android.com/devices/tech/security/enhancements/index.html

## Android Security Features: Permissions

- Advantage: can provide fine grained control to the functionality an application can use
  - Applications used to request the permissions they need only once at installation time
  - Since Android 6 (Marshmallow) most requests are done at run time.
  - Application developers can define their own permissions to control access to application services
- Disadvantages
  - Too fine grained?
  - End users are the final enforcers (blame is on them)
    - Users typically unaware of importance of permissions
    - Since Marshmallow permissions can be revoked w/o uninstalling app

## Permission protectionLevel

- Normal: low risk permission, granted by default at installation (e.g., RECEIVE_BOOT_COMPLETED)
- Dangerous: require user confirmation (e.g., CAMERA)
- Signature: granted if the two apps (sender and receiver) have been signed with the same certificate (user defined).
- SignatureOrSystem: granted to applications that are in the Android image (e.g., REBOOT).

## Android Permissions Research (I)

- Are applications over-privileged? [AAFER-CCS18]
  - Built permission map: Android API call <-> Permission
  - Code analysis to find Android API calls in applications
- Results:
  - Most examined applications, including system apps, are overprivileged
  - Over 8 unnecessary permissions averaged across 12 OEMs

## Android Permissions Research (II)

- How do end users deal with permission information at install time? [FELT2]
- Internet survey (308 users, lab interviews, 25 users)
  - Attention. 17% of Internet participants pay attention, 42% of laboratory participants are unaware of permissions
  - Comprehension. Only 3% of Internet and 24% of laboratory participants could answer questions about permissions correctly.
  - Behavior. Majority canceled installation at least once because of permissions

## Tips about Using Permissions

- Try to request as few permissions as possible
- For exposed functionality (e.g., Content Providers) between your apps use signature permissions (not confirmed by users)
- If sending sensitive data, require receiving applications to have signature permissions (or use explicit intents)

## Android Application Components

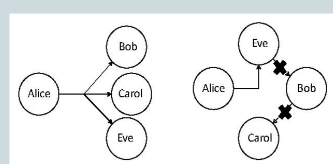
- Activities
  - Every screen that interacts with a user is an Activity
- Services
  - Background processes with no user interface
- Content Providers
  - Used to provide data access to other applications in a controlled way
  - Application that reads the data must request the permission defined by the application that provides the data
- Broadcast Receivers
  - Listen to broadcast messages (e.g., boot completed, time changed, battery low)

## Application Components (II)

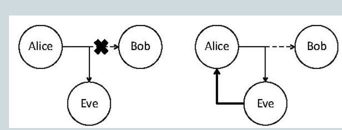
- Intents
  - Messages exchanged by application components
    - Action (e.g., ACTION_CALL)
    - Data (e.g., phone number)
- Intent types
  - Explicit: the application component to send the message to is explicitly named
  - Implicit: the system determines the component that will receive the message
- Every Activity, Service, BroadcastReceiver declares what types of Intents it can accept

## Android Inter-Application Communication Research (I)

- Implicit intents and broadcasts may be intercepted by malicious apps ([Chin], [Kantola])
- Broadcast theft
  - Eavesdropping
  - Denial of service (ordered broadcasts)
  - Data injection (ordered broadcasts)




- Activity hijacking
  - Malicious Activity registers to receive other Activities Intents




- Intent Spoofing
  - Malicious Activity sends spoofed Intent to Activity or BroadcastReceiver

Source of figures: [Chin]

## Android Inter-Application Communication Research (II)

- ComDroid [Chin]
- Analyzes disassembled application code
  - Intents and implicit intents usage (for Broadcast theft and Activity Hijacking)
  - BroadcastReceivers and Activities (for intent spoofing)
- Results
  - 100 applications
  - 401 warnings about exposed BroadcastReceivers and Activities
  - 1013 warnings about exposed Intents

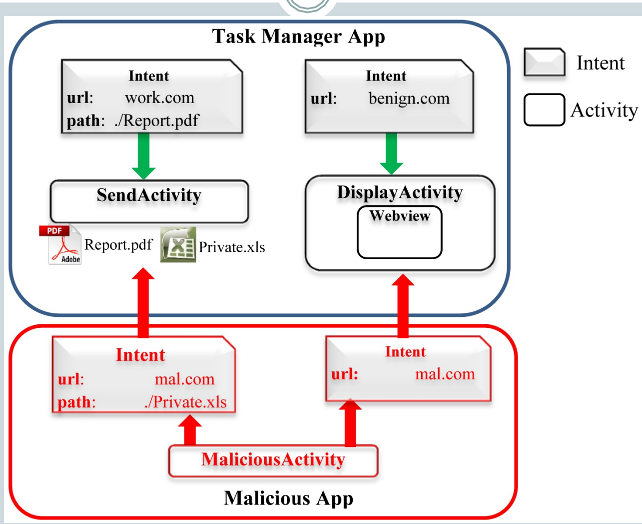
## Tips for Application Components

1. Set android:exported to false in the manifest file
   - Default since API 17
2. Always remove intent-filters and use explicit intents for intra-app communications
   - If there is an intent-filter the Activity is exported for matching intents
3. Restrict access to a component with signature type permissions
4. Specify required permissions that receiver must have for broadcasts

# INPUT VALIDATION

- Apps receive and send data from and to many sources (other apps, file system, network)
- Input's origin can be often subverted
- On Android, often there is no guarantee about an Intent's origin
- App developers should be aware that some input may not be trusted

# Confused Deputy Attack



## Confused Deputy Attack

- App A has internet permission and exposed components
- Attacker studies App A and determines paths through the application that lead from sources (e.g., intent data extraction) to sensitive sinks (e.g., network operations)
- Attacker sends intent with malicious data hoping to influence the execution at the sinks
- App A does not validate the input and performs a network operation on behalf of the attacker

## Research Studies

- Are there paths from sources to sinks? [Lu]
  - Methodology: Taint propagation through program slices
  - Creation of all possible slice permutations
  - Results: 206/5486 apps contain paths from sources to sinks
- Are developers performing sufficient input validation? [Gallingani]
  - Methodology: Taint propagation and symbolic execution
  - Results: 26/64 apps contain exploitable paths

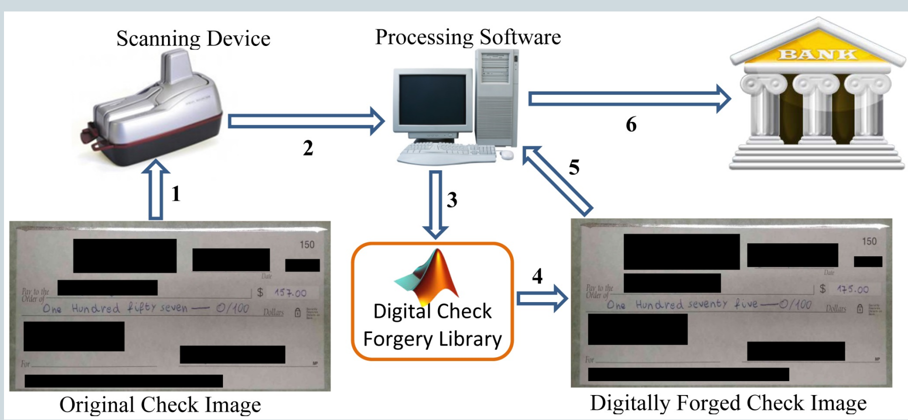
## Parameter Tampering

- Server trusts the data sent by client

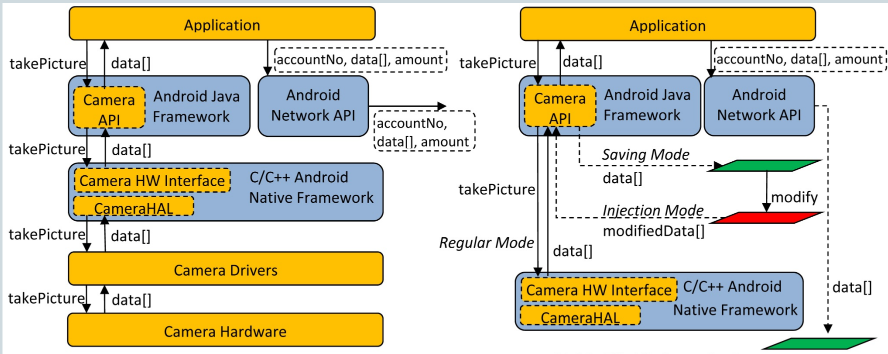
BUT

- Client can be subverted
- Environment where client is running can be subverted
- As a result, server can receive malicious data
- On Android even more so, because of (wrong) assumptions about privileges.

## Example



# On Android



(a) Original Software Stack
(b) Modified Software Stack

[Gjomemo]

# Input Validation Tips

- Always include input checks in your applications
  - What is the range of allowed values for the variables that you are receiving?
  - What is the input's provenance?
  - How is the input processed and used?
  - Does input flow to or influence sensitive operations?
- Minimize input from external sources if possible
  - E.g., don't read and save data from sdcard, use local directory
- Use safe APIs (and learn about them)
  - E.g., use parameterized SQL queries, rather than raw ones

## Complete Sandbox Bypassing (Rooting)

- Rooting grants users (and eventual attackers) complete control over the phone
- Applications are not isolated anymore
- Large number of rooted devices exists (e.g., RomManager has approximately 10 million downloads, CyanogenMod approximately 1 million downloads)
- Very hard to know what is inside a rooted image downloaded from the internet (better use the official Google source code)
- Applications may break

## Android Security Features: Provenance

- *Who wrote the applications I am running? Can I trust them?*
- Developers can self-sign their applications and post them on Google Play
- For a small fee anybody can sign up and post applications on Google Play
- Applications may be installed from alternative web sites (third party stores)
- Since February 2012: Google Bouncer (40% drop in malware in Google Play)

## Private Information Leakage

- How do popular applications deal with PII?
- Static code analysis to discover
  - Misuse of phone identifiers (tracking user behavior)
  - Exposure of physical location
  - Abuse of telephony services
  - Eavesdropping on Audio/video
  - Vulnerabilities
- Methodology
  - Code disassembly, retargeting (to java bytecode), and decompilation
  - Used static code analysis tools to analyze control and data flow
  - 1100 popular applications studied

## Private Information Leakage Research (II)

- Findings
  - 22.4% of applications read phone identifiers
  - Phone identifiers used as device fingerprints and for tracking user behavior (e.g., IMEI bound to search queries, attached to update requests, or on any network request)
  - Phone identifiers often sent to advertisement and analytics servers
  - 45.9% of applications try to access location (only 27% have permission to do so)
  - Location information sent to advertisement servers
  - 9% of apps leak private information in Logs and via Intents

## Android Malware

- Android is currently the most popular target for mobile malware authors
  - 275/277 malware families are for Android
  - 97% of malware in 2016 was for Android, ~3% Nokia.
- Some recent articles:
  - https://arstechnica.com/security/2017/03/preinstalled-malware-targets-android-users-of-two-companies/
  - http://blog.checkpoint.com/2016/07/01/from-hummingbad-to-worse-new-in-depth-details-and-analysis-of-the-hummingbad-andriod-malware-campaign/

Source: https://www.f-secure.com/documents/996508/1030743/Mobile_Threat_Report_Q1_2014.pdf

## Malware Incentives and Behavior (I)

- Information selling
  - Device identification numbers (IMEI)
  - Contact lists and email addresses
  - Browsing history
  - User location
  - Application data
- Fraudulent/aggressive ad campaigns
  - E.g., HummingBad

## Malware Incentives and Behavior (II)

- Stealing financial and other credential (bypassing two-factor authentication)
  - Famous examples: ZitMo, SPITMO
    - http://www.securelist.com/en/blog/208193760/
    - http://www.kaspersky.com/about/news/virus/2011/teamwork_how_the_zitmo_trojan_bypasses_online_banking_security
    - http://blogs.mcafee.com/mcafee-labs/spitmo-vs-zitmo-banking-trojans-target-android
- Intercept SMS messages sent by bank and forward to attacker
- Command and control via web requests or SMS
- Recent botnet client uses Twitter accounts as C&C center

## Malware Incentives and Behavior(III)

- A large percentage of malware places premium rate calls and text messages
- The wildest cases observed in China and Europe
- Malware silently sends text messages to premium rate numbers
- Some variants may intercept SMS status updates and calls from mobile carriers and drop them so users remain unaware

## Malware Incentives and Behavior(IV)

- Direct spamming
  - SMS from a friend more credible than email
  - Commands to malware often sent by SMS
  - Email spamming
- Search engine optimization
  - Malware places searches about a site on search engines to boost the sites ranking
- Government surveillance (E.g., FinFisher)
  - Source: https://www.eff.org/deeplinks/2012/07/elusive-finfisher-spyware-identified-and-analyzed

## Malware Propagation Techniques

- Installation by end users
  - Most end users are not aware of or do not read permissions requested by application
- Code repackaging
  - Attackers repackage malicious code inside legitimate apps and redistribute them using their own certificates
  - Attackers redress malware's UI as another famous application
  - Malware code downloaded by seemingly innocuous application at run time
  - Often present on third party markets but also on Google Play Store

## Malware Propagation Techniques (II)

- Root exploits
  - Rooting communities discover ways to root Android phones quickly
  - Often the malware authors use exploits discovered by the community
  - Study shows every device up to 2011 had a root exploit publicly available for at least 74% of the device lifetime [FELT3]
  - E.g., GameCIH (?)
- Unless phone is already rooted

## Links

- Secure Coding Android (and other languages): https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=111509535
- Dex2Jar: https://code.google.com/p/dex2jar/
- jd-gui: http://jd.benow.ca/
- ApkTool: https://code.google.com/hosting/moved?project=android-apktool
- FlowDroid: http://sseblog.ec-spride.de/tools/flowdroid/

## Links

- Android Source Code: https://source.android.com/source/index.html
- Proprietary Drivers for Google Phones hardware: https://developers.google.com/android/nexus/drivers
- ProGuard (Obfuscation): http://developer.android.com/tools/help/proguard.html
- AndroGuard: https://code.google.com/p/androguard/
- OWASP's Android page: https://www.owasp.org/index.php/OWASP_Mobile_Security_Project_-_Android
- VirusTotal: https://www.virustotal.com/
- Andrototal: http://andrototal.org/

## References


- **[Vidas]** - All Your Droid Belong Are Us
  - https://www.usenix.org/conference/woot11/all-your-droid-are-belong-us-survey-current-android-attacks
- **[Felt1]** - Android Permissions Demystified
  - http://www.cs.berkeley.edu/~dawnsong/papers/2011%20Android%20permissions%20demystified.pdf
- **[Felt2]** - Android Permissions: User Attention, Comprehension, and Behavior
  - http://www.guanotronic.com/~serge/papers/soups12-android.pdf
- **[Felt3]** - A Survey of Mobile Malware in the Wild
  - http://www.cs.swarthmore.edu/~bylvisa1/cs97/f13/Papers/mobilemalware.pdf
- **[Zhou]** - Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets
  - http://www.csc.ncsu.edu/faculty/jiang/pubs/NDSS12_DROIDRANGER.pdf

## References


- **[Chin]** - Analyzing Inter-Application Communication in Android
  - https://www.eecs.berkeley.edu/~daw/papers/intents-mobisys11.pdf
- **[Kantola]** - Reducing Attack Surfaces for Intra-Application Communication in Android
  - http://www.eecs.berkeley.edu/Pubs/TechRpts/2012/EECS-2012-182.html
- **[Enck1]** - TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on smartphones
  - https://www.usenix.org/legacy/event/osdi10/tech/full_papers/Enck.pdf
- **[Enck2]** - A Study of Android Application Security
  - http://www.enck.org/pubs/enck-sec11.pdf

- **[Lu]** CHEX: Statically Vetting Android Apps for Component Hijacking Vulnerabilities
  - http://www.zhichunli.org/publication/CHEX-CCS12.pdf
- **[Gallingani]** Practical Exploit Generation for Intent Message Vulnerabilities in Android (under submission)
- **[Gjomemo]** Digital Check Forgery Attacks on Client Check Truncation Systems
  - http://fc14.ifca.ai/papers/fc14_submission_145.pdf
- **[Aafer-CCS18]** Aafer, Y. et al. Precise Android API Protection Mapping and Reasoning, CCS 2018.