Towards a Trustworthy Android Ecosystem

Yan Chen
Lab of Internet and Security Technology
Northwestern University
Smartphone Security

• Ubiquity
  - Smartphones and mobile devices

  - Smartphone sales already exceed PC sales
  - The growth will continue

• Performance
  - Better than PCs of last decade
  - Samsung Galaxy S4 1.6 GHz quad core, 2 G memory

Source: Canalys estimates © Canalys 2012
### Android Dominance

- Android worldwide market share ~70%
- Android market share in US ~50%

(Credit: Kantar Worldpanel ComTech)

#### OS % Share of Smartphone Sales

<table>
<thead>
<tr>
<th>U.S MARKET</th>
<th>3 mo. ending Feb 12</th>
<th>3 mo. ending Feb 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>iOS</td>
<td>47.0</td>
<td>43.5</td>
</tr>
<tr>
<td>Android</td>
<td>45.4</td>
<td>51.2</td>
</tr>
<tr>
<td>RIM</td>
<td>3.6</td>
<td>0.7</td>
</tr>
<tr>
<td>Windows</td>
<td>2.7</td>
<td>4.1</td>
</tr>
<tr>
<td>Symbian</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>Other</td>
<td>0.8</td>
<td>0.4</td>
</tr>
</tbody>
</table>
Android Threats

• Malware
  – The number is increasing consistently
  – Anti-malware ineffective at catching zero-day and polymorphic malware

• Information Leakage
  – Users often have no way to even know what info is being leaked out of their device
  – Even legitimate apps leak private info though the user may not be aware
New Challenges

• New operating systems
  – Different design $\rightarrow$ Different threats

• Different architecture
  – ARM (Advanced RISC Machines) vs x86
  – Dalvik vs Java (on Android)

• Constrained environment
  – CPU, memory
  – Battery
  – User perception
Problems

- Malware detection
  - Offline
  - Real time, on phone
- Privacy leakage detection
  - Offline
  - Real time, on phone
- OS architecture or application vulnerabilities
- System hardening
  - Access control, ASLR, ...
Problems

• Malware detection
  – Offline
  – Real time, on phone

• Privacy leakage detection
  – Offline
  – Real time, on phone

• OS architecture or application vulnerabilities

• System hardening
  – Access control, ASLR, ...
Our Solutions

• AppsPlayground, CODASPY’13
  – Automatic, large-scale dynamic analysis of Android apps

• DroidChameleon, ASIACCS’13
  – Evaluation of latest Android anti-malware tools

• Uranine
  – Real-time information-flow tracking enabled by offline static analysis
  – With zero platform modification
AppsPlayground

Automatic Security Analysis of Android Applications
AppsPlayground

• A system for offline dynamic analysis
  – Includes multiple detection techniques for dynamic analysis

• Challenges
  – Techniques must be light-weight
  – Automation requires good exploration techniques
Architecture

Exploration Techniques

- Event triggering
- Intelligent input
- Fuzzing

Detection Techniques

- Kernel-level monitoring
- Taint tracking
- API monitoring

Disguise techniques

AppsPlayground
Virtualized Dynamic Analysis Environment
Architecture

Exploration Techniques

- Event triggering
- Intelligent input
- Fuzzing

Contributions

- Disguise techniques

Detection Techniques

- Kernel-level monitoring
- Taint tracking
- API monitoring

AppsPlayground

Virtualized Dynamic Analysis Environment

Contributions
Intelligent Input

- Fuzzing is good but has limitations
- Another black-box GUI exploration technique
- Capable of filling meaningful text by inferring surrounding context
  - Automatically fill out zip codes, phone # and even login credentials
  - Sometimes increases coverage greatly
Privacy Leakage Results

• AppsPlayground automates TaintDroid

• Large scale measurements - 3,968 apps from Android Market (Google Play)
  - 946 leak some info
  - 844 leak phone identifiers
  - 212 leak geographic location
  - Leaks to a number of ad and analytics domains
Malware Detection

• Case studies on DroidDream, FakePlayer, and DroidKungfu

• AppsPlayground’s detection techniques are effective at detecting malicious functionality

• Exploration techniques can help discover more sophisticated malware
DroidChameleon

Evaluating state-of-the-art Android anti-malware against transformation attacks
Introduction

Android malware – a real concern

Many Anti-malware offerings for Android

• Many are very popular

Objective

What is the resistance of Android anti-malware against malware obfuscations?

• Smartphone malware is evolving
  – Encrypted exploits, encrypted C&C information, obfuscated class names, ...
  – Polymorphic attacks already seen in the wild

• Technique: *transform* known malware
## Transformations: Three Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Trivial</strong></td>
<td>• No code-level changes or changes to AndroidManifest</td>
</tr>
<tr>
<td><strong>Detectable by Static Analysis - DSA</strong></td>
<td>• Do not thwart detection by static analysis completely</td>
</tr>
<tr>
<td><strong>Not detectable by Static Analysis – NSA</strong></td>
<td>• Capable of thwarting all static analysis based detection</td>
</tr>
</tbody>
</table>
Trivial Transformations

• Repacking
  – Unzip, rezip, re-sign
  – Changes signing key, checksum of whole app package

• Reassembling
  – Disassemble bytecode, AndroidManifest, and resources and reassemble again
  – Changes individual files
DSA Transformations

- Changing package name
- Identifier renaming
- Data encryption
- Encrypting payloads and native exploits
- Call indirections
- ...

21
Evaluation

• 10 Anti-malware products evaluated
  – AVG, Symantec, Lookout, ESET, Dr. Web, Kaspersky, Trend Micro, ESTSoft (ALYac), Zoner, Webroot
  – Mostly million-figure installs; > 10M for three
  – All fully functional

• 6 Malware samples used
  – DroidDream, Geinimi, FakePlayer, BgServ, BaseBridge, Plankton

• Last done in February 2013.
## DroidDream Example

<table>
<thead>
<tr>
<th></th>
<th>AVG</th>
<th>Symantec</th>
<th>Lookout</th>
<th>ESET</th>
<th>Dr. Web</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repack</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reassemble</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rename package</td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encrypt Exploit (EE)</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rename identifiers (RI)</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Encrypt Data (ED)</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Call Indirection (CI)</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RI+EE</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>EE+ED</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EE+Rename Files</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>EE+CI</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
## DroidDream Example

<table>
<thead>
<tr>
<th></th>
<th>Kasp.</th>
<th>Trend M.</th>
<th>ESTSoft</th>
<th>Zoner</th>
<th>Webroot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repack</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reassemble</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rename package</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Encrypt Exploit (EE)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Rename identifiers (RI)</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Encrypt Data (ED)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Call Indirection (CI)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>RI+EE</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>EE+ED</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>EE+Rename Files</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>EE+CI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>
Findings

• All the studied tools found vulnerable to common transformations
• At least 43% signatures are not based on code-level artifacts
• 90% signatures do not require static analysis of Bytecode. Only one tool (Dr. Web) found to be using static analysis
Signature Evolution

• Study over one year (Feb 2012 – Feb 2013)
• Key finding: Anti-malware tools have evolved towards content-based signatures
• Last year 45% of signatures were evaded by trivial transformations compared to 16% this year
• Content-based signatures are still not sufficient
Content-based Signatures are not sufficient

Analyze semantics of malware

Dynamic behavioral monitoring can help

• Need platform support for that
Takeaways

**Anti-malware vendors**
Need to have semantics-based detection

**Google and device manufacturers**
Need to provide better platform support for anti-malware
Conclusion

• Developed a systematic framework for transforming malware
• Evaluated latest popular Android anti-malware products
• All products vulnerable to malware transformations
Uranine

Real-time Privacy Leakage Detection without System Modification for Android
Motivation

• Android permissions are insufficient
  – User still does not know if some private information will be leaked

• Information leakage is more interesting (dangerous) than information access
  – E.g. a camera app may legitimately access the camera but sending video recordings out of the phone may be unacceptable to the user
Previous Solutions

• Static analysis: not sufficient
  – It does not identify the conditions under which a leak happens.
    • Such conditions may be legitimate or may not happen at all at run time
  – Need real-time monitoring

• TaintDroid: real-time but not usable
  – Requires installing a custom Android ROM
    • Not possible with some vendors
    • End-user does not have the skill-set
Our Approach

• Give control to the user
• Instead of modifying system, modify the suspicious app to track privacy-sensitive flows
• Advantages
  – No system modification
  – No overhead for the rest of the system
  – High configurability – easily turn off monitoring for an app or a trusted library in an app
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>Static Analysis</th>
<th>TaintDroid</th>
<th>Uranine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>Low (possibly High FP)</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Overhead</td>
<td>None</td>
<td>Low</td>
<td>Acceptable</td>
</tr>
<tr>
<td>System modification</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Configurability</td>
<td>NA</td>
<td>Very Low</td>
<td>High</td>
</tr>
</tbody>
</table>
Deployment A

By vendor or 3rd party service
Deployment B

By Market
Overall Scenario

Download → Instrument → Reinstall → Run → Alert User

Unmodified Android Middleware And Libraries
Challenges

• Framework code cannot be modified
  – Policy-based summarization of framework API

• Accounting for the effects of callbacks
  – Functions in app code invoked by framework code
  – Over-tainting techniques guarantee zero FN
Challenges

• Accommodating reference semantics
  – Need to taint objects rather than variables
  – Not interfering with garbage collection

• Performance overhead
  – Path pruning with static analysis
Instrumentation Workflow

App

to IR

Static Analysis

Framework Code Summarization Rules

Instrumentation (taint storage & propagation)

to Bytecode

Instrumented App
Preliminary Results

• Studied 20 apps
• Results in general align with TaintDroid
• Performance
  – Runtime overhead is within 50% for 85% of the apps evaluated and with 100% for all apps
  – Less than 20% instructions need to be instrumented in all apps evaluated
Runtime Performance
Fraction of Instructions Instrumented
Limitations

• Native code not handled
• Method calls by reflection may sometimes result in unsound behavior
• App may refuse to run if their code is modified
  – Currently, only one out of top one hundred Google Play apps did that
Conclusion

• AppsPlayground, CODASPY’13
  – Detected privacy leakage on large scale
  – Capable of detecting malware

• DroidChameleon, ASIACCS’13
  – Several popular Android anti-malware tools shown vulnerable

• Uranine
  – Real-time information-flow tracking with zero platform modification is possible

• More info and tools
  – http://list.cs.northwestern.edu/mobile/
Kernel-level Monitoring

- Useful for malware detection
- Most root-capable malware can be logged for vulnerability conditions
- Rage-against-the-cage
  - Number of live processes for a user reaches a threshold
- Exploid / Gingerbreak
  - Netlink packets sent to system daemons
Security in Software-defined Networks

Towards A Secure Controller Platform for OpenFlow Applications
SDN Architecture

- SDN apps defines routing behavior through controller.
- Current controllers assume full trust on apps, and do not check what apps send to switches.
Threat Model

- Two threat model
  - Exploit of existing benign-but-buggy apps
  - Distribution of malicious apps by attacker
- Plenty of potential attacks
Challenges

• Network resources are architecturally distinctive
  – It is not obvious which resources are dangerous and which ones are safe

• Controller has limited control on SDN apps
  – Only controller API calls go through controller, such as flow addition and statistics query
  – OS system calls do not go through controller, so apps can write whatever they want to the network, storage, etc.
Our Approach

- Permission Check + Isolation
- Contributions
  - Systematic Permission Set Design
  - Comprehensive App Sandboxing
Smartphone Security

• Lots of private data
  – Contacts, messages, call logs, location
  – Grayware applications, spyware applications
  – TaintDroid, PiOS, etc. found many leaks
  – Our independent study estimates about $\frac{1}{4}$th of apps to be leaking

• Exploits could cause user money
  – Dialing and texting to premium numbers
  – Malware such as FakePlayer already do this
Android Threats

• Privacy leakage
  – Users often have no way to know if there are privacy leaks
  – Even legitimate apps may leak private information without informing user

• Malware
  – Number increasing consistently
  – Need to analyze new kinds
## Dynamic vs. Static

<table>
<thead>
<tr>
<th></th>
<th>Dynamic Analysis</th>
<th>Static Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage</td>
<td>Some code not executed</td>
<td>Mostly sound</td>
</tr>
<tr>
<td>Accuracy</td>
<td>False negatives</td>
<td>False positives</td>
</tr>
<tr>
<td>Dynamic Aspects</td>
<td>Handled without additional effort</td>
<td>Possibly unsound for these</td>
</tr>
<tr>
<td>(reflection, dynamic loading)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution context</td>
<td>Easily handled</td>
<td>Difficult to handle</td>
</tr>
<tr>
<td>Performance</td>
<td>Usually slower</td>
<td>Usually faster</td>
</tr>
</tbody>
</table>
Disguise Techniques

- Make the virtualized environment look like a real phone
  - Phone identifiers and properties
  - Data on phone, such as contacts, SMS, files
  - Data from sensors like GPS
  - Cannot be perfect
Exploration Effectiveness

- Measured in terms of code coverage
  - 33% mean code coverage
    - More than double than trivial
    - Black box technique
    - Some code may be dead code
    - Use symbolic execution in the future

- Fuzzing and intelligent input both important
  - Fuzzing helps when intelligent input can’t model GUI
  - Intelligent input could sign up automatically for 34 different services in large scale experiments
Playground: Related Work

- **Google Bouncer**
  - Similar aims; closed system
- **DroidScope, Usenix Security’12**
  - Malware forensics
  - Mostly manual
- **SmartDroid, SPSM’12**
  - Uses static analysis to guide dynamic exploration
  - Complementary to our approach
Threat Mitigation at App level

• Offline analysis
  – Trustworthiness of app is known before use
  – Static analysis
  – Dynamic analysis

• Real-time monitoring
  – Often more accurate but with runtime overhead
  – User has control over app’s actions in real-time
The `toString()` method may be called by a framework API and the returned string used elsewhere.
Potential Defenses against malicious app

- **Server-side Security Check by Controller Vendor**
  - Static analysis
  - Dynamic analysis
- **Runtime Permission Check**
  - Enforce *the principle of least privilege* on apps
- **Principal Isolation**
- **Anomaly-based Behavior Monitoring**