ReDroid: Prioritizing Data Flows and Sinks for App Security Transformation

Ke Tian*, Gang Tan^, Daphne Yao*, Barbara Ryder*

*Department of Computer Science
Virginia Tech

^Department of CSE
Penn State University
Background

The need for application customization is growing

U.S. Department of Defense to Open Its Own App Store

Third Party App Store:
- Security and privacy concerns
- Apps have more security restrictions

The need for application customization is growing

http://mashable.com/2013/10/30/department-of-defense-app-store/#iJuBpfyLJaq4
https://thestack.com/security/2015/02/27
Our goal is to enforce more fine-grained policies via transformation.
Existing Solutions

- Google Dynamic Permission[1]
- RetroSkeleton[2]
  - Method-level rewriting approach
  - Ours
  - Sink-level
  - Sink sensitivity awareness

A policy example: to monitor the most dangerous sink

Low risk

Sink 1: add a bookmark

Sink 2: set time clock

Sink 3: Write to some local storage

High risk

Sink 4: send text messages

Policy: sensitive privacy leakage must be prevented
- How to measure the sensitivity?
- How to enforce the security policy?
The need for sink ranking

Design choices:
1. Sensitive API-based risk vs. Permission-based risk (permission -> score)
2. Sink Rewriting vs. Flow Rewriting (sink)

How to quantify risks of sinks in order to prioritize them?
Our workflow: sink ranking + bytecode rewriting

Program Analysis

1. Data flow graph construction
2. Risk score ranking and computation

Rewrite

3. IR transformation
4. Modification and recompile

Pipeline

1. Data flow graph construction
2. Risk score ranking and computation
3. IR transformation
4. Modification and recompile
We quantify sinks and sources according to their risks of permissions.
Our approach for quantifying the risks of permissions: Machine Learning

A huge Dataset

Extract Feature vectors (permissions)

Train and Classification

<table>
<thead>
<tr>
<th>Permission</th>
<th>Risk Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PHONT_ST</td>
<td>0.140</td>
</tr>
<tr>
<td>READ_SMS</td>
<td>0.107</td>
</tr>
<tr>
<td>INTERNET</td>
<td>0.007</td>
</tr>
</tbody>
</table>
Sensitivity propagations

- **s₁**: getDeviceID
  - **n₁**: PHONE_ST
  - **t₁**: sendTextMes.
    - **SEND_SMS**

- **s₂**: getLocation
  - **n₂**: LOCATION
  - **n₃**: addMessage
    - **READ_SMS**
  - **t₂**: sendHttpPost
    - **INTERNET**
  - **t₃**: sendTextMes.
    - **SEND_SMS**

<table>
<thead>
<tr>
<th></th>
<th>Self permission</th>
<th>Aggregate permission</th>
<th>Risk Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>SEND_SMS</td>
<td>SEND_SMS, PHONE_ST</td>
<td>0.189</td>
</tr>
<tr>
<td>t₂</td>
<td>INTERNET</td>
<td>INTERNET, PHONE_ST</td>
<td>0.147</td>
</tr>
<tr>
<td>t₃</td>
<td>SEND_SMS</td>
<td>SEND_SMS, READ_SMS, LOCATION</td>
<td>0.151</td>
</tr>
</tbody>
</table>
Evaluate and validate risk metrics

jp.co.jags, android.TigerJumping → url.init() → jp.Adlantis (Ads- http)

org.ohny.weekend, org.qstar.guardx and uk.org.crampton.battery → execute() → com.android.Flurry

Geinimi-037c* → sendTextMessage() → SMS trojan

Matching with real-world security reports

<table>
<thead>
<tr>
<th>$T_i$</th>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$T_3$</th>
<th>$T_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>com.ju6.a</td>
<td>uk.co.lilhermit.android.core.Native</td>
<td>com.adwo.adsdk.L</td>
<td>com.adwo.adsdk.i</td>
</tr>
<tr>
<td>$M$</td>
<td>a()</td>
<td>runcmd_wrapper()</td>
<td>a()</td>
<td>a()</td>
</tr>
<tr>
<td>$F$</td>
<td>Android.util.Log int e()</td>
<td>Android.util.Log int e()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$r(T_i)$</td>
<td>0.170</td>
<td>0.156</td>
<td>0.007</td>
<td>0</td>
</tr>
</tbody>
</table>
Next, we describe how our bytecode transformation works on Android apps

Two rewriting strategies for vetting sensitive sinks

- Passive and proactive rewriting strategies
  - System logs

- Proactive dynamic checking & ICC relay
  - App communication channel
Summary of our transformation capabilities

<table>
<thead>
<tr>
<th>Type</th>
<th>Vulnerability</th>
<th>Our Framework Addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inter-app Com. (IAC)</td>
<td>ICC hijacking, Collusion</td>
<td>✓</td>
</tr>
<tr>
<td>Stand-alone App</td>
<td>Privacy Leak, Reflection, String Obfuscation, Dynamic Code Loading</td>
<td>✓ ✓ ✓ ✓</td>
</tr>
</tbody>
</table>

Our Advantages:
- Single app: data leak, privacy leak, detect malware.
- Inter-app Communication: data leak through communication channels
ICC (Inter-Component Communication)

- Communication mechanism in Android
- Pass data and information among Apps
- Can be used for data redirection
An example: detect ICC leakage (ICC Relay)

1. Identify sensitive data flows in an ICC (startActivity with implicit intent)
2. Redirect the intent into a secure proxy app
3. Intent is checking inside the proxy app
4. The communication is relayed from the proxy app
Evaluate rewriting performance

<table>
<thead>
<tr>
<th>App Category</th>
<th>#of ICC Exits</th>
<th>Logging Success</th>
<th>ICC Relay Success</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Re.</td>
<td>In.</td>
</tr>
<tr>
<td>ICCBench</td>
<td></td>
<td>Re.</td>
<td>In.</td>
</tr>
<tr>
<td>icc_implicit_action</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>icc_implicit_category</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>icc_implicit_data</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Icc_implicit_mix</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>icc_implicit_src_sink</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>icc_dynregister</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>DroidBench(IccTA)</td>
<td></td>
<td>Re.</td>
<td>In.</td>
</tr>
<tr>
<td>iac_startActivity</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>icc_startActivity</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>iac_startService</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>iac_broadcast</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Summary</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>

Logging based rewriting achieve high accuracy and robustness

ICC relay based failed two cases because security protection. ICC is efficient in IAC-based protection

Transformation is efficient and valid
Conclusions:

- ReDroid can prioritize and harden Android apps
- Our ML based maximum likelihood mapping from permissions to risks is general
- Experiments confirmed the accuracy of our sink ranking algorithm
- We demonstrated multiple real-world rewriting scenarios

**Check our tool:**
https://github.com/ririhedou/AppRewriting
Thanks!
Android rewriting

- **What?**
  - Rewrite/Instrument Android apps (statically).
  - Edit app bytecode/IR (without source code)
  - Modify app behaviors according to security specifications (defined by analysts or users)

- **Why?**
  - Mitigate vulnerabilities (enhance security)
  - Easily used for security analysis (monitoring)

- **How? A pipeline**
Technical Challenges

- How to identify which part needs to be transformed?

- How to keep the valid execution of apps after transformation?

- How to automate the transformation process?
Our rewriting v.s. state-of-the-art in Android

<table>
<thead>
<tr>
<th>Rewriting Granularity</th>
<th>RetroSkeleton and [10][19]</th>
<th>ReDroid (Ours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package-level (Repackage)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Class-level (Class Inject)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Method-level (Method Invoc.)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ICC-level (Intent Redirect)</td>
<td>–</td>
<td>✓</td>
</tr>
<tr>
<td>Prioritization-based Rewriting</td>
<td>–</td>
<td>✓</td>
</tr>
</tbody>
</table>

- Include Android specific components (inter-component communication)
- More fine-grained control for rewriting specifications
- Introduce flow-/sink- aware analysis to extend rewriting feasibility
- **Motivations**
  1. Security analysts need to analyze (unknown) apps.
  2. Apps are complex (number of sensitive flows & nodes).

![Graphs showing distribution of sensitive flows and nodes.]

**Submit** → **Manual vet** → **PASS/FAIL**