Analyzing ARM Native Code for Tracking Information Flow

Woo-Yeon Lee
Seo-Yoon Choi
Tae-Hun Kim
Byung-Gon Chun

CMS laboratory
Seoul National University
Privacy Leak in Mobile Environment

• Third-party “apps” may leak users’ privacy-sensitive data or manifest malicious behavior
Why do we Target ARM Native Code?

• Platform Environmental Reason
  – Android : 49% of the apps packaged with third-party native library (increasing trend)
  – Tizen : Native apps written as ARM native code.

• Lots of studies about information flow tracking, but not in ARM-instruction level
  – Tainttrace, Panorama, TaintBochs for x86
  – Taintdroid for byte-code level
Approach

• Dynamically monitor ARM native code’s behavior to detect leakage of user’s privacy-sensitive data

• Main Challenge
  – Architecture Dependent
    • ARM’s limited control feature
Taint Tracking

• Technique used to track information dependencies from an origin

• Three Factors
  – Taint Source
  – Taint Propagation
  – Taint Sink

\[
v_1 = \text{taint\_source}() \\
\text{...} \\
v_3 = v_2 + v_1 \\
\text{...} \\
\text{taint\_sink}(v_3)
\]
System Overview

• ARM Binary Analysis Tool (ABAT)
Dynamic Taint Tracking

- Taint Map
  Require fast search
  => Hash table-based taint tag storage
     (Key : address, value : taint tag)
  No data type at the instruction level
  => Taint tags per each byte address
Dynamic Taint Tracking

1. Detect Taint Source
   => Insert new taint into Taint Map

   - Taint Map
     Require fast search
     => Hash table-based taint tag storage
        (Key: address, value: taint tag)

     No data type at the instruction level
     => Taint tags per each byte address
Dynamic Taint Tracking

1. Detect Taint Source
   => Insert new taint into Taint Map

2. Detect Taint Propagation
   => Propagate taint tag in Taint Map

- Taint Map
  Require fast search
  => Hash table-based taint tag storage
     (Key: address, value: taint tag)

No data type at the instruction level
=> Taint tags per each byte address
Dynamic Taint Tracking

1. Detect Taint Source => Insert new taint into Taint Map
2. Detect Taint Propagation => Propagate taint tag in Taint Map
3. Detect Taint Sink => Access to tainted data alerts the data leak

- Taint Map
  - Require fast search
  - Hash table-based taint tag storage
    (Key : address, value : taint tag)

No data type at the instruction level
=> Taint tags per each byte address
ARM Architecture

• Advanced RISC architecture
  – 32bit-fixed instruction length
  – PC is a general register
  – Single execution cycle
  – Conditional execution

• Extension
  – Thumb / Thumb-2 mode (16bit)

• Challenges
  – Implicit branch
  – Restricted features to control program flow
Taint Tracking with DBI

• Inserts additional codes into original application to trace and maintain information about the propagation.

• Handle over 800 ARM instructions:

<table>
<thead>
<tr>
<th>Before Instrumentation</th>
<th>After Instrumentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD Rd, Rn, &lt;immediate&gt;</td>
<td>ADD Rd, Rn, &lt;immediate&gt; MOV τ(Rd),τ(Rn)</td>
</tr>
<tr>
<td>ADD Rd, Rn, Rm</td>
<td>ADD Rd, Rn, Rm OR τ(Rd), τ(Rn), τ(Rm)</td>
</tr>
<tr>
<td>MOV Rd, &lt;immediate&gt;</td>
<td>MOV Rd, &lt;immediate&gt; MOV τ(Rd), 0</td>
</tr>
<tr>
<td>MOV Rd, Rn</td>
<td>MOV Rd, Rn MOV τ(Rd), τ(Rn)</td>
</tr>
</tbody>
</table>
Current status & Future work

• Current status & Future work
  – Finish Basic Implementation
  – Taint tracking module is on implementation and verification stage
  – Reduce overhead with optimized DBI

• Details on poster session