6.858 Quiz 2 Review

Android Security

Haogang Chen
Nov 24, 2014
## Security layers

<table>
<thead>
<tr>
<th>Layer</th>
<th>Role</th>
</tr>
</thead>
</table>
| Reference Monitor| **Mandatory Access Control (MAC) for RPC:**
                  | enforce access control policy for **shared resources**               |
| Java VM          | **Memory safety:**
                  | (neither required nor trusted)                                        |
| Linux Kernel     | **Isolation:**
                  | apps run with different UIDs. (principals are apps, as opposed to users) |
Basic architecture

• Apps are composed of components

• 4 types of components
  • **Activity:** UI, only one active at a time
  • **Service:** background processing, RPC server
  • **Content provider:** provides read/write RPC
  • **Broadcast receiver:** listen for notifications
Intent: RPC primitive

• Has 4 fields
  • **Component:** target
  • **Action:** opcode
  • **Data:** arguments
  • **Category:** for filtering

• The *reference monitor* checks sender’s permission labels upon message delivery.
Permission labels

- **Application** defines permissions as string labels
  - `<permission name="com.android.phone.DIALPERM"></…>`

- Application asks for permissions in its **manifest**
  - `<use-permission name="com.android.phone.DIALPERM"></…>`

- Application assigns a **type** for each permission
# Permission types

<table>
<thead>
<tr>
<th>Type</th>
<th>Reference Monitor’s grant policy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Normal</strong></td>
<td>Silent check, no user interaction required. (no security guarantee for any serious use…)</td>
</tr>
<tr>
<td><strong>Dangerous</strong></td>
<td>Ask the user upon app installation. (useful when you want to interact with others’ apps)</td>
</tr>
<tr>
<td><strong>Signature</strong></td>
<td>Silently grant to apps signed by the same developer. (useful when you only talk to your own apps)</td>
</tr>
</tbody>
</table>
Implicit and broadcast intent

- Implicit intent
  - Omit the “target” field; let Android figure out the receiver
  - Receivers declare interested actions and categories using intent filters

- Broadcast intent
  - Problem: how to ensure only someone gets the broadcast?
  - Solution: protected broadcast (not MAC)
    - Request for a permission when broadcasting
      sendBroadcast(intent, “perm.FRIEND_NEAR”)
Summary

• **Permissions**: “Who are allowed talk to me?”

• **Permission types**: “How to grant permissions to an app?”

• **Intent filters**: “What (implicit intent) do I want to see?”

• **Protected broadcast**: “Who are allowed to see my (broadcast) intent?”
6.858 Quiz 2 Review

TaintDroid

Haogang Chen
Nov 24, 2014
Motivation

• Limitation of the reference manager
  • “What resource can I access?”
  • No guarantee on *how the data is being used*.
    • E.g., a photo editor can silently upload your photo stream to its server
  • TaintDroid: track information flow for sensitive data
Taint tracking basics

- **Source**: origin of sensitive data
  - E.g., photos, contacts, GPS coordinates
- **Sink**: undesired destination
  - E.g., network interface, TCB boundary
- **Propagation**: how information flows from source to sink
  - E.g., variable copy, arithmetic operations, indexing, message passing, system calls, file read/write.
Approach

• Attach a “tag” for each piece of sensitive data

• Propagate the tag together with the data

• Challenges

  • Fine-grained tracking can be extremely slow

  • Coarse-grained tracking introduces false positives

  • Key contribution: trade-offs between performance and accuracy
TaintDroid: multi-level tracking
<table>
<thead>
<tr>
<th>Component</th>
<th>Trusted?</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>System app.</td>
<td>Y</td>
<td><strong>Taint source:</strong> annotate data from sensitive content provider (e.g. camera app)</td>
</tr>
<tr>
<td>User app.</td>
<td>N</td>
<td>User apps runs inside Java VMs. They are untrusted and unmodified</td>
</tr>
<tr>
<td>Java VM</td>
<td>Y</td>
<td><strong>Variable-level tracking:</strong> store and propagate taint tags in shadow memory for every variable</td>
</tr>
<tr>
<td>RPC library</td>
<td>Y</td>
<td><strong>Message-level tracking:</strong> propagate taint tags when serializing/deserializing messages</td>
</tr>
<tr>
<td>System library</td>
<td>Y</td>
<td><strong>Method-level tracking:</strong> annotate how taints propagate among arguments and return values</td>
</tr>
<tr>
<td>Storage library</td>
<td>Y</td>
<td><strong>File-level tracking:</strong> attach and propagate taint tags in file’s extended attribute.</td>
</tr>
<tr>
<td>Network library</td>
<td>Y</td>
<td><strong>Taint Sink:</strong> annotate the interface, and report any tagged data that reaches the sink</td>
</tr>
</tbody>
</table>
Limitation of taint tracking

• Cannot capture control-flow dependencies

```c
// “dirty” is tainted
int laundry(int dirty) {
    int clean;
    if (dirty == 0)
        clean = 0
    else if (dirty == 1)
        clean = 1
    else if (dirty == 2)
        clean = 2
    else ...
    return clean;
}
```