Reptor: Realizing API Virtualization on Android
Taeyeon Ki, Alexander Simeonov, Karthik Dantu, Steven Y. Ko, Lukasz Ziarek
Department of Computer Science and Engineering

Motivation

- Openness spurs innovation in system research.
  - Observation 1: Innovation in app space is widely open to third parties.
  - Observation 2: Distribution of Android platform-level modifications is a stiff barrier to open innovation. Only a select few vendors can control the innovation on Android.

Proposal

- A set of wrapper classes injected into an app
- A wrapper class for each platform API class that a third-party developer wants to replace
- Rewriting the app so that the app code uses wrapper classes instead of platform API classes
- Interception of any and every platform API call made by an app

Reptor Overview

- A third-party developer provides an app binary and custom code that want to be injected as input to Reptor.
- Once Reptor finishes, the developer obtains a modified app with new behavior as intended via the code injection.

Contribution

- Mitigating the lack of openness in mobile systems by proposing a new technique called API Virtualization
- Exploring and addressing a unique set of challenges that API Virtualization brings in order to correctly and completely handle all features of Android and Java.
- Realizing the API Virtualization prototype, and showing its feasibility and practicality

Reptor Performance

- Samsung Galaxy Nexus Devices running Android 4.4
- Stay-awake mode enabled
- CPU governor set to "Performance"

Call Latency to Invoke Android Platform Methods

- To measure call latency, we use a micro-benchmark app that calls eleven platform methods from four categories: device information, network, storage, and sensing (GPS).

Heap Usage of Temple Run

- To measure runtime memory usage, we use one popular game that uses accelerometer and gyroscopes on a mobile phone for game play and contains a heavy UI component.

Power Consumption Measurement

<table>
<thead>
<tr>
<th>App Name</th>
<th>Average Consumption (10 minutes / 5 runs)</th>
<th>Std. Deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>TempleRun</td>
<td>991.5J</td>
<td>29.5J</td>
</tr>
<tr>
<td>TempleRun*</td>
<td>992.95J</td>
<td>22.3J</td>
</tr>
</tbody>
</table>

Instrumentation Statistics

<table>
<thead>
<tr>
<th>App Name</th>
<th>#Class</th>
<th>APK Size</th>
<th>Instrumenting Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>TempleRun</td>
<td>1213</td>
<td>23.8M</td>
<td>N/A</td>
</tr>
<tr>
<td>TempleRun*</td>
<td>2689</td>
<td>24.7M</td>
<td>26.44 (sec)</td>
</tr>
</tbody>
</table>

* Denotes an instrumented app.

Demo Workflow

- Original version Temple Run
- Inserting Custom Code through "Reptor"
- Signing & Installing
- Playing Instrumented Temple Run

Both the non-instrumented and instrumented version are installed on Samsung Galaxy Nexus running Android version 4.4.

- Play the non-instrumented and instrumented version of Temple Run.
- Compare the performance.
- Scan to watch Reptor demo video.

RMS Lab: https://nrcse.buffalo.edu/