Outline - RV-Android

1. Background
2. RV-Android Overview and Build Process
3. Practising RV-Android
4. Demo and take-home exercises
5. Case study: Using monitoring for ad blocking
6. Future work
Background
Android Today

- Market share: over 75%
The Rise of Android’s ads and malware...

- Half of the applications are bundled with ads.
- Android malware increasing.
- Unsatisfactory validation process.
- Libraries share the permissions of the host application.
- Android is a first-class target of mobile malware writers: 5K malware found each day in 2015Q1
Android (from an RV perspective)

- Open-source operating system based on Linux.
- Applications are written in Java (and native code).
- Dalvik Virtual Machine.
- Applications are distributed as APKs.
- Permission system for privileged resources.
- Developer’s Guide
  

- API Reference
  

- SE Android: SELinux support for Android (extended MAC features in the file system)
Android Programming Components

- Activity
  - interactive screen (an action-enabled window)
  - lifecycle and callback methods
- Service (component for long-running ops with no UI)
- Broadcast Receiver (registering for system or app events)
- Content Provider
- Context: handle to the system (environment data)
Aspect-Oriented Programming

- AOP aims to facilitate modularity through the use of **aspects**.
- **Joinpoint**: an identifiable «point» at runtime (e.g., method call).
- **Pointcut**: an expression for matching joinpoints.
  
  \[
  \text{call}(* \text{com.google.ads}..*())
  \]

- **Advice**: piece of code that can be attached to run either after, before, or around a pointcut.
  
  Object \texttt{around}() :
  
  \[
  \text{call}(* \text{com.google.ads}..*()) \{ \text{return null;} \}
  \]

- Implementation for Java: AspectJ.
Related Work

❖ **Static validation techniques**
  ❖ Built in mechanism in Android to grant permissions at installation time.
  ❖ Stonaway (Felt et al.): checking the least privilege principle.
  ❖ ComDroid (Chin et al.): inter-application communication (intents).

❖ **Dynamic validation techniques**
  ❖ TaintDroid (Enck et al.): information-flow properties.
  ❖ Customization of Android OS (Bauer et al.).
  ❖ RV-Droid (Falcone et al.): monitoring and enforcing properties on applications.
  ❖ Instrumenting Android apps with AspectBench Compiler (Bodden et al.).
  ❖ Droid Force (Rasthofer et al.): monitoring/enforcement of system-wide data-centric policies.
RV-Android Overview and Build Process
RV-Android: overview

❖ RV-Android consists of:
  ❖ a tool to generate monitoring libraries
  ❖ a runtime environment for on-device monitoring.
❖ Based on Java-MOP and RV-Monitor:
  ❖ RV-Monitor provides efficient monitoring algorithms
  ❖ AspectJ for instrumentation (other instrumentation methods are possible)
  ❖ Java-MOP generates RV-Monitor lib and aspects
RV-Android: build process

(AspectJ and RV-Monitor runtimes in the monitored and instrumented application)
RV-Android: advantages of the build process

- An easy-to-use and flexible build process.
  - process can take any or a combination of the JavaMOP, RV-Monitor, and aspect inputs.
- Alternative instrumentation methods possible (MOP, abc, manual, etc).
- Leveraging RV-Monitor and Java-MOP:
  - efficient monitoring,
  - plugin architecture.
RV-Android for Developpers

- Two use cases for developers:
  - developing properties for their app
  - checking their app against existing properties

- Two distributions:
  - Developer source edition
    - create Aspect and properties to ship natively with applications
  - Command-line distribution to modify 3rd-party apps
    - `./rv_android [apk] [keystore] [keystore password] [signing key alias] [monitors_directory] [aspects_directory]`
RV-Android for End Users

- For end-users wishing to gain confidence in their applications
RV-Android GUI
Practising RV-Android
Blocking Device Location

- Security property intended to block malicious or questionably-sourced applications from accessing a user’s location through the Android location API.
- Extension: location spoofing and «blurring». 
aspect BlockGetLocation extends Activity {

    // To obtain the context (and have a reference to the created activity)
    private ContextWrapper contextWrapper;
    private Object object;

    Object around(): call(* android.location..*(..)) && !within(BlockGetLocation) {
        String method = thisJoinPoint.getSignature().getName();
        String classe = thisJoinPoint.getSignature().getDeclaringType().getName();
        /* Log information about method and class */
        Object object = thisJoinPoint.getThis();
        Toast.makeText((android.content.ContextWrapper) object).getApplicationContext(),
            "Application accessing location information", Toast.LENGTH_LONG).show();
        return null;
    }
}
// Advice to get the context application
after(): execution(void Activity.onCreate(..))
    && !within(BlockGetLocation) {
    try {
        object = thisJoinPoint.getThis();
        contextWrapper = new ContextWrapper(
        ((android.content.ContextWrapper) object)
            .getApplicationContext());
    } catch (Exception e) {
        // Log error message
    }
}
Logging Network Accesses

- Log and notify the user when an application attempts to use an Android API call related to networking and connections to the Internet.
Logging Network Accesses - AspectJ

aspect WebAspect extends Activity {

    // Android Internet methods
    pointcut webCall() : call(* android.net..*(..))
        || call (* android.webkit.WebView..*(..))
        || call (* java.net.HttpURLConnection..*(..)) && !within(WebAspect);

    // Android creation of an activity
    pointcut onCreate(): execution(* onCreate(..)) && !within(WebAspect);

    // Reference to the context
    private ContextWrapper contextWrapper;

    // Internal activity
    private Activity activity;
}
Logging Network Accesses - AspectJ

```java
after(): webCall() {
    try{
        if (contextWrapper != null && activity != null){
            activity.runOnUiThread(new Runnable() {
                public void run() {
                    // Toast message to inform the user
                    Toast.makeText(contextWrapper.getApplicationContext(), "Application accessing to Internet", Toast.LENGTH_LONG).show();
                }
            });
        } else { /* Log error about the missing context application */ }
    }
} catch(Exception e){
    /* Log exception using thisJoinPoint.getTarget().toString() */
}
```
// Advice to get the context application
after(): onCreate() {
    try {
        activity = new Activity();
        Object object = thisJoinPoint.getThis();

        contextWrapper = new ContextWrapper(((android.content.ContextWrapper) object).getApplicationContext());
    }

    catch (Exception e) { /* Log error message */ }
}
Using Monitoring Oriented Programming (Chen and Rosu)

```java
import java.lang.*;
Android_DenyLocation() {
    // Deny all location package calls
    event location Object around():
        call(* android.location..*(..)) {
            __TOAST("Application accessing location information.");
            __SKIP;
        }
}
```
import java.lang.*;

Android_MonitorNetwork() {

    // Display application toast on all network API calls
    event web_call after(): call (* android.net..*(..))
        || call(* android.webkit.WebView..*(..))
        || call(* java.net.HttpURLConnection..*(..)) {
            __TOAST("Application accessing the Internet");
        }
}
Combining the two previous properties - A MOP Monitor

```java
Android_MonitorLocationNetwork() {
    event web_call Object around(): call(* android.net..*(..)) ||
        call(* android.webkit.WebView..*(..)) ||
        call(* java.net.HttpURLConnection..*(..)) {
            __TOAST("Application accessing the Internet");
        }
    event location Object around(): call(* android.location..*(..)) {
        __TOAST("Application accessing location information.");
    }
    fsm:
    start [location -> unsafe, web_call -> start]
    unsafe [web_call -> deny, location -> unsafe]
    deny [web_call -> deny, location -> unsafe]

    @deny {__SKIP;}
}
```

- All attempts to change file permissions are blocked
  => wide range of potential attack and privilege exploitation exploits are blocked.
A Real Attack

- Generic properties over the API are ineffective against real attacks.
- Tor libraries are commonly used by malware. Tor network allows malware (spyware, adware, and ransomware), to obfuscate its network connections to command and control servers.

```java
public static void copyRawFile(Context ctx, int resid, File file, String mode, boolean isZipd) throws IOException, InterruptedException {
    // Copy file
    // Change the permissions
    Runtime.getRuntime().exec("chmod "+mode+" "+abspath).waitFor();
}
```
Preventing the attack

❖ A simple MOP monitor designed to skip all calls to any exec method in the Runtime class.

❖ no direct execution of shell code in the currently running process

```java
Android_BlockShellExec() {

    // Skip all shell execution calls
    event shell_call Object around(): call(* Runtime+.exec(..)) {
        __SKIP;
    }
}
```
Demo and take-home exercises
Turn your complexity into confidence.
Enforce safety policies, detect policy violations, find race conditions and more.

We provide solutions to analyze your team’s code through cutting-edge program analysis research. Our tools are automatic, generate no false positives, and integrate easily into your team’s existing workflow.

**RV-Monitor** is a runtime monitoring tool that allows for checking and enforcement of safety properties over the execution of your software.

**RV-Predict** is a predictive runtime analysis tool that detects data races in your concurrent software, reporting no false positives.

**RV-Match** is a symbolic execution tool that provides strong, semantics-based correctness guarantees over one, more or all program executions.

We offer the best tools for reliable software development and analysis, by leaders in the field. With both generic and custom solutions, let us help improve your codebase today.
RV-Android

RV-Android allows you to write MOP, AspectJ, and RV-Monitor properties over all binary Android applications. A user friendly, GUI interface featuring RV-Android technology is coming soon.

Download the beta of RV-Android here. See the README for instructions on installing and running RV-Android. A POSIX-compliant system is required.

Example Applications

- SimpleMediaPlayer: RV-Monitor Property Correcting Java API Error (property, instrumentation, application, monitored application)
- OSMTracker: RV-Monitor Property Blocking User Location Leaks (property, application, instrumentation, monitored application)
- OSMTracker: AspectJ Property Logging Application Method Calls (property, application, monitored application)
- AdBlock: AspectJ Property Blocking Third-Party Advertisements (property, application, monitored application)
- Orbot: JavaMOP Property Blocking TOR Network Installation (property, application, monitored application)

Note: Due to an ongoing major overhaul in the JavaMOP project and the difficulty of installing JavaMOP, Example 5 is bundled in the distribution as the equivalent AspectJ property. The property is however manually compatible with JavaMOP.
Running the Examples - Preparation

- Requirements:
  - a shell script interpreter (/bin/sh). RV-Android will thus run only on Linux and other POSIX-compliant systems (eg - OS X). It is likely that RV-Android will work with Cygwin, though we do not support or test on Windows at this time.
  - **both** RV-Monitor and JavaMOP from runtimeverification.com, adding both bin directories to your PATH as specified in the installers and README files.

- The following commands should work in your environment (and do not return not found):
  - rv-monitor -help
  - ajc -help
  - javamop -help

- Install both the JDK and Android SDK, providing the jarsigner command. See the basic setup page of the Android SDK for more information.
Running the Examples

```
chmod +x instrument_apk.sh

./instrument_apk.sh [apk] [keystore] [keystore password] [signing key alias] [monitors_directory] [aspects_directory]
```

- **[apk]** is the relative path to the Android APK you want to instrument
- **[keystore]** is the relative path to an Android keystore ([https://developer.android.com/tools/publishing/app-signing.html#cert](https://developer.android.com/tools/publishing/app-signing.html#cert))
- **[keystore password]** is the keystore's password (sent to standard input of the signing process)
- **[signing_key_alias]** is the alias of the key in the keystore to sign the final application with
- **[monitors_directory]** is the relative path to a directory containing .rvm files describing the monitors (use monitors/ if unsure)
- **[aspects_directory]** is the same as above, but for aspects (.aj AspectJ files)
## Running the Examples

<table>
<thead>
<tr>
<th>Index</th>
<th>Application</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SimpleMediaPlayer</td>
<td>Correcting Java API Error</td>
</tr>
<tr>
<td>2</td>
<td>OSMTracker</td>
<td>Blocking User Location Leaks</td>
</tr>
<tr>
<td>3</td>
<td>OSMTracker</td>
<td>Logging Application Method Calls</td>
</tr>
<tr>
<td>4</td>
<td>Sample/any app</td>
<td>Blocking Third-Party Advertisements</td>
</tr>
<tr>
<td>5</td>
<td>Orbot</td>
<td>Blocking TOR Network Installation</td>
</tr>
</tbody>
</table>

See the README.md in the distribution for a full description of these examples and properties.
Example 1- API property enforcement

- Property being enforced is that "there must be an on-prepared listener set before the media player API can be started".

- A simple demo media player is never properly prepared in the code for playback, meaning the onPrepared listener which should start the music as per the API is never called.

- On-Device, the expected behavior is:
  - that the monitored version does not play music,
  - the unmonitored version does.

- While toyish, this example shows off the ability of RV-Android to enforce arbitrary Android API properties.
Case study:
Using monitoring for ad blocking
Aspect Creation

- Take into account the different mechanisms by which advertisement objects are triggered.

- Challenges:
  - Adware apps use dynamic method invocation to defeat static analysis.
    
    ```java
    Class.forName("com.google.ads.AdActivity")
    .getDeclaredMethod("startActivity") .invoke(null);
    ```
  - Properly allocating and deallocating intercepted objects required by apps (e.g., registering and unregistering broadcast receivers, ...).
Example - aspect for blocking inmobi ads

Object around() : execution(* com.inmobi.androidsdk.*.loadNewAd(..)) { return null; }

Object around(): call(Object java.lang.reflect.Method.invoke(..)) {
    java.lang.reflect.Method target =
        (java.lang.reflect.Method)(thisJoinPoint.getTarget());
    Object[] args = thisJoinPoint.getArgs();
    if (args != null && args.length > 0 && args[0] != null) {
        String receiver = args[0].getClass().getName();
        if (target.getName().compareTo("loadNewAd") == 0 &&
            receiver.startsWith("com.inmobi.androidsdk"))
            return;
    }
    return proceed();
}
Amending the Application

1. APK file is passed to an Ad Analyzer:
   ❖ search through the libraries used by the application,
   ❖ comparison with known advertisement network libraries,
   ❖ selection of aspects to use for ad blocking.

2. APK and the set of aspects required for blocking are passed to Weave Droid:
   ❖ remote or local weaving of the aspects into the application.

3. The output APK contains the ad blocking behavior in it, and therefore fails to display ads.
mi-AdBlocker: about

- Supports devices running on Android version 2.3.3 and higher.
- 2,190 LoC aspect library capable of disabling over 30 different ad network libraries.
- Process:
  - scan of the device for the use of advertisement libraries,
  - suppression of advertisements on a per-application basis.
mi-AdBlocker: demo
Case study: principles

- 860 popular apps from different categories (games, utilities, misc).
- 3 phases to our testing process:
  (any error during a phase implies failure to the phase and the successor phases)
  - Modification: Amending with ad-blocking aspects and repackaging.
  - Execution: Installing, initial launch, and uninstallation.
  - Thorough: All windows were checked to ensure proper functioning.
  (randomized sample)
## Case study: results

<table>
<thead>
<tr>
<th></th>
<th>Modification</th>
<th>Execution</th>
<th>Thorough</th>
</tr>
</thead>
<tbody>
<tr>
<td>Games</td>
<td>341 96.19%</td>
<td>328 85.98%</td>
<td>52 77.61%</td>
</tr>
<tr>
<td>Utility</td>
<td>95 98.95%</td>
<td>94 96.81%</td>
<td>52 94.12%</td>
</tr>
<tr>
<td>Misc.</td>
<td>424 97.41%</td>
<td>413 93.22%</td>
<td>30 100%</td>
</tr>
</tbody>
</table>
Encountered Issues

❖ **Obfuscation:**
   - renaming of methods, fields, etc (not much encountered as ad library should be preserved):
     - dex2jar crashes on names obfuscated with unicode chars,
     - « missing type » errors (calls and type ref without corresponding libraries).
     - encrypting the code (interception of the decrypted code in the class loader).

❖ **Anti-tampering code** (mostly in games).
   - signature verification,
   - package signature verification,
     ```java
     Signature[] sig = getPackageManager().getPackageInfo(app, PackageManager.GET_SIGNATURES).signatures;
     if (sig[0].hashCode() != authenticSignature) fail();
     ```
   - file signature verification => redirect the check to the original file or intercept the call.

❖ **Dynamically-loaded code** => customised class loader that analyses the loaded code.
### Related Ad blocking apps

#### Comparison with Ad-Blocking Software on the Market

<table>
<thead>
<tr>
<th>App</th>
<th>Root</th>
<th>Proxy</th>
<th>Reboot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ad Block plus</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>AirPush Block</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>Adway</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>My InternetSecurity</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>mi-Adblocker</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>
Future Work
Android M

- Android M is the new permission model:
  - blanket permission on categories are not available anymore;
  - permissions can be revoked by the user at any time.
- Android M can replace many of the previous security monitors.
- **However:**
  - RV-Android can alleviate the burden to the user
  - RV-Android allows a more fine-grained use of permissions:
    - RV-Android monitors can interact with the cloud
    - behavioural and context-sensitive permissions
Integrating with Other Approaches

- RV-Android as a **testing** tool:
  - leveraging automated unit testing tools
  - testing a large number of applications automatically
- Integrating with other **dynamic-analysis tools**:
  - Taint analysis to track sensitive information and preventing it from leaving through any unprivileged API calls or other “sinks”.
- Integration with **static analysis**:
  - improve the efficiency of runtime monitors;
  - generate properties to be checked at runtime.
Towards a Public Property Database

- **Objective:** development of properties that thoroughly encompass both known misuses of Android/Java API’s and violations by existing malware.

- A property database with
  - annotated copies of Android/Java API’s with RV-Monitor properties
  - AspectJ instrumentation built in.

- Property database has ca 180 safety properties that can be simultaneously monitored

  https://github.com/runtimeverification/property-db/ (now for Java API only)

- We plan to:
  - provide privacy profiles focused on stronger security guarantees for the security conscious user, (e.g., avoiding data exfiltration when using specific API’s);
  - build a comprehensive open source database covering common property cases, with community contribution and support
Thanks!

- We encourage audience members to try our tools, write their properties, and contribute in any way they can.

- Contact:
  - philip.daian@runtimeverification.com
  - grigore.rosu@runtimeverification.com
  - https://runtimeverification.com
  - https://runtimeverification.com/android/