Easily Instrumenting Android Applications for Security Purposes

Eric Bodden

with a lot of help from:

Steven Arzt
Siegfried Rasthofer
About myself

• Diplom 2005 at RWTH Aachen, Germany
• PhD 2009 at McGill University
  Topic: Static/dynamic analysis of typestate properties / API protocols
  • Used Soot / AspectBench Compiler
• Since 2009: Center for Advanced Security Research Darmstadt (CASED)
• Since 2011: European Center for Security and Privacy by Design (EC SPRIDE)
• Since 2012: Emmy Noether Research Group RUNSECURE
• Since 2013: Professor for Secure Software Engineering at Fraunhofer SIT and TU Darmstadt
Our research

• Like to combine static and dynamic program analysis to get the best out of both worlds
• Previously: API-usage mistakes; reflective method calls
• Currently: application-level taint analysis
Program – The Roadmap for Today

- Android Instrumentation – why, what and how
- The Workshop VM
- Android Platform Overview
- AspectJ
- Tracematches
- Soot and Jimple Overview
- Manual Instrumentation

Coffee break at 15:30
Instrumenting Android and Java Applications as Easy as abc

Steven Arzt, Siegfried Rasthofer, and Eric Bodden
Secure Software Engineering Group, European Center for Security and Privacy by Design (EC SPRIDE), Technische Universität Darmstadt, Germany
{steven.arzt,siegfried.rasthofer,eric.bodden}@ec-spride.de

Abstract. Program instrumentation is a widely used mechanism in different software engineering areas. It can be used for creating profilers and debuggers, for detecting programming errors at runtime, or for securing programs through inline reference monitoring.

This paper presents a tutorial on instrumenting Android applications using Soot and the AspectBench compiler (abc). We show how two well-known monitoring languages – Tracematches and AspectJ – can be used for instrumenting Android applications. Furthermore, we also describe the more flexible approach of manual imperative instrumentation directly using Soot’s intermediate representation Jimple. In all three cases no source code of the target application is required.

Keywords: Android, Java, Security, Dynamic Analysis, Runtime Enforcement.

1 Introduction

According to a recent study [1], Android now has about 75% market share in the mobile-phone market, with a 91.5% growth rate over the past year. With Android phones being ubiquitous, they become a worthwhile target for security and privacy violations. Attacks range from broad data collection for the purpose of targeted advertisement, to targeted attacks, such as the case of industrial espionage. Attacks are most likely to be motivated primarily by a social element: a significant number of mobile-phone owners use their device both for private and work-related communication [2]. Furthermore, the vast majority of users installs apps containing code whose trustworthiness they cannot judge and which they cannot effectively control.

One approach to combat such threats is to augment Android applications obtained from arbitrary untrusted sources with additional instrumentation code. This code alters the behaviour of the target application and can thus enforce certain predefined security policies such as disallowing data leaks of confidential information. Since the instrumentation code runs as an integrated part of the target application, it has full access to the runtime state, thereby avoiding the imprecisions that usually come with static analysis approaches [3–5]. It has full...
Handout is available

- Contains important commands
- Also cheat sheet for German keyboard layout (which the VM uses)
What, why, and how we are going to do it today

INSTRUMENTING ANDROID
What are we going to do?

Instrument Android applications to enforce (security) policies
Why Android?

Who of you owns an Android phone?
Why Android?

Sold mobile devices

Q1/2013

http://www.gartner.com/newsroom/id/2482816

Android
iOS
BlackBerry
Microsoft
Bada
Symbian
Other OSes
What are we going to do?

Instrument Android applications to enforce (security) policies
Why Policies for Android? (1)

- Large variety of sensitive data stored on phone
  - Contacts
  - Emails
  - SMS Messages
  - Photos
  - ...
- Privacy-sensitive sensors built in
  - GPS
  - Camera
  - Microphone
  - ...
Why Policies for Android? (2)

- Various threats already appeared “in the wild”
  - Malware sending costly premium SMS messages
  - Private data leaking to ad companies and adversaries
  - Phones used for tracking people
  - Phones being part of bot networks
  - Exploiting phone by root exploits
  - ...
What are we going to do?

**Instrument** Android applications to enforce (security) policies
Android Stack

Applications
System Apps + User Apps

Application Framework
Different Managers (e.g., for Activity, Content, Location, etc.)

Libraries
Sqlite, OpenGL, SSL

Runtime
Dalvik VM, Core libs

Linux Kernel
Display, camera, wifi, audio, ...
Why Bytecode Instrumentation?

• Detect vulnerabilities at runtime
• Monitor application behavior
• Enforce security policies

• Advantages of application-level bytecode instrumentation:
  • No application source code necessary
  • No phone-rooting necessary
  • No modification of the OS necessary
  • OS version independent
  • Instrumentation is JIT-compiled
Android Stack: Application-Layer Security

Applications
System Apps + User Apps

Application Framework
Different Managers (e.g., for Activity, Content, Location, etc.)

Libraries
Sqlite, OpenGL, SSL

Runtime
Dalvik VM, Core libs

Linux Kernel
Display, camera, wifi, audio, ...

07.11.13 | Secure Software Engineering Group | Eric Bodden | 17
Virtual Machine for the Hands-On Lab

THE WORKSHOP VM
Installing VirtualBox

- Windows version on our USB Stick!
- Else go to: https://www.virtualbox.org/wiki/Downloads
  - Download latest VirtualBox for your system
  - Download latest VirtualBox Extension Pack

- Install VirtualBox 4.2.16
- Install VirtualBox 4.2.16 Extension Pack

- Any later version should do as well
VM Setup

• Copy prepared VM files from stick to hard disk
  • May take some time, hard disk is about 8.3 GB
  • Do not run the VM from the stick!

• Go to „Machine -> Add -> …RV 2013.vbox

• Run the virtual machine as it is
  • Better: 2 GB of RAM and sufficient graphics memory
    if you can spare it
What we give you

• Virtual machine running on VirtualBox or VMWare
  • Available for Windows, Linux, Mac OS

• Debian 7.1 with Gnome
• Eclipse KEPLER
• Android SDK
  • Preconfigured Android Emulator
• Soot and abc
Getting Started with the VM

• Log in as user “rv2013” with password “rv2013”

• Eclipse and Android SDK manager are in the launcher
  • Look under “Programming”
  • **Launch Emulator before Eclipse! Don’t close it!**

• Soot and abc installed to /opt/soot
• Android SDK installed to /opt/android-sdk-linux
• RV Sample App is in ~/RV2013Examples/exampleApp
The Android Emulator Inside a VM

- Emulator known to be slow
  - Normally uses hardware acceleration
  - Hardware support not available in a VM

- Tricks you can do:
  - Start the emulator before you start Eclipse
  - Leave the emulator running
  - Use snapshots (configured in our VM)
A Quick Look at the VM
Running Android Applications

• Use the run/debug buttons in Eclipse
  • Emulator is default target
  • Use “Run Configurations” dialog if you want a real phone

• Use the command line:
  • `adb install RV2013.apk`
  • `adb uninstall de.ecspride`

Uninstall uses package name!
A platform from development to release

ANDROID PLATFORM OVERVIEW
Activity Lifecycle

Activity starts

onCreate() → onStart() → onResume() → onPause() → onStop() → onDestroy()

onRestart() → onDestroy()

Activity is running

Activity is shut down
public class RV2013 extends Activity {

    private EditText phoneNr, message;
    private SmsManager smsManager = SmsManager.getDefault();

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_rv2013);
        Log.i("INFO", "in onCreate");
    }

    public void sendSms(View v){
        Log.i("INFO", "in sendSms");
        phoneNr = (EditText)findViewById(R.id.phoneNr);
        message = (EditText)findViewById(R.id.message);

        smsManager.sendTextMessage(phoneNr.getText().toString(), null, message.getText().toString(), null, null);
    }
}
APK Build Process

Android Project → Compilation and Packaging → Android Package (.apk)

- .dex
- resources .arsc
- uncompiled resources
- AndroidManifest.xml

Device or Emulator → Signing

APK File

- Layout-File
- Signatures
- Meta-Information
- Bytecode
- Compiled Resources
Exploring and installing SMS Messenger example

LAB SESSION
The Task

1. Scale the emulator, start it

2. Open the RV2013 app in Eclipse
   It should already be in your workspace

3. Install it on the emulator

4. Play around with it and look for Logcat outputs in Eclipse
Solution: Emulator
**Solution: LogCat**

<table>
<thead>
<tr>
<th>Level</th>
<th>Time</th>
<th>PID</th>
<th>TID</th>
<th>Application</th>
<th>Tag</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>08-13 16:17:17.583</td>
<td>297</td>
<td>620</td>
<td>system_process</td>
<td>ActivityManager START u0 {act=android.intent.action 0000 cmp=de.ecspride/RV2013} from</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>08-13 16:17:17.802</td>
<td>297</td>
<td>620</td>
<td>system_process</td>
<td>dalvikvm</td>
<td>GC_FOR_ALLOC freed 276K, 15% free ;</td>
</tr>
<tr>
<td>I</td>
<td>08-13 16:17:17.813</td>
<td>297</td>
<td>620</td>
<td>system_process</td>
<td>dalvikvm-heap</td>
<td>Grow heap (frag case) to 8.076MB fc</td>
</tr>
<tr>
<td>D</td>
<td>08-13 16:17:17.922</td>
<td>297</td>
<td>314</td>
<td>system_process</td>
<td>dalvikvm</td>
<td>GC FOR_ALLOC freed 146K, 19% free ;</td>
</tr>
<tr>
<td>E</td>
<td>08-13 16:17:18.422</td>
<td>37</td>
<td>126</td>
<td></td>
<td>SurfaceFlinger:ro.sf.lcd_density must be defined in onCreate</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>08-13 16:17:19.002</td>
<td>659</td>
<td>659</td>
<td>de.ecspride</td>
<td>INFO</td>
<td>in onCreate</td>
</tr>
<tr>
<td>E</td>
<td>08-13 16:17:19.762</td>
<td>37</td>
<td>37</td>
<td></td>
<td>SurfaceFlinger:ro.sf.lcd_density must be defined in onCreate</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>08-13 16:17:19.913</td>
<td>659</td>
<td>659</td>
<td>de.ecspride</td>
<td>EGL_emulation eglSurfaceAttrib not implemented</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>08-13 16:17:20.422</td>
<td>297</td>
<td>314</td>
<td>system_process</td>
<td>ActivityManager Displayed de.ecspride/Rv2013: +2s;</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>08-13 16:17:21.022</td>
<td>389</td>
<td>389</td>
<td>com.android.input.Choreographer</td>
<td>Skipped 53 frames! The application</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>08-13 16:17:21.472</td>
<td>297</td>
<td>297</td>
<td>system_process</td>
<td>HostConnection::get() New Host Conr</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>08-13 16:17:21.793</td>
<td>37</td>
<td>126</td>
<td></td>
<td>SurfaceFlinger:ro.sf.lcd_density must be defined in onCreate</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>08-13 16:17:21.932</td>
<td>389</td>
<td>389</td>
<td>com.android.input.EGL_emulation eglSurfaceAttrib not implemented</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>08-13 16:17:22.334</td>
<td>419</td>
<td>419</td>
<td>com.android.launcher.Choreographer</td>
<td>Skipped 69 frames! The application</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>08-13 16:17:33.092</td>
<td>389</td>
<td>389</td>
<td>com.android.input.Choreographer</td>
<td>Skipped 33 frames! The application</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>08-13 16:17:33.952</td>
<td>389</td>
<td>389</td>
<td>com.android.input.Choreographer</td>
<td>Skipped 30 frames! The application</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>08-13 16:17:35.992</td>
<td>860</td>
<td>860</td>
<td>de.ecspride</td>
<td>dalvikvm</td>
<td>GC CONCURRENT freed 137K, 8% free ;</td>
</tr>
<tr>
<td>I</td>
<td>08-13 16:17:40.432</td>
<td>659</td>
<td>659</td>
<td>de.ecspride</td>
<td>INFO</td>
<td>in sendSms</td>
</tr>
</tbody>
</table>
Which Policies Do We Want to Enforce?

EXAMPLE POLICIES
Example Policies

• Recall: Application sends SMS messages
  • May cost money
  • May be used for spamming

• Policy 1: Do not send messages to 0900 numbers
  • May cost much more than just the normal SMS charges

• Policy 2: Do not send more than three messages to same number
  • Could be considered as spam
Policy 1: No Premium SMS Messages

• Policy 1: Do not send messages to 0900 numbers

• Idea:
  • Intercept all calls to SmsManager.sendTextMessage()
  • If phone number starts with 0900, raise an alert
  • Otherwise, proceed as normal

• Can be done using all the tools
  • Most straightforward pick: AspectJ
Policy 2: A Closer Look

- Policy 2: Do not send more than three messages to same number

- Idea:
  - Intercept all calls to SmsManager.sendTextMessage()
  - On every call, increment a counter by 1
    - If the counter below or equal to 3, proceed normally
    - If the counter exceeds 3, raise an alert and block

- Can be done using all the tools
  - Most straightforward pick: Tracematches
Our Toolkit: Tracematches, abc, and Soot

- Tracematches
- abc Runtime
- abc
- Soot
- toDex
- Dexpler
The Pointcut/Advice Model for Android Applications

ASPECTJ
The Pointcut-Advice Model

void SmsManager.sendTextMessage(…)

When to do something?

Instrument AspectJ

Instrumented Application

Pointcut

Advice

Android App

What to do?
AspectJ: How It Works

Three phases for generating the instrumented application:

- **Matching**: Which pointcut applies where?
- **Weaving**: Place the pieces of advices
- **Code Generation**: Generate final APK file
Instrumentation with AspectJ

```java
public void sendSms(View v) {
    phoneNr = (EditText) findViewById(R.id.phoneNr);
    message = (EditText) findViewById(R.id.message);

    smsManager.sendTextMessage(phoneNr.getText().toString(), null, message.getText().toString(), null, null);
}
```
import android.telephony.SmsManager;
import android.app.PendingIntent;
import android.util.Log;

public aspect SendSMS_3sms {
    pointcut sendSms() : call (void SmsManager.sendTextMessage
        (String, String, String, PendingIntent, PendingIntent));
}

Pointcut
AspectJ: A Simple Example (1)

```java
import android.telephony.SmsManager;
import android.app.PendingIntent;
import android.util.Log;

public aspect SendSMS_3sms {
  pointcut sendSms() :
    call (* SmsManager.sendTextMessage(..));

  after(): sendSms() {
    Log.i("Aspect", "SMS message sent.");
  }
}
```

Pointcut

"after" advice
public void sendSms(View v) {
    phoneNr = (EditText)findViewById(R.id.phoneNr);
    message = (EditText)findViewById(R.id.message);

    smsManager.sendTextMessage(phoneNr.getText().toString(), null, message.getText().toString(), null, null);

    Log.i("Aspect", "SMS message sent.");
}

import android.telephony.SmsManager;
import android.app.PendingIntent;
import android.util.Log;

public aspect SendSMS_3sms {
  pointcut sendSms(String no) : call (* SmsManager.sendTextMessage(..))
    && args(no, ..);

  after(String no): sendSms(no) {
    Log.i("Aspect", "SMS message sent to no. " + no);
  }
}
**AspectJ: Placeholder Semantics**

```
pointcut sendSms(String no) : call (* SmsManager.sendTextMessage(..)) && args(no, ..);

any arguments
```

```
pointcut sendSms(String no) : call (* SmsManager.sendTextMessage(..)) && args(no, *

1 further argument
```
Recap on Policy 1: No Premium SMS msgs.

- Policy 1: Do not send messages to 0900 numbers

- Idea:
  - Intercept all calls to SmsManager.sendTextMessage()
  - If phone number starts with 0900, raise an alert
  - Otherwise, proceed as normal

- We need to replace the original code
  - “around” advice: instead-of, with the ability to “proceed” to original code
Policy 1: No Premium SMS Messages

```java
import android.telephony.SmsManager;
import android.app.PendingIntent;
import android.util.Log;

public aspect SendSMS_PremiumAspect {
  pointcut sendSms(String no) : call (void SmsManager.sendTextMessage(..)) && args(no, ..);

  void around(String no): sendSms(no) {
    if (no.startsWith("0900"))
      Log.e("Aspect", "Premium SMS message blocked.");
    else
      proceed(no);
  }
}
```
Recap on Policy 2: Prevent SMS Spam

• Policy 2: Do not send more than three messages to the same number

• Idea:
  • Intercept all calls to SmsManager.sendTextMessage()
  • On every call, increment a counter by 1
  • If the counter below or equal to 3, proceed normally
  • If the counter exceeds 3, raise an alert and block
Policy 2: No SMS Spam

import ...

public aspect SendSMS_PremiumAspect {
    Map<String, Integer> counter = new HashMap<String, Integer>();

    pointcut sendSms(String no) : call (void SmsManager.sendTextMessage(..)) && args(no, ..);

    void around(String no): sendSms(no) {
        if (counter.containsKey(no)) counter.put(no, counter.get(no) + 1); else counter.put(no, 1);
        if (counter.get(no) > 3)
            Log.e("Aspect", "SMS spam message blocked.");
        else
            proceed(no);
    }
}
AspectJ – Running abc

```
java -cp /opt/soot/abc-ja-exts-complete.jar abc.main.Main
   -cp android-support-v4.jar:android.jar:abc-ja-exts-complete.jar
   -ext abc.ja
   -android
   -injars ~/RV2013Examples/exampleApp/RV2013/bin/RV2013.apk
SendSMS_PremiumAspect.aj
```

Use SendSMS_PremiumAspect.sh
AspectJ in abc

- File name and aspect name must match
  - Extension must be “aj” or “java”
- Use the JastAdd frontend, not Polyglot!
  - Option -ext abc.ja
- Look for warnings in abc’s output
Run an Aspect for Android

LAB SESSION
The Task

Create an aspect that only allows 3 SMS messages per premium number, but an unlimited number of messages to normal numbers.

Tip: Combine the aspects for the two policies.
Copy and then modify existing .sh and .aj file.

Tip: The files are located under

~/RV2013Examples/aspectsAndTMs/aspects
~/RV2013Examples/exampleApp
Solution: A Combined Policy

import ...

public aspect SendSMS_PremiumAspect {
    Map<String, Integer> counter = new HashMap<String, Integer>();

    pointcut sendSms(String no) :
        call (void SmsManager.sendTextMessage(..)) && args(no, ..);

    void around(String no): sendSms(no) {
        if (no.startsWith("0900")) {
            if (counter.containsKey(no)) counter.put(no, counter.get(no) + 1); else counter.put(no, 1);
            if (counter.get(no) > 3) Log.e("Aspect", "Premium SMS message blocked.");
            else proceed(no);
        } else proceed(no);
    }
}
How to test instrumentation?

- Install on Real Phone (SMS cost money!)
- Install on Emulator
- Check Logcat Output
- Do not forget to:
  - Sign the APK
  - Zipalign the APK
Limitations of AspectJ

- Use around advice to block policy violations
  - Does not remove dependent code / “backwards slice”
  - Example: Remove all debug outputs, computation of debug values remains

- No global reasoning about the program
  - Premium SMS messages may only be sent to numbers entered by the user

- Monitors for sequences cumbersome to implement
  - Remember the map for the counts per phone number
  - Can we do better?
Our Toolkit: Tracematches, abc, and Soot
Sequence-Based Monitoring in Android Applications

TRACEMATCHES
Tracematches: The Paper

Adding trace matching with free variables to AspectJ

Chris Allan, Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Ondrej Lhotak, Oege de Moor, Damien Sereni, Ganesh Sittampalam and Julian Tibble

OOPSLA 2005

http://dl.acm.org/citation.cfm?id=1094839
Recap on Policy 2: Prevent SMS Spam

- Policy 2: Do not send more than three messages to same number

- Looks like an automaton
  - “SMS message sent” is an event
  - Use states for counting
  - Normal states (s0, .. s3), alert state s4

- Use one automaton per phone number
  - Always the same structure, we just need a single blueprint
Policy 2: The Automaton

Policy 1: Do not send more than three messages to same number

Finite-state automata can be expressed as regular expressions!

send, send, send, send+

send[3] send+
Policy 2: Declarative State Machine Defs.

- Tracematches handles the automaton for us!
  - Declaratively instrument apps with automaton-based monitors
  - Regular expression defines the monitor
  - If the monitor automaton accepts, user-defined code is run
  - No custom bookkeeping for automaton required!

- Allows for much more concise definition of policy 2
Policy 2: The Big Picture

Automaton / RegExp: When to do something?

Instrument

Android App

Code: What to do?

Instrumented Application
import android.telephony.SmsManager;
import android.app.PendingIntent;
import android.util.Log;

public aspect SMSSpam {
    tracematch (String no) {
        sym sendSms after:
            call (void SmsManager.sendTextMessage(..)) && args (no,..);
        sendSms[3] sendSms+ {
            Log.e("SPAM", "SMS spam detected to no: " + no);
        }
    }
}

No manual bookkeeping required
Tracematches in abc

- File name and aspect name must match
  - Extension must be “aj” or “java”
- Must redefine symbols for each tracematch
  - But can reuse pointcuts
- Use the JastAdd frontend, not Polyglot!
- Enable the TM extension! -ext abc.ja.tm
- Look for “symbol never matches” warnings
  - Good first hint at what has gone wrong
Tracematches – Running abc

```java
java -cp /opt/soot/abc-ja-exts-complete.jar abc.main.Main
    -cp android-support-v4.jar:android.jar:abc-ja-exts-complete.jar
    -ext abc.ja.tm
    -android
    -injars ~/RV2013Examples/exampleApp/RV2013/bin/RV2013.apk
SendSMS_3sms.aj
```

Use SendSMS_3sms.sh
The Task

Change the tracematch such that it prevents SMS spam instead of just reporting it.

Tip: Use an “around” advice. You don’t need to call “proceed” since your code is only called in the alert state.

Folder is:

~/RV2013Examples/aspectsAndTMs/tracematches
Solution: Tracematches – Prevent SMS Spam

```java
import android.telephony.SmsManager;
import android.app.PendingIntent;
import android.util.Log;

public aspect SMSSpam {
    void tracematch(String no) {
        sym sendSmsA around(no):
            call (void SmsManager.sendTextMessage(..)) && args (no,..);

        sendSmsA[3] sendSmsA+ {
            Log.e("SPAM", "SMS spam prevented to no: " + no);
        }
    }
}
```
Tracematches – Limitations

- Tracematches only support finite state machines / regular expressions
- Tracematches cannot share symbol definitions
- No possibility of custom bookkeeping inside the automaton
  - Not possible to enforce more complex privacy policies
The Machinery Behind It All

SOOT AND JIMPLE
What is Soot?

• a free compiler infrastructure, written in Java (LGPL)
• used by hundreds of researchers worldwide, both in academia and industry
• was originally designed to analyze and transform Java bytecode
• original motivation was to provide a common infrastructure with which researchers could compare analyses (points-to analyses)
• has been extended to include decompilation and visualization
• now fully supports reading and writing dalvik bytecode by two modules called Dexpler and toDex
The Soot Framework

Input / Output

- .dex
- .java
- .jimple
- .xml
- .apk
- .class
- .shimple
- .grimple
- .jasmin
- .baf

Soot: Analyze, Optimize, Tag

Jimple IR | Callgraph | Basic Transformations
Soot: Packs and Phases

Whole Program Packs

Body Packs
Soot: Transformers (1)
Soot: Transformers (2)

PackManager.v().getPack("jtp").add(new Transform("jtp.myAnalysis", new MyBodyTransformer()));

public class MyBodyTransformer extends BodyTransformer {
    protected void internalTransform(Body b, String phaseName, Map options) {
        ...
    }
}

method's body

phase-name (e.g., jtp.myAnalysis)

Settings (-p phase opt:val)
Soot: Libraries and Applications

- Application classes: All classes in the process directory (the APK)
  - These classes are loaded including all method bodies
  - These classes get transformed and written out again

- Library classes: Class referenced from application classes
  - Includes the full hierarchy of referenced classes
  - Only method signatures are loaded
  - These classes are NOT transformed, nor written out

- No other classes are loaded by default!
Phantom classes

Phantom classes model classes that Soot cannot find on its classpath…

- Can contain phantom methods and phantom fields
  - Created on demand as required to type check
- Phantom methods have no body

- Phantom classes mark the boundaries of the “known world” – Hic sunt dracones
Obtaining Soot

• For the brave of heart: Compile it from source
  • https://github.com/Sable/heros
  • https://github.com/Sable/jasmin
  • https://github.com/Sable/soot

• The easy way: Download the nightly build
  • http://vandyk.st.informatik.tu-darmstadt.de/abc/soot.jar

• Do NOT use the outdated 2.5.0 release version!
Running Soot on the Command Line

```
java -jar /opt/soot/soot.jar
    -allow-phantom-refs
    -android-jars /opt/android-sdk-linux/platforms
    -src-prec apk
    -process-dir ~/RV2013Examples/exampleApp/RV2013/bin/
    RV2013.apk
    -output-format jimple

use runSoot.sh
```
public static void initializeSoot() {
    soot.Main.main(new String[] {
        "-allow-phantom-refs",
        "-validate",
        "-output-format", "dex",
        "-process-dir", Settings.apk,
        "-force-android-jar", Settings.androidJAR,
        "-src-prec", "apk",
        "-cp", Settings.androidJAR
    });
}
Running Soot using Code (2)

```java
public static void initializeSoot(args) {
    G.reset();

    Options.v().set_allow_phantom_refs(true);
    Options.v().set_output_format(Options.output_format_jimple);
    Options.v().set_process_dir(Collections.singletonList(apk));
    Options.v().set_android_jars(androidJAR);
    Options.v().set_src_prec(Options.src_prec_apk);

    soot.Main.main(args);
}
```
Soot and Platform Versions

- Soot will use Android JAR defined in manifest
- Option 1: Install correct platform version
- Option 2: Force specific platform version:

    -force-android-jar /opt/android-sdk-linux/platforms/android-17/android.jar
The Output

Jimple files in “sootOutput” folder

One for each application class
The Jimple IR

• Jimple: Like Java, but Simple

• One file per class

• Variable-based three-operand language
  • $x = a + b \times c$; becomes $t = b \times c$; $x = a + t$
  • No operand stack, just local variables
  • No complex nested statements

• Optimized for static analysis and instrumentation
public class de.ecsprise.RV2013 extends android.app.Activity {
    private android.widget.EditText message;
    private android.widget.EditText phoneNr;
    private android.telephony.SmsManager smsManager;
    ...
}

public class RV2013 extends Activity {
    private EditText phoneNr, message;
    private SmsManager smsManager = SmsManager.getDefault();
    ...
}

But what happened to the initializer?
The Jimple IR – Explicit Constructors

public void <init>() {

dec.scpride.RV2013 $r0;
android.telephony.SmsManager $r1;

$r0 := @this: de.ecspride.RV2013;
specialinvoke $r0.<android.app.Activity: void <init>()>();

$r1 = staticinvoke <android.telephony.SmsManager: android.telephony.SmsManager getDefault>();

$r0.<de.ecspride.RV2013: android.telephony.SmsManager smsManager> = $r1;
return;
}

Constructor

Locals

"this" local

Super call

Static invoke

Field assignment

Explicit return
The Jimple IR - Statements
The Jimple IR – Expressions (1)
The Jimple IR – Expressions (2)

- DivExpr - soot.jimple
- MulExpr - soot.jimple
- OrExpr - soot.jimple
- RemExpr - soot.jimple
- ShrExpr - soot.jimple
- ShlExpr - soot.jimple
- SubExpr - soot.jimple
- UsrExpr - soot.jimple
- XorExpr - soot.jimple
- CastExpr - soot.jimple
- InstanceOfExpr - soot.jimple
- InvokeExpr - soot.jimple
- DynamicInvokeExpr - soot.jimple
- InstanceInvokeExpr - soot.jimple
- StaticInvokeExpr - soot.jimple
- NewArrayExpr - soot.jimple
- NewExpr - soot.jimple
- NewMultiArrayExpr - soot.jimple
- UnopExpr - soot.jimple

Press 'Ctrl+T' to see the supertype hierarchy.
Soot, Polyglot, JastAddJ

• Several packages contain classes / interfaces with the same name

• Make sure to only use soot.jimple.*

• Do not reference the following:
  • soot.jimple.internal.*
  • soot.JastAddJ.*
  • polyglot.*
Manual Instrumentation

SOOT AND JIMPLE
Step 1: New Body Transformer

PackManager.v().getPack("jtp").add(
    new Transform("jtp.myAnalysis", new MyBodyTransformer()));

soot.Main.main(new String[] { ... });
Step 2: Iterating over classes and methods

@Override
protected void internalTransform(Body body, String arg0, Map arg1) {

    Iterator<Unit> i = body.getUnits().snapshotIterator();
    while (i.hasNext()) {
        Unit u = i.next();
        //do something
    }
}
Adding/Removing Statements

... 

Jimple Statement 1

insertBefore(newStmt, stmt)

Jimple Statement 2

insertAfter(newStmt, stmt)

Jimple Statement 3

remove(stmt)

Jimple Statement 4
Removing Statements

```java
while (i.hasNext()) {
    Stmt s = (Stmt)i.next();
    if (s.containsInvokeExpr()) {
        String declaringClass =
            s.getInvokeExpr().getMethod().getDeclaringClass().getName();
        if (declaringClass.equals("android.util.Log"))
            body.getUnits().remove(s);
    }
}
...
```

- check for invoke expressions
- get the class name
- check for a specific class
while (i.hasNext()) {
    Stmt s = (Stmt)i.next();
    if (s.containsInvokeExpr()) {
        String declaringClass =
            s.getInvokeExpr().getMethod().getDeclaringClass().getName();
        String name = s.getInvokeExpr().getMethod().getName();

        if (declaringClass.equals("android.telephony.SmsManager") &&
            name.equals("sendTextMessage")) {
            List<Unit> toastStmts = makeToast(body, "here");
            body.getUnits().insertBefore(toastStmts, s);
        }
    }
}
The Task

Before every call to `sendTextMessage`, check whether the phone number is a 0900 number. In case of a constant 0900 number just remove the statement otherwise skip the call. If it is not a 0900 number, proceed.

Tip:

$z0 = \text{virtualinvoke } r3.<\text{java.lang.String: boolean startsWith(java.lang.String)}>("0900")
if $z0 == 1 \text{ goto } \text{nop}

\text{virtualinvoke } r6.<\text{android.telephony.SmsManager: void sendTextMessage(...)}>\text{nop}
Solution: Premium-Rate SMS Check

VirtualInvokeExpr vinvokeExpr = generateStartsWithMethod(body, phoneNumberLocal);

...
AssignStmt astmt = Jimple.v().newAssignStmt(localBoolean, vinvokeExpr);
generated.add(astmt);

...
EqExpr equalExpr = Jimple.v().newEqExpr(localBoolean, one);
NopStmt nop = insertNopStmt(body, u);

...
IfStmt ifStmt = Jimple.v().newIfStmt(equalExpr, nop);

...
body.getUnits().insertBefore(generated, u);
Important trick

To try out an instrumentation scheme before actually implementing it, proceed as follows:

• Convert APK to .jimple files (-f jimple)
• “Instrument” .jimple files manually by editing them in your favorite text editor
• Convert .jimple files back into a .dex file (-src-prec jimple)
• Reassemble APK
• Test the generated APK
• If it works, automate the instrumentation
Simplifying instrumentation with

CUSTOM RUNTIME LIBRARIES
public static void maketoast(Context context, String message) {
    Toast.makeText(context, message, Toast.LENGTH_LONG).show();
}
Without runtime library

Jimple Statement 1

\[ \text{insertBefore}(\text{newStmt}, \text{stmt}) \]

\[ <\text{Complex code}> \]

Jimple Statement 2

\[ <\text{Complex code}> \]

\[ \text{insertAfter}(\text{newStmt}, \text{stmt}) \]

Jimple Statement 3

Jimple Statement 4

...
With runtime library

...  

Jimple Statement 1

\[\text{insertBefore}(\text{newStmt}, \text{stmt})\rightarrow \text{Library.myCode1}(\text{foo}, \text{bar});\]

Jimple Statement 2

\[\text{Library.myCode2}(\text{bar}, \text{baz}); \leftarrow \text{insertAfter}(\text{newStmt}, \text{stmt})\]

Jimple Statement 3

Jimple Statement 4

...
Using a custom runtime library...

• Move complex code into hand-written reusable library
• Instrumentation simply calls (static) library methods

• Helps you reuse complicated code
• Advantage:
  • Instead of generating the code, can just write it directly
  • In most cases virtually no added runtime cost
• Disadvantage:
  • Less flexible instrumentation
Runtime libraries and Soot

Must put your library on Soot’s classpath:

- `java -cp soot.jar MyInstrMain -cp myLib.jar ...

In your custom main class, add “basic classes” to make Soot aware of them:

```java
class MyInstrMain {
    public static void main(String[] args){
        Scene.v().addBasicClass("de.ecspride.Library");
        ...
        soot.Main.main(args);
    }
}
```
How do I get my library into the APK?

As easy as abc...

• Soot can put it into the APK for you

• Within your transformer, the “basic class” is readily accessible:
  
  ```java
  c = Scene.v().getSootClass("de.ecspride.Library")
  ```

• To have it inlined into the output APK, simply flag it as a so-called
  “application class”:
  
  ```java
  c.setApplicationClass();
  ```

• Because Soot outputs all application classes, this will cause it to output
  the library class as well.

• If your library contains many classes, do this for each one.
Speeding up instrumented applications

STATIC OPTIMIZATIONS
import ...

public aspect BlacklistURLs {
    pointcut openUrl (String tgt) :
        execution (URLConnection+.new(String)) && args (tgt);
    around (String tgt): openUrl (tgt) {
        List<String> blacklist = downloadBlacklist();
        if (!blacklist.contains(tgt))
            proceed(tgt);
    }
}
Optimizations - Techniques

• Do expensive work only once
  • Initialize variables, use singletons
  • Aspect is no more fully declarative, but a lot faster

• Only instrument where you really need to
  • Statically reason about the program first
  • Instrumentation costs at every run, static analysis only once

• Minimize the number of events to be tracked / pointcuts to be monitored
Optimizations Available in Soot

• Use existing transformers in Soot
  • Constant propagation and folding
  • Copy propagation
  • Conditional branch folding
  • Unconditional branch folding
  • Dead code elimination
  • Dead assignment elimination

• Soot already applies some transformations when loading DEX code
WRAP-UP / CONCLUSION
Recap: What We Have Covered

- The Android platform and its tools
- Instrumenting apps with AspectJ
- Instrumenting apps with Tracematches
- The Soot framework
- Manually instrumenting apps with Soot
Thanks!

• Siegfried Rasthofer and Steven Arzt

• The “abc gang”:
  Pavel Avgustinov, Julian Tibble, Laurie Hendren, Oege de Moor

• The “Dexpler gang”:
  Alexandre Bartel, Jacques Klein, Martin Monperrus, Yves Le Traon

• For “toDex”:
  Thomas Pilot

• All of the many contributors to Soot and abc.
Get Started on Your Own Projects
Eric Bodden
Secure Software Engineering Group (EC SPRIDE)

Email: eric.bodden@ec-spride.de
Blog: http://sse-blog.ec-spride.de
Website: http://sse.ec-spride.de