Reconstructing Dalvik applications

Marc Schönefeld

University of Bamberg

SyScan’09
# Agenda

1. Introduction
2. Dalvik development from a RE perspective
3. Parsing Strategy
4. Processing the results
5. Finalizing

Marc Schönefeld (University of Bamberg) - Reconstructing Dalvik applications
Marc Schönefeld

- since 2002 Talks on Java-Security on intl. conferences (Blackhat, RSA, DIMVA, PacSec, CanSecWest, HackInTheBox)
- day time busy for Red Hat (since 2007)
- PhD Student at University of Bamberg (since 2005)
As a reverse engineer I have the tendency to look in the code that is running on my mobile device

Coming from a JVM background I wanted to know what Dalvik is really about

Wanted to learn some yet another bytecode language

I prefer coding to doing boring stuff, like filling out tax forms
• Dalvik is the runtime that runs userspace Android applications
• invented by Dan Bornstein (Google)
• named after a village in Iceland
• register-based
• runs own bytecode dialect (not java bytecode)
## Dalvik vs. JVM

<table>
<thead>
<tr>
<th></th>
<th>Dalvik</th>
<th>JVM</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Architecture</strong></td>
<td>Register</td>
<td>Stack</td>
</tr>
<tr>
<td><strong>OS-Support</strong></td>
<td>Android</td>
<td>Multiple</td>
</tr>
<tr>
<td><strong>RE-Tools</strong></td>
<td>few</td>
<td>many</td>
</tr>
<tr>
<td><strong>Executables</strong></td>
<td>APK</td>
<td>JAR</td>
</tr>
<tr>
<td><strong>Constant-Pool</strong></td>
<td>per Application</td>
<td>per Class</td>
</tr>
</tbody>
</table>

Introduction

Marc Schönefeld

Pre-Talk

Agenda

Introduction

Introduction

Intro

Dalvik development from a RE perspective

Parsing Strategy

Static analysis of the code

Processing the results

Decompile the code

Finalizing

Built with open source tools
Dalvik Development process

- Dalvik apps are developed using java developer tools on a standard desktop system (like eclipse),
- compiled to java classes (javac)
- transformed to DX with the dx tool (classes.dex)
- classes.dex plus meta data and resources go into a dalvik application ’apk’ container
- this is transferred to the device or an emulator (adb, or download from android market)
Dalvik Development process

Marc Schönefeld

Pre-Talk
Agenda
Introduction

Introduction
Dalvik development from a RE perspective

Parsing Strategy
Static analysis of the code

Processing the results
Decompile the code

Finalizing
Built with open source tools

Marc Schönefeld (University of Bamberg)  Reconstructing Dalvik applications  SyScan’09  8 / 34
## Dalvik runtime libraries

<table>
<thead>
<tr>
<th></th>
<th>Dalvik</th>
<th>JVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.io</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>java.net</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>android.*</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>com.google.*</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>javax.swing.*</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **Marc Schönefeld** (University of Bamberg)  
- **Reconstructing Dalvik applications**  
- **SyScan’09**
Dalvik applications are available as apk files, no source included, so you buy/download a cat in the bag. How can you find out, whether

- the application contains malicious code, ad/spyware, or phones home to the vendor?
- has unpatched security holes (dex generated from vulnerable java code)?
- contains copied code, which may violate GPL or other license agreements?
Reconstructing Dalvik applications

Marc Schönefeld

Pre-Talk
Agenda
Introduction

Introduction
Intro

Dalvik development from a RE perspective

Parsing Strategy
Static analysis of the code

Processing the results
Decompile the code

Finalizing
Built with open source tools

Dalvik development from a RE perspective

Marc Schönefeld (University of Bamberg)  Reconstructing Dalvik applications  SyScan’09  11 / 34
Reconstructing Dalvik applications

Marc Schönefeld

Pre-Talk
Agenda
Introduction

Introduction
Intro

Dalvik development from a RE perspective

Parsing Strategy
Static analysis of the code

Processing the results
Decompile the code

Finalizing
Built with open source tools

Filling the gap

*java
jad/dava
undx
*class/*.jar
javac
dx
*.dex
Tool design choices

- How to parse dex files?
  - write a complicated DEX parser
  - or utilize something existing
- How to translate to class files (bytecode library)?
  - ASM
  - BCEL
The dexdump tool of the android sdk can perform a complete dump of dex files, it is used by undx

<table>
<thead>
<tr>
<th></th>
<th>dexdump</th>
<th>parsing directly</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Speed</strong></td>
<td>Time advantage, do not have to write everything from</td>
<td>Direct access to binary structures (arrays, jump tables)</td>
</tr>
<tr>
<td><strong>Control</strong></td>
<td>dexdump has a number of nasty bugs</td>
<td>Immediate fix possible</td>
</tr>
<tr>
<td><strong>Available info</strong></td>
<td>Filters a lot</td>
<td>All you can parse</td>
</tr>
</tbody>
</table>

The decision was to use as much of useable information from dexdump, for the rest we parse the dex file directly.
• This is useful dexdump output, good to parse
• This is useful dexdump output, omitting important data

```plaintext
name : '<clinit>'
type : '(V
access : 0x10008 (STATIC CONSTRUCTOR)
code -
registers : 1
ins : 0
couts : 0
insn size : 34 16-bit code units
000310: 1200
000320: 6000 0200
000326: 1300 1000
00032a: 2300 0f00
00032e: 2600 0700 0000
000334: 6900 0000
000338: 0e00
00033a: 0000
00033c: 0003 0200 1000 0000 3000 3100 3200 ... 000e: array-data (20 units)
catches : (none)
positions :
0x0000 line=7
0x0003 line=8
```
### Parsing Strategy

**Extract classes.dex from *.apk file**

**Parse global structures (constants)**

**For each class in dex**

**Parse class meta data**

**For each method in class**

**Parse method meta data**

**For each instructions in method**

**Transform to java bytecode**

**Generate java method (BCEL method)**

**Generate java class (BCEL method)**

**Store class in jar**

---

### Pre-Talk Agenda

**Introduction**

- Dalvik development from a RE perspective

**Parsing Strategy**

- Static analysis of the code

**Processing the results**

- Decompile the code

**Finalizing**

- Built with open source tools

- We chose the BCEL library from Apache as it has a very broad functionality (compared to alternatives like ASM and javassist)
- Extract global meta information
- Transform into relevant BCEL constant structures
- Retrieve the string table to prepare the Java constant pool
Process classes

- Transform each class
- Parse Meta Data
- Process methods
- Generate BCEL class
- Dump class file

### Parsing Strategy

**Static analysis of the code**

<table>
<thead>
<tr>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>extract classes.dex from *.apk file</td>
</tr>
<tr>
<td>parse global structures (constants)</td>
</tr>
<tr>
<td><strong>for each class in dex</strong></td>
</tr>
<tr>
<td>parse class meta data</td>
</tr>
<tr>
<td><strong>for each method in class</strong></td>
</tr>
<tr>
<td>Parse method meta data</td>
</tr>
<tr>
<td><strong>for each instructions in method</strong></td>
</tr>
<tr>
<td>transform to java bytecode</td>
</tr>
<tr>
<td>generate java method (BCEL method)</td>
</tr>
<tr>
<td>generate java class (BCEL method)</td>
</tr>
<tr>
<td>store class in jar</td>
</tr>
</tbody>
</table>
Reconstructing Dalvik applications
Marc Schönefeld

Pre-Talk
Agenda
Introduction

Introduction
Intro

Dalvik development from a RE perspective

Parsing Strategy
Static analysis of the code

Processing the results
Decompile the code

Finalizing
Built with open source tools

Marc Schönefeld (University of Bamberg) Reconstructing Dalvik applications

---

**Process class Meta Data**

- Extract Class Meta Data
- Visibility, class/interface, classname, subclass
- Transfer static and instance fields

<table>
<thead>
<tr>
<th>Parsing Strategy</th>
<th>Static analysis of the code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process class Meta Data</td>
<td>Raleigh, NC, USA</td>
</tr>
<tr>
<td>for each class in dex</td>
<td>Raleigh, NC, USA</td>
</tr>
<tr>
<td>parse class meta data</td>
<td>Raleigh, NC, USA</td>
</tr>
<tr>
<td>for each method in class</td>
<td>Raleigh, NC, USA</td>
</tr>
<tr>
<td>Parse method meta data</td>
<td>Raleigh, NC, USA</td>
</tr>
<tr>
<td>for each instructions in method</td>
<td>Raleigh, NC, USA</td>
</tr>
<tr>
<td>transform to java bytecode</td>
<td>Raleigh, NC, USA</td>
</tr>
<tr>
<td>generate java method (BCEL method)</td>
<td>Raleigh, NC, USA</td>
</tr>
<tr>
<td>generate java class (BCEL method)</td>
<td>Raleigh, NC, USA</td>
</tr>
<tr>
<td>store class in jar</td>
<td>Raleigh, NC, USA</td>
</tr>
</tbody>
</table>

---
### Parsing Strategy

**Process the individual methods**

- Extract Method Meta Data
- Parse Instructions
- Generate JAVA method

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>extract classes.dex from *.apk file</td>
</tr>
<tr>
<td>2</td>
<td>parse global structures (constants)</td>
</tr>
<tr>
<td>3</td>
<td>for each class in dex</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>for each method in class</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>generate java method (BCEL method)</td>
</tr>
<tr>
<td>6</td>
<td>generate java class (BCEL method)</td>
</tr>
<tr>
<td>7</td>
<td>store class in jar</td>
</tr>
</tbody>
</table>
### Parsing Strategy

**Parse Method Meta Data**

- **extract classes.dex from *.apk file**
- **parse global structures (constants)**
- **for each class in dex**
  - **parse class meta data**
  - **for each method in class**
    - **Parse method meta data**
    - **for each instructions in method**
      - transform to java bytecode
      - generate java method (BCEL method)
    - generate java class (BCEL method)
  - generate class in jar

- **transform method meta data to BCEL method structures**
- **extract method signatures,**
- **set up local variable tables,**
- **map Dalvik registers to JVM registers**
Reconstructing Dalvik applications

Marc Schönefeld

Pre-Talk
Agenda
Introduction

Introduction
Intro

Dalvik development from a RE perspective

Parsing Strategy
Static analysis of the code

Processing the results
Decompile the code

Finalizing
Built with open source tools

Generate the instructions

- first create BCEL InstructionList
- create NOP proxies for every Dalvik instruction to prepare jump targets (satisfy forward jumps)
- For every Dalvik instruction add an equivalent JVM bytecode block to the JVM InstructionList

<table>
<thead>
<tr>
<th>Parsing Strategy</th>
<th>Generate the instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>extract classes.dex from *.apk file</td>
<td></td>
</tr>
<tr>
<td>parse global structures (constants)</td>
<td></td>
</tr>
<tr>
<td>for each class in dex</td>
<td></td>
</tr>
<tr>
<td>parse class meta data</td>
<td></td>
</tr>
<tr>
<td>for each method in class</td>
<td></td>
</tr>
<tr>
<td>Parse method meta data</td>
<td></td>
</tr>
<tr>
<td>for each instructions in method</td>
<td></td>
</tr>
<tr>
<td>transform to java bytecode</td>
<td></td>
</tr>
<tr>
<td>generate java method (BCEL method)</td>
<td></td>
</tr>
<tr>
<td>generate java class (BCEL method)</td>
<td></td>
</tr>
<tr>
<td>store class in jar</td>
<td></td>
</tr>
</tbody>
</table>
Store generated data in BCEL structures

- generate the BCEL structures
- store to current context
- in the end we have a class file for each defined class in the dex
Reconstructing Dalvik applications

Marc Schönefeld

Pre-Talk
Agenda
Introduction

Introduction
Intro

Dalvik development from a RE perspective

Parsing Strategy
Static analysis of the code

Processing the results
Decompile the code

Finalizing
Built with open source tools

Store generated data in BCEL structures

Dalvik

```java
public static MD5 getMD5() {
  MD5 md5 = new MD5();
  md5.setMessage("hello world");
  return md5.getDigest();
}
```

JVM code

Challenges

- Assign Dalvik regs to jvm regs
- obey stack balance rule (when processing opcodes)
- type inference (reconstruct flow of data assignment opcodes)
Reconstructing Dalvik applications

Marc Schonefeld

Pre-Talk
Agenda
Introduction

Introduction
Intro

Dalvik development from a RE perspective

Parsing Strategy
Static analysis of the code

Processing the results
Decompile the code

Finalizing
Built with open source tools

Store generated data in BCEL structures

Dalvik

JVM code

public static MD5 getInstance();

Code:
0: getstatic #14; //Field md5:LMD5;
3: astore_0
4: aload_0
5: ifnonnull 20
8: new #4; //class MD5
11: astore_0
12: aload_0
13: invokespecial #73; //Method "<init>"():V
16: aload_0
17: putstatic #14; //Field md5:LMD5;
20: getstatic #14; //Field md5:LMD5;
23: astore_0
24: aload_0
25: areturn
Now we have bytecode, what to do with it?

- Analyze the code with static checking tools (findbugs)
- Programming bugs, vulnerabilities, license violations
Reconstructing Dalvik applications

Marc Schönefeld

Pre-Talk
Agenda
Introduction

Introduction
Intro

Dalvik development from a RE perspective

Parsing Strategy
Static analysis of the code

Processing the results
Decompile the code

Finalizing
Built with open source tools

Processing the results
Decompile the code

Now we have bytecode, what to do with it?

Decompile it!

• Feed the generated jar into a decompiler
• It will spit out JAVA-like code
• Structural equal to the original source (but some differences due to heavy reuse of stack variables)

Marc Schönefeld (University of Bamberg) Reconstructing Dalvik applications SyScan’09 29 / 34
Now we have bytecode, what to do with it?

Graph it!

- Findbugs comes with a control-flow-graph analyzer
- Generate nodes and arrays (50 locs)
- Write DIA file
- enjoy that you can reuse tools from the java world
Now we have bytecode, what to do with it?

Graph it!

- Findbugs comes with a control-flow-graph analyzer
- Generate nodes and arrays (50 locs)
- Write DIA file
- Enjoy that you can reuse tools from the java world
Some smaller facts

Hard Facts and Trivia

- 4000 lines of code
- written in JAVA, only external dependency is BCEL
- command line only
- licensing is **GPL** (look out for undx on fedorahosted soon)
- will be published after having tested successfully with recent cupcake binaries and optimized dex files
- **undx** name suggested by Dan Bronstein
• Thank you for your attention
• Time for Q & A
• or send me a mail

marc.schoenefeld -at- gmx DOT org
This presentation was build with open source tools:

- Fedora 10
- Latex
- Beamer
- OpenJDK