Program Transformation for Aiding Static Analysis in Android Applications

Presented by Zhenyu Ning

Outline

1. Background
2. Motivation
3. Related work
4. Implementation
5. Evaluation
6. Future work

Outline

1. Background
2. Motivation
3. Related work
4. Implementation
5. Evaluation
6. Future work

Android JVM

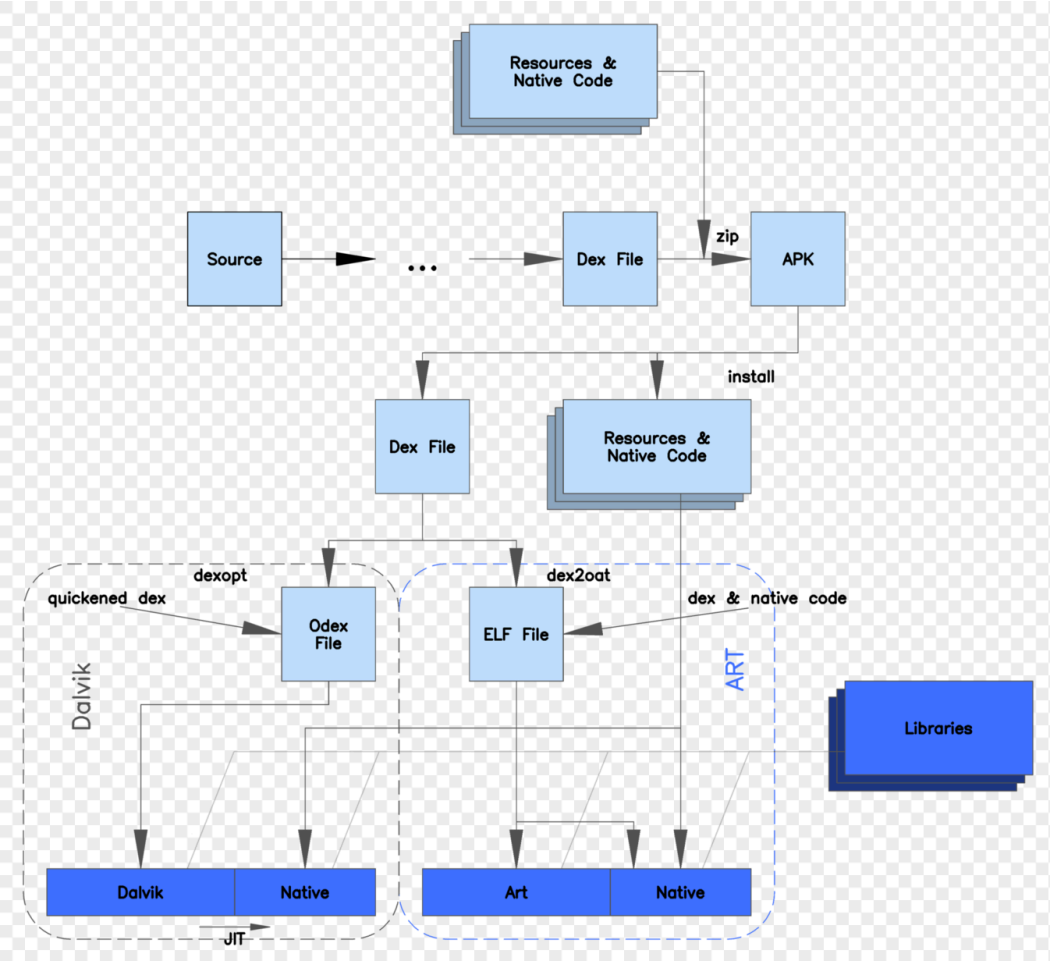


Figure from: https://en.wikipedia.org/wiki/Dalvik_(software)

Application Analysis

- Static Analysis

  Decompile the application, and analyze its byte codes.

  tools: dex2jar, jd-gui, etc.

- Dynamic Analysis

  Execute the application in an isolated execution environment, and analyze the execution.

  tools: Android emulator, QEMU, etc.

Background

- Static analysis

  FlowDroid, DroidSafe, HornDroid

- Dynamic analysis

  DroidScope, TaintDroid, TaintART

- Hybrid analysis

  Harvester

Outline

1. Background
2. Motivation
3. Related work
4. Implementation
5. Evaluation
6. Future work

Motivation

- Static analysis tools suffer from
  - Code obscuration and packing
  - Self-modifying code
- Dynamic analysis tools suffer from
  - Implicit taint flows
  - Performance vs. accuracy
  - Large-scale analysis

Motivation

- Use dynamic analysis to solve packed and self-modifying code.
- Use static analysis to detect implicit flows
- Make the analysis applicable in large-scale analysis.

Outline

1. Background
2. Motivation
3. Related work
4. Implementation
5. Evaluation
6. Future work

Relative Work

- DexHunter
  - Dump Dex file from memory
- AppSpear
  - Use runtime data structure to rebuild Dex file

Assume there exists a clear boundary between packer's code and the application's code

```
1  package com.test;
2
3  public class Main extends Activity {
4    private static final String PHONE = "800-123-456";
5    protected void onCreate(Bundle savedInstanceState) {
6      // ...
7      leak();
8    }
9
10   public void leak() {
11     String a = getSensitiveData(); // source
12     for (int i = 0; i < 2; ++i) {
13       normal(a);
14       bytecodeTamper(i);
15     }
16   }
17
18   public void normal(String param) {
19     // do something normal
20   }
21
22   public void sink(String param) {
23     // send param through text message.
24     SmsManager.getDefault().sendTextMessage(PHONE, null,
           param, null, null); // sink
25   }
26
27   /* While i = 0:
28    *  modify Line 11 to String a = "non-sensitive data"
29    *  modify Line 13 to sink(a)
30    * While i = 1:
31    *  modify Line 11 to String a = getSensitiveData()
32    *  modify Line 13 to normal(a) */
33   public void native bytecodeTamper(int i);
34 }
```

Outline

1. Background
2. Motivation
3. Related work
4. Implementation
5. Evaluation
6. Future work

Implementation

- Just-In-Time instruction-level collection
- Offline reassembling

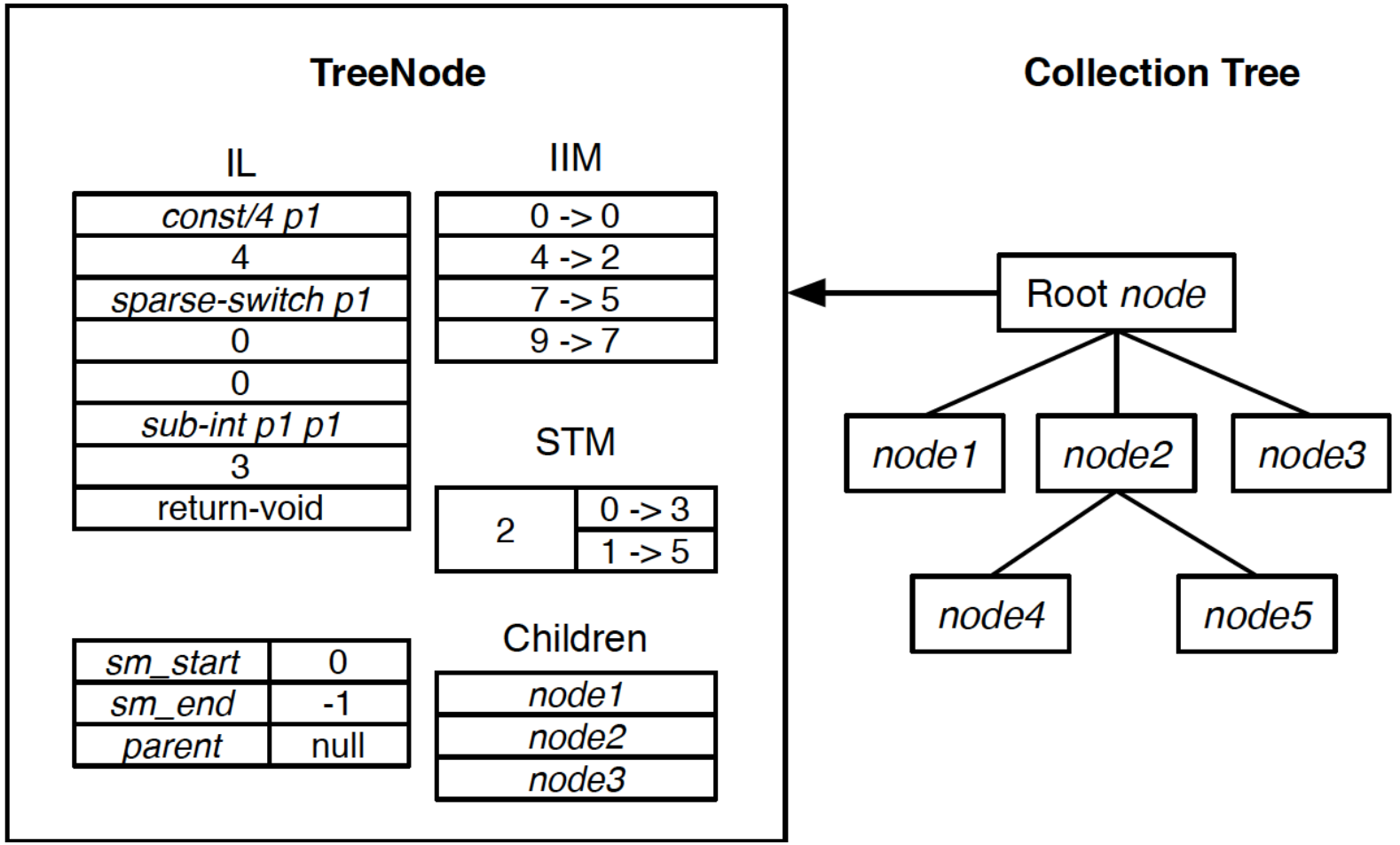
Implementation

- Code scale
  - Loops
- Self-modifying code

Implementation

- The bytecode of a method is organized in an 16-bit array
- A variable *dex_pc* indicates the index of the executing instruction
- **Solution**: Compare instructions with same *dex_pc*

Implementation



Implementation

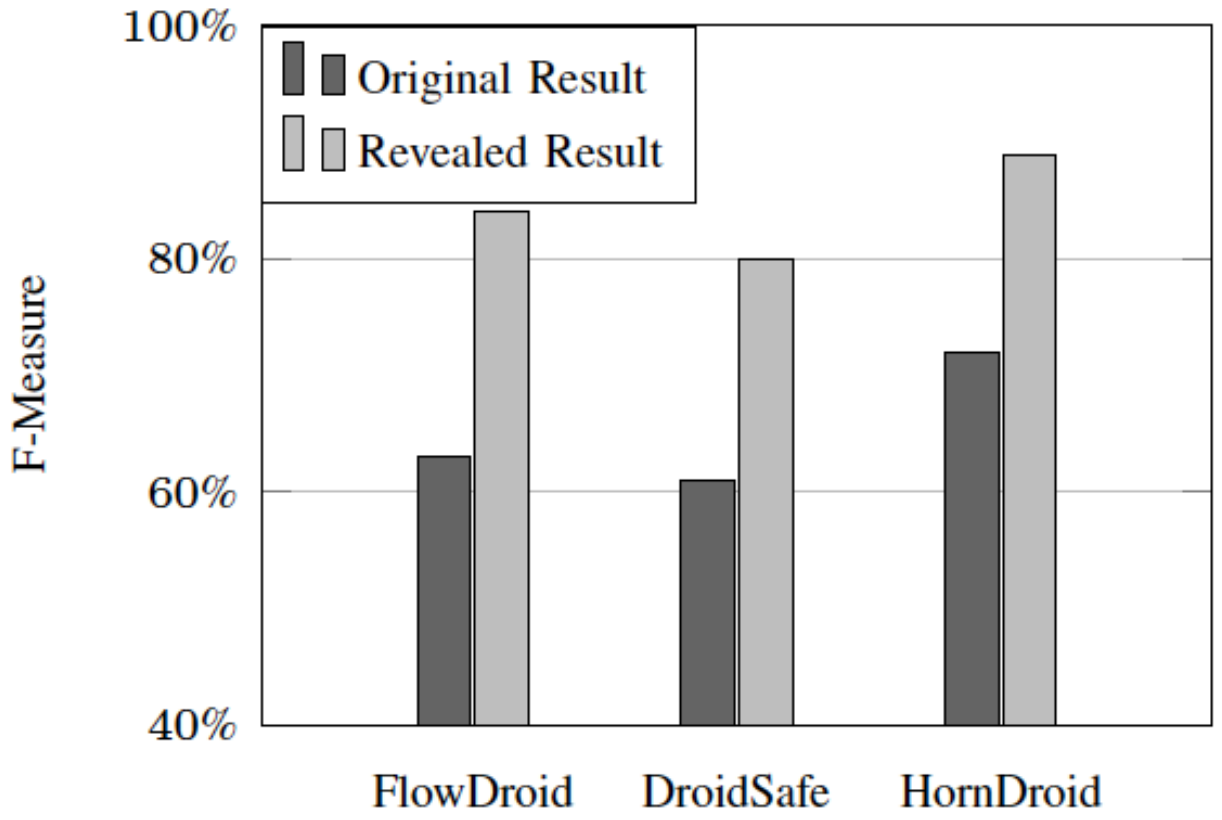
```
1  Root Node:
2    String a = getSensitiveData();
3    for (int i = 0; i < 2; ++i) {
4      normal(a);
5      bytecodeTamper(i);
6    }
7
8  Child Node: (Line 13 in Code 1)
9    sink(a);
```

```
1  String a = getSensitiveData();
2  for (int i = 0; i < 2; ++i) {
3    if (EXPRESSION) {
4      normal(a);
5    } else {
6      sink(a)
7    }
8    bytecodeTamper(i);
9  }
```

Outline

1. Background
2. Motivation
3. Related work
4. Implementation
5. Evaluation
6. Future work

Evaluation on DroidBench



Evaluation on DroidBench

Samples	Leak #	# of Leak Detected		
		TaintDroid [15]	TaintART [39]	DROIDREVEALER
Button1	1	0	0	1
Button3	2	0	0	2
EmulatorDetection1	1	0	1	1
ImplicitFlow1	2	0	0	2
PrivateDataLeak3	2	1	1	1

Evaluation on real-world apps

Package Name	Version	Sample Set	# of Installs	Original	Revealed
com.lenovo.anyshare	3.6.68	A	100 million	0	4
com.moji.mjweather	6.0102.02	A	1 million	0	5
com.rongcai.show	3.4.9	A	100 thousand	0	3
com.wawoo.snipershootwar	2.6	B	10 million	0	4
com.wawoo.gunshootwar	2.6	B	10 million	0	5
com.alex.lookwifipassword	2.9.6	B	100 thousand	0	2
com.gome.eshopnew	4.3.5	C	15.63 million	0	3
com.szzc.ucar.pilot	3.4.0	C	3.59 million	0	5
com.pingan.pabank.activity	2.6.9	C	7.9 million	0	14

Outline

1. Background
2. Motivation
3. Related work
4. Implementation
5. Evaluation
6. Future work

Future work

- Code coverage
- Native code
- Regular JVM in x86

Thank you!