http://tracedroid.few.vu.nl

Introduction

Application Fundamentals

Apps are written in Java, executed by a VM

Building blocks:
- *Activity* — Single screen with a UI
- *Service* — Background components
- *Receiver* — Listener for specific announcements, e.g., *boot completed, sms received*

Distributed as signed jar files (.apk)

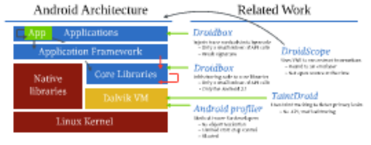
Scope

Limit method tracing to Java code
- Interesting features only accessible via Java
- Existing tools for tracing native code
  - strace
  - ltrace

Use dynamic analysis
- Evade obfuscation
- Existing tools for static analysis
  - *Androguard*
  - *Dexter*

Android Architecture

Related Work



Android Profiler
- Trace app internally
- Trace app to application framework
- Trace app to core libraries

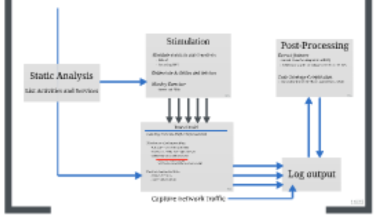
~~Trace application framework to core libraries~~
~~Trace core libraries internally~~

Extend Android Profiler to suit our needs

Implementation

TraceDroid Analysis Platform

Automated analysis



TraceDroid Analysis Platform

Inspection tool

Quickly analyze >100K lines of trace output

Load trace output into Python objects
- Interactive shell
- Call graphs for control flow analysis

TraceDroid: A Fast and Complete Android Method Tracer

Mobile Malware

+614%

Android: 92%

How do we automate analysis?

Contributions

TraceDroid

Android OS with comprehensive method tracing capabilities:
- Parameter resolution
- Return values
- Object representation

TraceDroid Analysis Platform

Framework for automated dynamic analysis

Detect suspicious activity

Ease post analysis

Evaluation

TraceDroid is fast

Benchmark: browse to 8 cached webpages

Visit each page 10 times before computing average load time

Simulation Effectiveness

Simulation Effectiveness

Code coverage of 33% is fairly low

Simulation efforts vary per app

Conclusions

Conclusions

TraceDroid
- Fast and comprehensive Android method tracer
- May be used as a debugging tool by developers

TraceDroid Analysis Platform
- Automated analysis of unknown applications
- Quickly identify suspicious applications
- Interactive environment to ease post-analysis

Future work

TraceDroid

Unpack arrays

Run on real hardware

TraceDroid Analysis Platform

Compute code coverage per package

Improve stimulation
- Symbolic execution?

http://tracedroid.few.vu.nl/

Submit your .apk for automated analysis

.tar.gz output containing:
- method traces
- network dump
- call graph

Contact me if you would like to analyze a batch

No source or inspect tool available yet

Demo

ZitMo: Zeus in the Mobile
- Collaborates with PC-based Zeus
- Steals mobile TAN codes

TraceDroid: A Fast and Complete Android Method Tracer

Hack in the Box 2013, Kuala Lumpur

Victor van der Veen

About me

Currently

- 2.04 m
- Security Consultant at ITQ
- ITQ CTF team member

Past

- HITB CTF {KUL|AMS} 2010, 2011, 2012, 2013
- Memory Errors: The Past, The Present and the Future (RAID 2012)
- (partial) Implementation of a trustworthy voting machine
- Worked on Andrubis with the iSecLab team in Vienna

About me

Currently

- 2.04 m
- Security Consultant at ITQ
- ITQ CTF team member

Past

- HITB CTF {KUL|AMS} 2010, 2011, 2012, 2013
- Memory Errors: The Past, The Present and the Future (RAID 2012)
- (partial) Implementation of a trustworthy voting machine
- Worked on Andrubis with the iSecLab team in Vienna

Mobile Malware

March 2012	38,689 samples
March 2013	276,259 samples

+614%

Android: 92%

How do we automate analysis?

Contributions

TraceDroid

Android OS with comprehensive method tracing capabilities:

- Parameter resolution
- Return values
- Object representation

TraceDroid Analysis Platform

Framework for automated dynamic analysis

Detect suspicious activity

Ease post analysis

Introduction

Application Fundamentals

Apps are written in Java, executed by a VM

Building blocks:

• *Activity*	Single screen with a UI
• *Service*	Background components
• *Receiver*	Listener for specific announcements *e.g., boot completed, sms received*

Distributed as signed jar files (.apk)

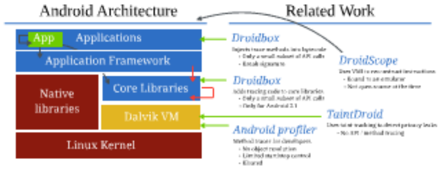
Scope

Limit method tracing to Java code
- Interesting features only accessible via Java
- Existing tools for tracing native code
  - *strace*
  - *ltrace*

Use dynamic analysis
- Evade obfuscation
- Existing tools for static analysis
  - *AndroGuard*
  - *Dex2Jar*

Android Architecture

Related Work


App
Applications
Application Framework
Native libraries
Core Libraries
Dalvik VM
Linux Kernel
Droidbox
Injects trace methods into bytecode
• Only a small subset of API calls
• Break signature
DroidScope
Uses VMI to reconstruct instructions
• Bound to an emulator
• Not open source at the time
Droidbox
Adds tracing code to core libraries
• Only a small subset of API calls
• Only for Android 2.1
TaintDroid
Uses taint tracking to detect privacy leaks
• No API / method tracing
Android profiler
Method tracer for developers
• No object resolution
• Limited start/stop control
• Bloated

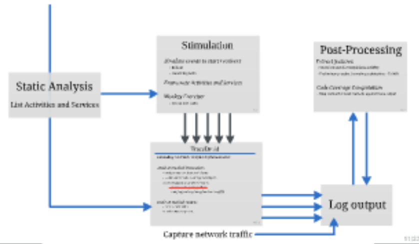
Android Profiler
- Trace app internally
- Trace app to application framework
- Trace app to core libraries
- ~~Trace application framework to core libraries~~
- ~~Trace core libraries internally~~

Extend Android Profiler to suit our needs

Implementation

TraceDroid Analysis Platform

Automated analysis



TraceDroid Analysis Platform

Inspection tool

Quickly analyze >100K lines of trace output

Load trace output into Python objects
- Interactive shell
- Call graphs for control flow analysis

TraceDroid: A Fast and Complete Android Method Tracer

Mobile Malware

March 2012	[illegible] samples
March 2013	176,259 samples

+614%

Android: 92%

How do we automate analysis?

Contributions

TraceDroid

Android OS with comprehensive method tracing capabilities:
- Parameter resolution
- Return values
- Object representation

TraceDroid Analysis Platform

Framework for automated dynamic analysis

Detect suspicious activity

Ease post analysis

Evaluation

TraceDroid is fast

Benchmark: browse to 8 cached webpages

Visit each page 16 times before computing average load time

Speedup of [illegible] compared to original profiler

Simulation Effectiveness

Compare automated analysis against manual input (180 seconds)

	Manual		TraceDroid
- [illegible] benign	[illegible]	-2.45%	[illegible]
- [illegible] malicious	[illegible]	+3.79%	[illegible]

TraceDroid's coverage is about as good as manual analysis

Likely of higher quality due to receiver stimulation

Analysis of ~[illegible] samples	Code Coverage	
- [illegible] benign	[illegible]	~33%
- [illegible] malicious	[illegible]	

Simulation Effectiveness

Code coverage of 33% is fairly low
- (third-party) Libraries
- Unreachable code
- Complex applications

Simulation effects vary per app

Monkeys tech is growing

Conclusions

Conclusions

TraceDroid
- Fast and comprehensive Android method tracer
- May be used as a debugging tool by developers

TraceDroid Analysis Platform
- Automated analysis of unknown applications
- Quickly identify suspicious applications
- Interactive environment to ease post-analysis

Future work

TraceDroid

Unpack arrays
- Obfuscation could pass parameters in Object[]

Run on real hardware
- Evade emulator detection

TraceDroid Analysis Platform

Compute code coverage per package

Improve stimulation
- *Symbolic execution?*

http://tracedroid.few.vu.nl/

Submit your .apk for automated analysis

.tar.gz output containing:
- method traces
- network dump
- call graph

Contact me if you would like to analyze a batch

No source or inspect tool available yet

Demo

ZitMo: Zeus in the Mobile
- Collaborates with PC-based Zeus
- Steals mobile TAN codes

Scope

Limit method tracing to Java code

- Interesting features only accessible via Java
- Existing tools for tracing native code

    strace

    ltrace

Use dynamic analysis

- Evade obfuscation
- Existing tools for static analysis

    AndroGuard

    Dex2Jar

Application Fundamentals

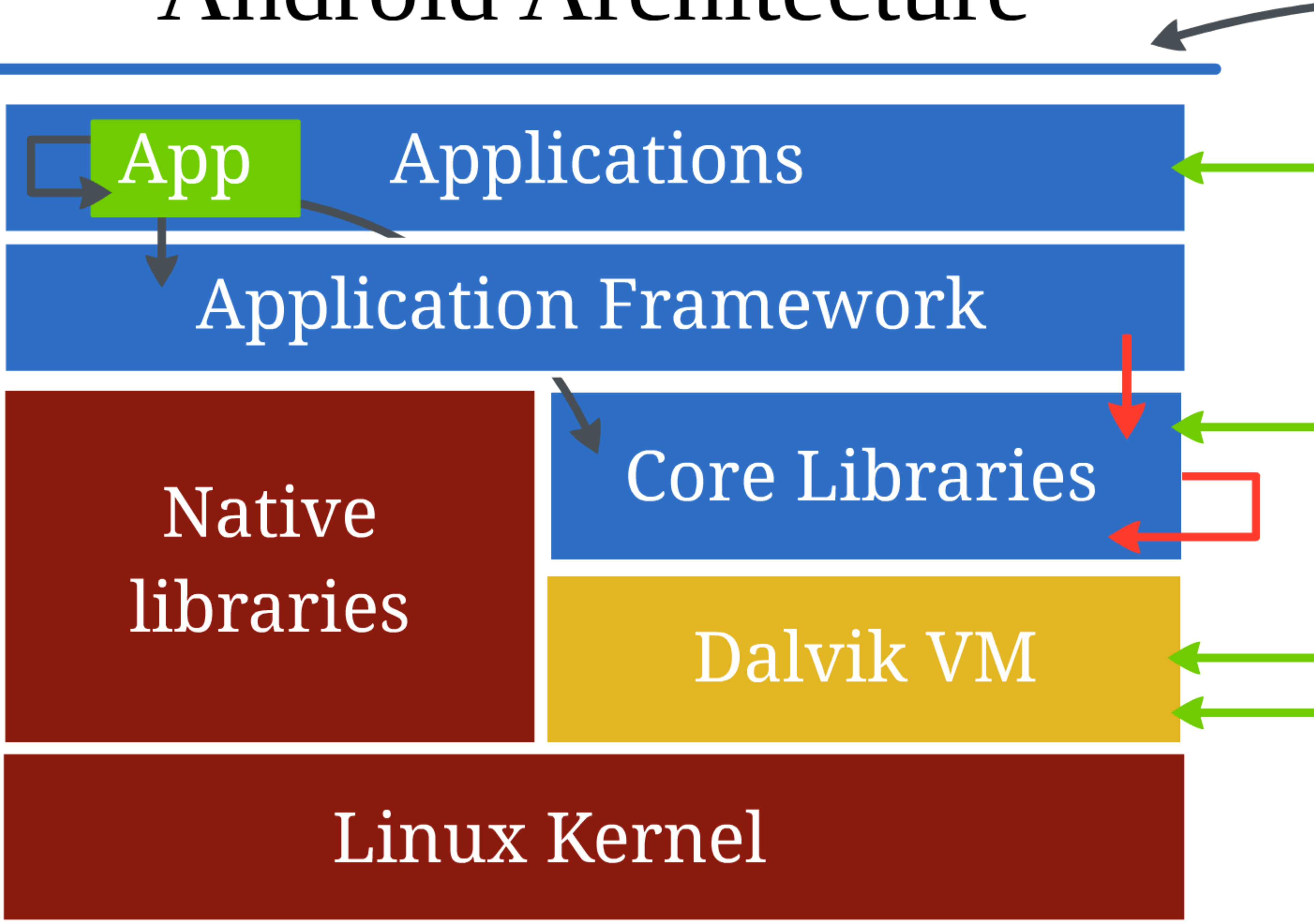
Apps are written in Java, executed by a VM

Building blocks:

• *Activity*	Single screen with a UI
• *Service*	Background components
• *Receiver*	Listener for specific announcements *e.g., boot completed, sms received*

Distributed as signed jar files (.apk)

Android Architecture



Related Work

Droidbox

Injects trace methods into bytecode

- Only a small subset of API calls
- Break signature

DroidScope

Uses VMI to reconstruct instructions

- Bound to an emulator
- Not open source at the time

Droidbox

Adds tracing code to core libraries

- Only a small subset of API calls
- Only for Android 2.1

TaintDroid

Uses taint tracking to detect privacy leaks

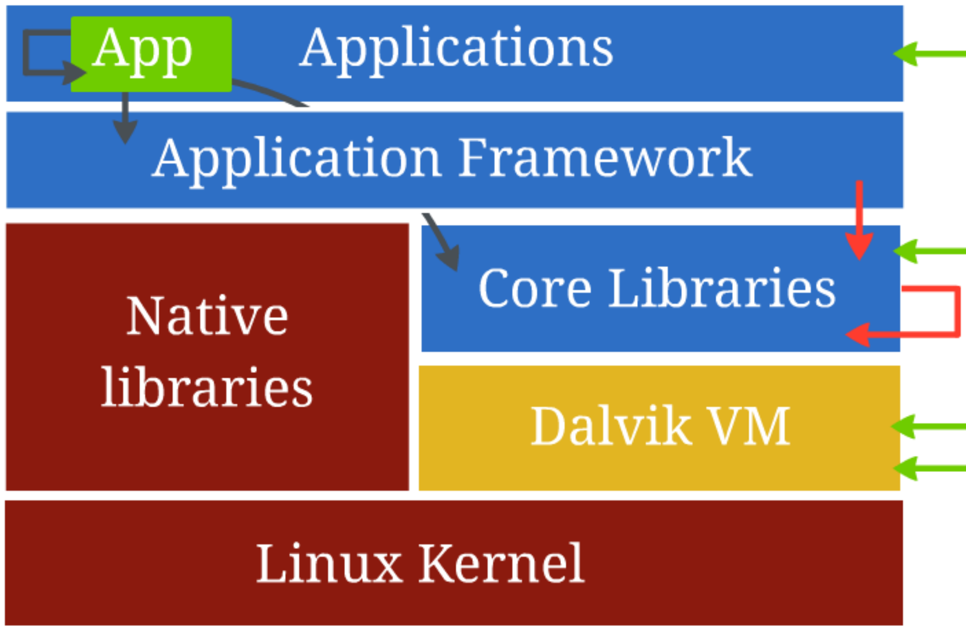
- No API / method tracing

Android profiler

Method tracer for developers

- No object resolution
- Limited start/stop control
- Bloated

Android Architecture



Related Work

Droidbox

Injects trace methods into bytecode
- Only a small subset of API calls
- Break signature

Droidbox

Adds tracing code to core libraries
- Only a small subset of API calls
- Only for Android 2.1

DroidScope

Uses VMI to reconstruct instructions
- Bound to an emulator
- Not open source at the time

TaintDroid

Uses taint tracking to detect privacy leaks
- No API / method tracing

Android profiler

Method tracer for developers
- No object resolution
- Limited start/stop control
- Bloated

Android Profiler

- Trace app internally
- Trace app to application framework
- Trace app to core libraries

- ~~Trace application framework to core libraries~~
- ~~Trace core libraries internally~~

Extend Android Profiler to suit our needs

TraceDroid

Extending Android's Profiler implementation

Hook on method invocations

- Fetch parameters from stack frame
- Lookup and invoke .tostring() for Objects
- Convert signatures and descriptors

  ~~*foo(Ljava/lang/String;Z[[J)V*~~

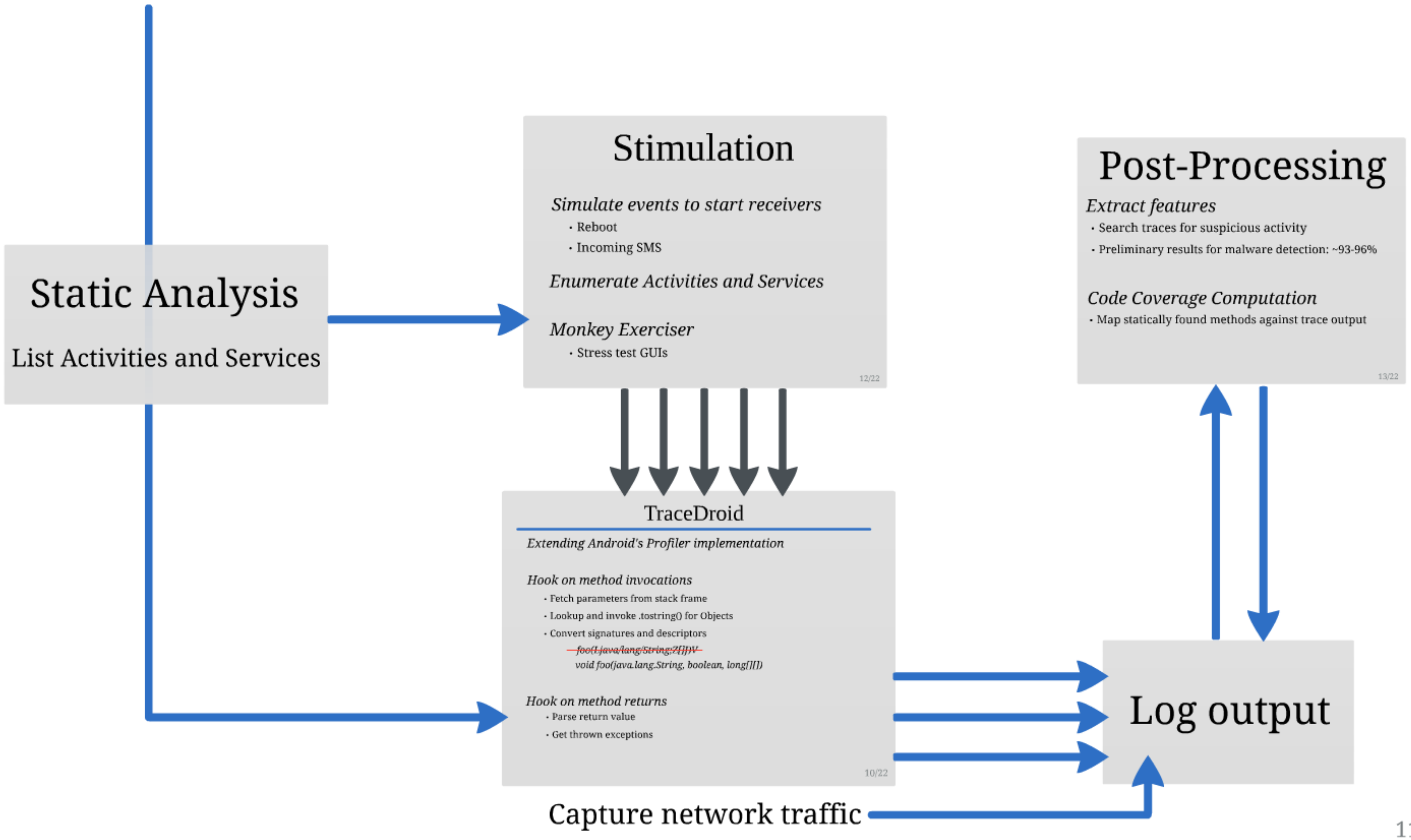
  void foo(java.lang.String, boolean, long[][])

Hook on method returns

- Parse return value
- Get thrown exceptions

TraceDroid Analysis Platform

Automated analysis



Static Analysis

List Activities and Services

Stimulation

Simulate events to start receivers
- Reboot
- Incoming SMS

Enumerate Activities and Services

Monkey Exerciser
- Stress test GUIs

12/22

Post-Processing

Extract features
- Search traces for suspicious activity
- Preliminary results for malware detection: ~93-96%

Code Coverage Computation
- Map statically found methods against trace output

13/22

TraceDroid

Extending Android's Profiler implementation

Hook on method invocations
- Fetch parameters from stack frame
- Lookup and invoke .tostring() for Objects
- Convert signatures and descriptors

  ~~*foo(Ljava/lang/String;Z[[J)V*~~

  void foo(java.lang.String, boolean, long[][])

Hook on method returns
- Parse return value
- Get thrown exceptions

10/22

Log output

Capture network traffic

Stimulation

Simulate events to start receivers

- Reboot
- Incoming SMS

Enumerate Activities and Services

Monkey Exerciser

- Stress test GUIs

Post-Processing

Extract features

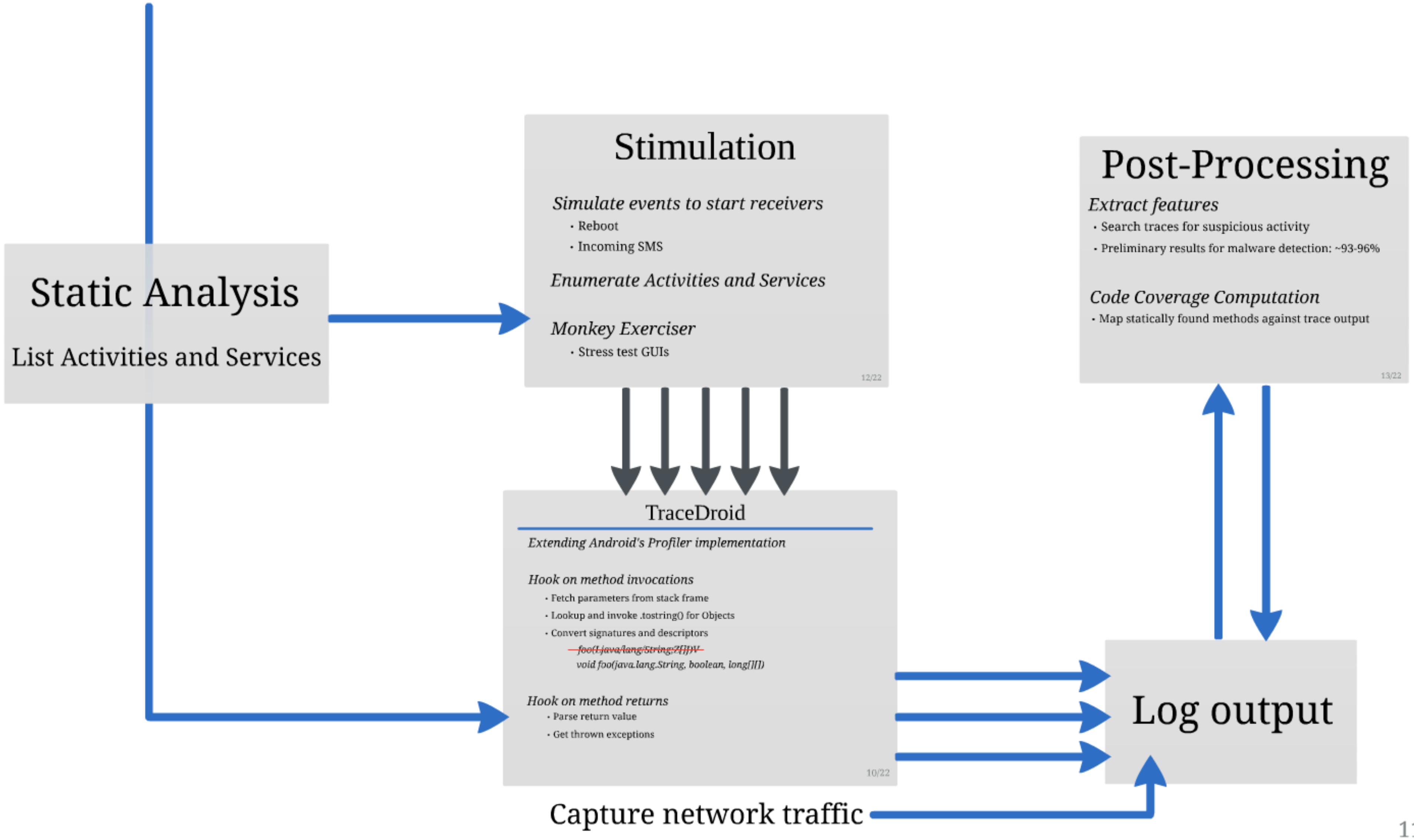
- Search traces for suspicious activity
- Preliminary results for malware detection: ~93-96%

Code Coverage Computation

- Map statically found methods against trace output

TraceDroid Analysis Platform

Automated analysis



Static Analysis

List Activities and Services

Stimulation

Simulate events to start receivers

- Reboot
- Incoming SMS

Enumerate Activities and Services

Monkey Exerciser

- Stress test GUIs

12/22

Post-Processing

Extract features

- Search traces for suspicious activity
- Preliminary results for malware detection: ~93-96%

Code Coverage Computation

- Map statically found methods against trace output

13/22

TraceDroid

Extending Android's Profiler implementation

Hook on method invocations

- Fetch parameters from stack frame
- Lookup and invoke .tostring() for Objects
- Convert signatures and descriptors

  ~~*foo(Ljava/lang/String;Z[J])V*~~

  void foo(java.lang.String, boolean, long[][])

Hook on method returns

- Parse return value
- Get thrown exceptions

10/22

Log output

Capture network traffic

TraceDroid Analysis Platform

Inspection tool

Quickly analyze >100K lines of trace output

Load trace output into Python objects

- Interactive shell
- Call graphs for control flow analysis

TraceDroid is fast

Benchmark: browse to 8 cached webpages

Visit each page 10 times before computing average load time

Speedup of 1.45 compared to original profiler

Simulation Effectiveness

Compare automated analysis against manual input (180 seconds)

	Manual		*TraceDroid*
• 17x benign	38.49%	−2.45%	36.04%
• 18x malicious	27.61%	+3.79%	31.40%

TraceDroid's coverage is about as good as manual analysis

Likely of higher quality due to receiver stimulation

Analysis of ~500 samples	*Code Coverage*	
• 250x benign	35.02%	~33%
• 242x malicious	31.10%	

Simulation Effectiveness

Code coverage of 33% is fairly low

- (third-party) Libraries
- Unreachable code
- Complex applications

Simulation effects vary per app

Simulation effects vary per app

Monkeys suck at gaming

Demo

ZitMo: Zeus in the Mobile

- Collaborates with PC-based Zeus
- Steals mobile TAN codes

Future work

TraceDroid

Unpack arrays

- Obfuscation could pass parameters in Object[]

Run on real hardware

- Evade emulator detection

TraceDroid Analysis Platform

Compute code coverage per package

Improve stimulation

- Symbolic execution?

Conclusions

TraceDroid

- Fast and comprehensive Android method tracer
- May be used as a debugging tool by developers

TraceDroid Analysis Platform

- Automated analysis of unknown applications
- Quickly identify suspicious applications
- Interactive environment to ease post-analysis

http://tracedroid.few.vu.nl/

Submit your .apk for automated analysis

.tar.gz output containing:

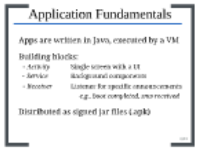
- method traces
- network dump
- call graph

Contact me if you would like to analyze a batch

No source or inspect tool available yet

http://tracedroid.few.vu.nl

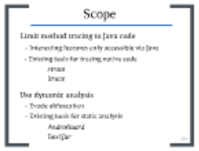
Introduction


Application Fundamentals

Apps are written in Java, executed by a VM

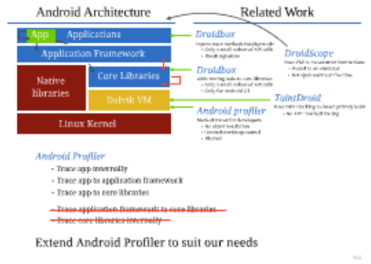
Building blocks:
- *Activity* Single screen with a UI
- *Service* Background components
- *Receiver* Listener for specific announcements
  e.g., boot completed, sms received

Distributed as signed jar files (.apk)


Scope

Limit method tracing to Java code
- Interesting features only accessible via Java
- Existing tools for tracing native code
  strace
  ltrace

The dynamic analysis
- Evade obfuscation
- Existing tools for static analysis
  Androguard
  Dexter


Android Architecture

Related Work

Android Profiler
- Trace app internally
- Trace app to application framework
- Trace app to core libraries

~~- Trace application framework to core libraries~~
~~- Trace core libraries internally~~

Extend Android Profiler to suit our needs

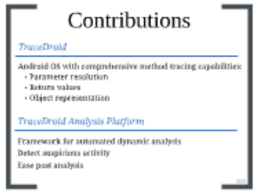
TraceDroid: A Fast and Complete Android Method Tracer

Mobile Malware

+614%

Android: 92%

How do we automate analysis?


Contributions

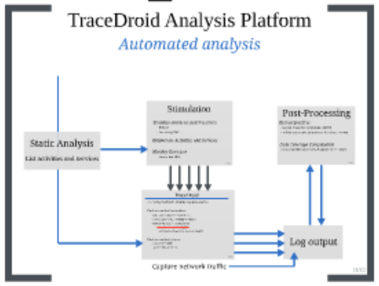
TraceDroid

Android OS with comprehensive method tracing capabilities
- Parameter resolution
- Return values
- Object representation

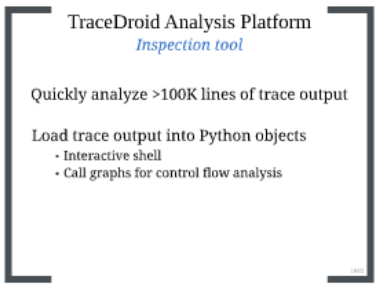
TraceDroid Analysis Platform

Framework for automated dynamic analysis
Detect suspicious activity
Ease post analysis

Implementation


TraceDroid Analysis Platform

Automated analysis


TraceDroid Analysis Platform

Inspection tool

Quickly analyze >100K lines of trace output

Load trace output into Python objects
- Interactive shell
- Call graphs for control flow analysis

Conclusions

Conclusions

TraceDroid
- Fast and comprehensive Android method tracer
- May be used as a debugging tool by developers

TraceDroid Analysis Platform
- Automated analysis of unknown applications
- Quickly identify suspicious applications
- Interactive environment to ease post-analysis

Future work

TraceDroid

Unpack arrays
- Obfuscation could put parameters in objects

Run on real hardware
- Evade emulator detection

TraceDroid Analysis Platform

Compute code coverage per package

Improve stimulation
- Symbolic execution?

http://tracedroid.few.vu.nl/

Submit your .apk for automated analysis
.tar.gz output containing:
- method traces
- network dump
- call graph

Contact me if you would like to analyze a batch

No source or inspect tool available yet

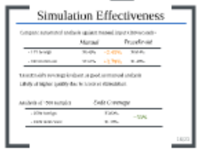
Demo

ZitMo: Zeus in the Mobile
- Collaborates with PC-based Zeus
- Steals mobile TAN codes

Evaluation

TraceDroid is fast

Benchmark: browse to 8 cached webpages


Simulation Effectiveness

Manual / TraceDroid

Code Coverage

Simulation Effectiveness

Code coverage of 33% is fairly low

Simulation effects vary per app