Implementation

TraceDroid Analysis Platform
Automated analysis

Quickly analyze >100K lines of trace output
Load trace output into Python objects
- Interactive shell
- Call graphs for control flow analysis

Conclusions

TraceDroid
- Fast and comprehensive Android instrumentation
- Automated analysis of software applications
- Easily extendable and adaptable
- Integration with existing toolsets

Evaluation

Demo

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

http://tracedroid.few.vu.nl/
Submit your apk for automated analysis
Analyze 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
TraceDroid: A Fast and Complete Android Method Tracer

OWASP BeNeLux Day 2013, Amsterdam

Victor van der Veen

About me
Security Consultant at ITQ

Past
• MSc. in Computer Science, VU University
• Capture the Flag 'hacking' competitions
• 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

Twitter: @vvdveen
E-Mail: vvanderveen@itq.nl
About me

Security Consultant at ITQ

Past

• MSc. in Computer Science, VU University
• Capture the Flag 'hacking' competitions
• 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

Twitter: @vvdveen
E-Mail: vvanderveen@itq.nl
Mobile Malware

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

+614%

Android: 92%

How do we automate analysis?
Contributions

Idea

Trace apps in an emulated environment
Monitor behavior

TraceDroid

Modified Android OS for method tracing
Framework for automated dynamic analysis
Detect suspicious activity
Ease post analysis
Introduction

Scope
- Android application instrumentation
- Method traces from several Android applications
- Dynamic analysis of Android applications
- Dynamic analysis of Android traces
- Static analysis of Android traces
- Call graphs of control flow analysis

Android Architecture
- Key Points
  - Applications
  - Related Work
  - Android Profiler
- Contributions
  - TraceDroid
  - Combined approach to tool design

Android Profiler
- TraceDroid framework
- Android application instrumentation
- Method traces from several Android applications
- Dynamic analysis of Android application traces
- Static analysis of Android application traces
- Call graphs of control flow analysis

Conclusions

Evaluation

Implementation

TraceDroid Analysis Platform
- Automated analysis
- Load trace output into Python objects
- Interactive shell
- Call graphs for control flow analysis

Contributions

TraceDroid is fast
- New and improved tool
- Improved tool for Android analysis
- Increased performance

zitmo is fast
- New and improved tool
- Improved tool for Android analysis
- Increased performance

Demo

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

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

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

Contact me if you would like to analyze a batch

No source or inspect tool available yet
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)
Related Work

**Droidbox**
Injests 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

**Android profiler**
Method tracer for developers
- No object resolution
- Limited start/stop control
- Bloated
Android Architecture

- App
- Applications
- Application Framework
- Native libraries
- Core Libraries
- Dalvik VM
- Linux Kernel

Related Work

- **Droidbox**
  - Injests 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

- **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

---

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

```java
public static final Method HOOK_METHOD
    = new MethodParser()
    .parseSignature(new MethodSignature("foo(Ljava/lang/String;Z[];)V"))
    .method();
```

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

Post-Processing
- Extract features
  - Search traces for suspicious activity
- Preliminary results for malware detection: ~99.96%
- Code Coverage Computation
  - Map statically found methods against trace output

TraceDroid
- Extending Android's Profiler implementation
- Hook on method invocations
  - Track parameters from stack frames
  - Lookup and invoke (invoking) for objects
  - Convert signatures and descriptors
    - `foo(int bar, long arg) throws Exception`
- Hook on method returns
  - Insert return value
  - Get thrown exceptions

Capture network traffic
Log output
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

Post-Processing
Extract features
- Search traces for suspicious activity
- Preliminary results for malware detection: ~99.96%

Code Coverage Computation
- Map statically found methods against trace output

TraceDroid
Extending Android's Profiler implementation

Hook on method invocations
- Fetch parameters from stack frame
- lookup and invoke (return) for objects
- Convert signatures and descriptors

Hook on method returns
- Intercept return value
- Intercept thrown exceptions

Capture network traffic
Log output
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)

<table>
<thead>
<tr>
<th></th>
<th>Manual</th>
<th>TraceDroid</th>
</tr>
</thead>
<tbody>
<tr>
<td>17x benign</td>
<td>38.49%</td>
<td>-2.45%</td>
</tr>
<tr>
<td>18x malicious</td>
<td>27.61%</td>
<td>+3.79%</td>
</tr>
</tbody>
</table>

TraceDroid's coverage is about as good as manual analysis

Likely of higher quality due to receiver stimulation

Analysis of ~500 samples

<table>
<thead>
<tr>
<th></th>
<th>Code Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>250x benign</td>
<td>35.02%</td>
</tr>
<tr>
<td>242x malicious</td>
<td>31.10%</td>
</tr>
</tbody>
</table>
Simulation Effectiveness

Code coverage of 33% is fairly low

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

Simulation effects vary per app

![Breakdown of Simulation Techniques]

Monkeys suck at gaming
Simulation effects vary per app

Monkeys suck at gaming
Demo

ZitMo: Zeus in the Mobile

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

**TraceDroid**

- Fast and comprehensive Android method tracer
- Automated analysis of unknown applications
- Quickly identify suspicious applications
- Interactive environment to ease post-analysis
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
Implementation

Quickly analyze >100K lines of trace output

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

Conclusions

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

Evaluation

Demo

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

Submit your .apk for automated analysis
.tar.gz output containing:
- method trace
call graph
Contact me if you would like to analyze a batch

No source or inspect tool available yet