Northeastern University

Systems Security Lab

Messing with the Android Runtime

SyScan Singapore 2013

Collin Mulliner, April 26th 2013, Singapore
crm[at]ccs.neu.edu

$ finger collin@mulliner.org

- 'postdoc' Security Researcher
  - $HOME = Northeastern University, Boston, MA, USA
  - `cat .project`

    specialized in *mobile handset security*

- Current work
  - Android security
  - Android security

- Past work
  - Some Bluetooth security work
  - A lot on SMS and MMS security
  - Mobile web usage and privacy
  - Some early work on NFC phone security

Introduction

- Android Application Security
  - Find vulnerabilities (audit)
  - Analyze malware
  - RE … what is this application doing

- What does this thing do? How does this thing work?
  - Disassemble → look at smali code
  - Run in emulator/sandbox → look at traces / network
  - (Static) instrumentation → look at app while it runs

Introduction

- Android Application Security
  - Find vulnerabilities (audit)
  - Analyze malware
  - RE … what is this application doing

- What does this thing do? How does this thing work?
  - Disassemble → look at smali code
  - Run in emulator/sandbox → look at traces / network
  - (Static) instrumentation → look at app while it runs

- **This talk is about Dynamic Instrumentation**
  - **Instrumentation at the Dalvik level (not bytecode!)**

Static Instrumentation on Android

- Unpack APK
  - Convert manifest back to plain text, ...
- Disassemble DEX classes
  - Get smali code
- Instrument smali code
  - Modify smali code, add own code
- Repackage application
  - Compile code, Sign, etc...
- Install and run
  - Hope it works... (bug in patch, self integrity check, ...)

Dynamic Instrumentation

- Change/modify application code at runtime
  - Allows to add and remove code/hooks on-the-fly
  - Technique has been around for many years

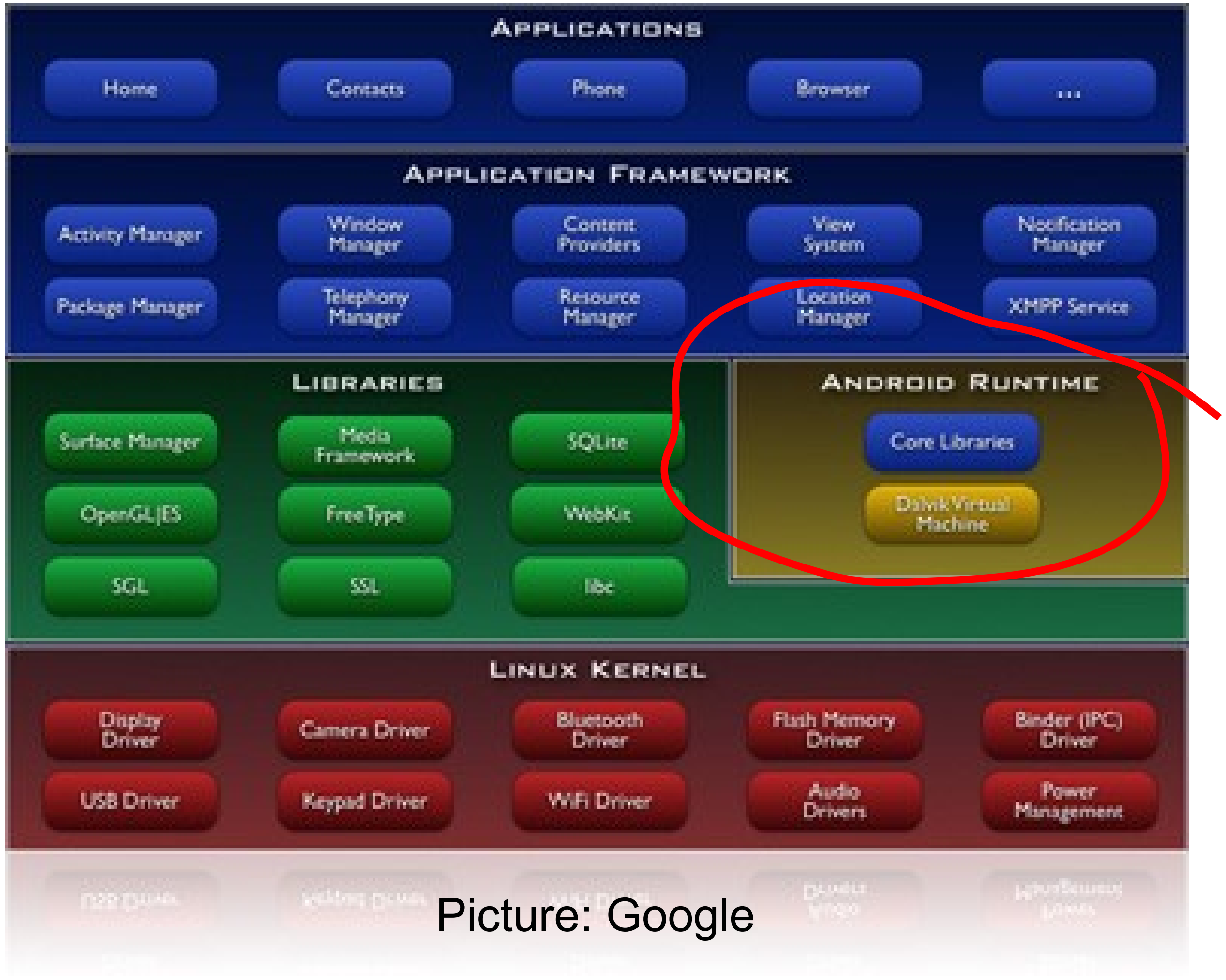
- Instrument library calls: quick overview what happens
  - No disassembly needed

- Still need to disassemble for target specific stuff
  - Find the interesting stuff to instrument

Dynamic Instrumentation on Android

- No: unpacking, compile, repacking
  - Saves us time
- APK not modified
  - Defeat 'simple' integrity checks
- But Android Apps are written in Java and run in a VM...

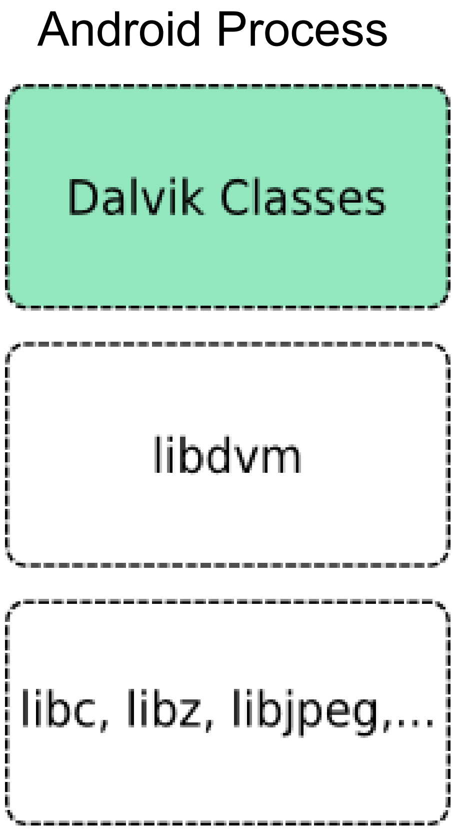
Android



Picture: Google

Android Runtime

- Dalvik Virtual Machine (DVM)
  Core Libraries (java.x.y)
  - Executes: Framework and Applications
- Application
  - Process for “MainActivity”
  - Additional process(s) for “Service”
- Framework
  - system_server
  - ...



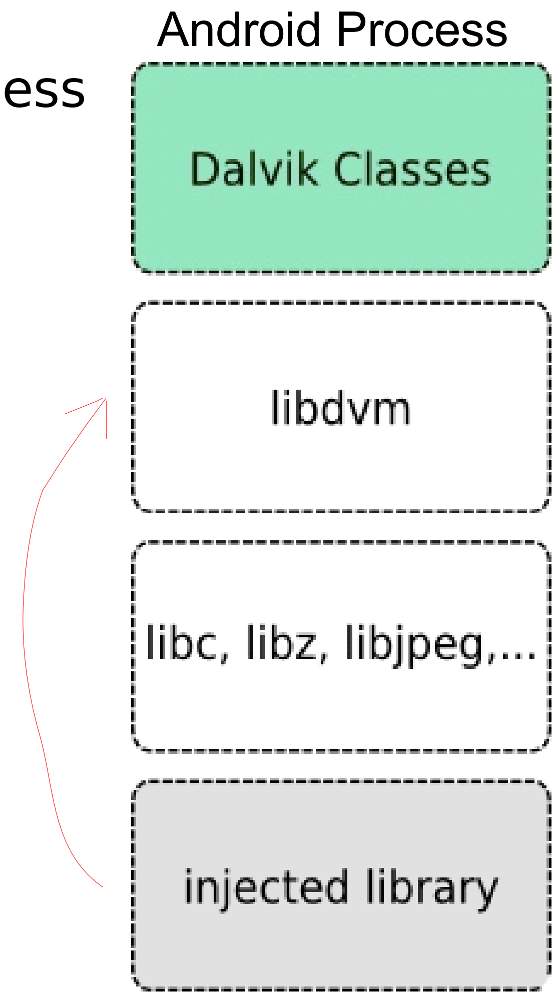
Dalvik Instrumentation – The Basic Idea

- Convert Dalvik method to native (JNI) method
  - We get control of the execution
- Call original Dalvik method from native method
  - This creates an in-line hook of the Dalvik method
- Implement instrumentation code using JNI
  - Access to everything (private, protected doesn't exist)

Dalvik Instrumentation – Tech Overview

- Inject 'shared object' (.so) into running process
  - Provides the native code
  - My talk: *Dynamic Binary Instrumentation on Android*

- Native code 'talks to the DVM'
  - Resolve symbols from DVM
  - Call DVM functions to:
    - Lookup classes and methods
    - Hook method
    - Call original method



Messing with the Android Runtime

- The Runtime “runs”
  - Applications and their Services
  - The Android System/Framework
- What can we do with this
  - Aid reverse engineering
  - Attacks
  - Test stuff fast
- **Examples…**

Monitor / Reverse Applications

- How does the application work?
  - Maybe App is obfuscated, strings are “encrypted”
- Instrument interesting methods to see what App does
  - String operations
  - Reflection
  - …

```
String   java.lang.StringBuffer.toString()
int      java.lang.String.compareTo(..)
int      java.lang.String.compareToIgnoreCase(..)
String   java.lang.StringBuilder.toString()


Method   java.lang.Class.getMethod(..)
```

Attack “stuff”

- Two Apps talk to each other via some IPC
  - Instrument one side to attack the other side

- Disable Signature Verification

```
boolean java.security.Signature.verify(byte[]) { … }
```

  - Used for all kinds of things…
  - Patch to always “return true;”
    (used it to attack various things)

Rapid Prototyping of Framework Modifications

- Defense against SMS OTP stealing Trojans [1]
  - Change local SMS routing based on SMS content

- For the prototype we needed to change code in the framework

```
com/android/internal/telephony/SMSDispatcher.java
protected void dispatchPdus(byte[] pdus) { … }
```

- Instead of recompiling Android just replace the method
  → save a lot of time
  → test on many different devices without custom compile

[1] *SMS-based One-Time Passwords: Attacks and Defense (short paper)* Collin Mulliner, Ravishankar Borgaonkar, Patrick Stewin, Jean-Pierre Seifert To appear In the Proceedings of the 10th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA 2013) Berlin, Germany, July 2013

Conclusions

- Dynamic Instrumentation via the Android Runtime allows
  - Modification of Apps and the Framework in memory
  - Doesn't break APK signatures
  - Portable across devices
  - Super stable (not a hack)
  - But can only replace whole functions
    - no bytecode modification
- Possible to stir up Android AppSec quite a bit
  - Obfuscation and use of reflection is kinda useless
- We have various ongoing projects based on this
  - Students doing interesting stuff

EOF

Thank you!

twitter: @collinrm
crm[at]ccs.neu.edu
http://mulliner.org/android