DroidScope:
Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis

Lok Yan
Heng Yin
August 10, 2012
Android

System Services

Java Components

Apps

Native Components
Android

Java Components

Native Components

System Services

Apps
Motivation: Static Analysis

Dalvik/Java Static Analysis:
ded, Dexpler, soot, Woodpecker, DroidMoss

Native Static Analysis:
IDA, binutils, BAP

Linux Kernel

Zygote
System Services
Java Component
Java Component
Java Libraries
Dalvik VM
Native Component
System Libraries
Motivation: Dynamic Analysis

Android Analysis: TaintDroid, DroidRanger

System Calls

Zygote
System Services
Java Component
Java Component
Java Libraries
Dalvik VM
JNI
Native Component
System Libraries
Linux Kernel

logcat, adb
Motivation: Dynamic Analysis

External Analysis: Anubis, Ether, TEMU, ...
DroidScope Overview

- Zygote
- System Services
- Java Component
- Java Component
- Java Libraries
- Native Component
- System Libraries
- Dalvik VM
- JNI

Integration Points:
- API Tracer
- Native Insn. Tracer
- Dalvik Insn. Tracer
- Taint Tracker

Instrumentation Interface:
- Java-level View
- OS-level View
Goals

- Dynamic binary instrumentation for Android
  - Leverage Android Emulator in SDK
  - No changes to Android Virtual Devices
  - External instrumentation
    - Linux context
    - Dalvik context
  - Extensible: plugin-support / event-based interface
- Performance
  - Partial JIT support
  - Instrumentation optimization
Roadmap

➢ External instrumentation
  – Linux context
  – Dalvik context

• Extensible: plugin-support / event-based interface

• Evaluation
  – Performance
  – Usage
Linux Context: Identify App(s)

• Shadow task list
  – pid, tid, uid, gid, euid, egid, parent pid, pgd, comm
  – argv[0]

• Shadow memory map
  – Address Space Layout Randomization (Ice Cream Sandwich)

• Update on
  – fork, execve, clone, prctl and mmap2
Java/Dalvik View

• Dalvik virtual machine
  – register machine (all on stack)
  – 256 opcodes
  – saved state, glue, pointed to by ARM R6, on stack in x86

• mterp
  – offset-addressing: fetch opcode then jump to 
    
    (dvmAsmInstructionStart + opcode * 64)
  
    – dvmAsmSisterStart for emulation overflow

• Which Dalvik opcode?
  1. Locate dvmAsmInstructionStart in shadow memory map
  2. Calculate opcode = (R15 - dvmAsmInstructionStart) / 64.
Just In Time (JIT) Compiler

• Designed to boost performance
• Triggered by counter - mterp is always the default
• Trace based
  – Multiple basic blocks
  – Multiple exits or chaining cells
  – Complicates external introspection
  – Complicates instrumentation
Disabling JIT

- Update Program Counter (PC)
- Is Code in JIT code cache?
  - Yes: Execute JIT code block
  - No: Decrement block Counter
    - Is Counter 0?
      - Yes: Request JIT Compilation for Code block and reset Counter
      - No: Emulate Code Using mterp
Roadmap

✅ External instrumentation
  – Linux context
  – Dalvik context

➢ Extensible: plugin-support / event-based interface

• Evaluation
  – Performance
  – Usage
Instrumentation Design

• Event based interface
  – Execution: e.g. native and Dalvik instructions
  – Status: updated shadow task list

• Query and Set, e.g. interpret and change cpu state

• Performance
  – Example: Native instructions vs. Dalvik instructions
  – Instrumentation Optimization
Dynamic Instrumentation

- Update PC
- inCache?
  - yes
  - Translate
  - Execute
  - no

- (un)registerCallback
  - needFlush?
    - yes
    - flushType
      - invalidateBlock(s)
      - flushCache
## Instrumentation

<table>
<thead>
<tr>
<th>Events</th>
<th>NativeAPI</th>
<th>LinuxAPI</th>
<th>DalvikAPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>instruction begin/end</td>
<td>context switch</td>
<td></td>
<td>Dalvik instruction begin method begin</td>
</tr>
<tr>
<td>register read/write</td>
<td>system call</td>
<td></td>
<td>method begin</td>
</tr>
<tr>
<td>memory read/write</td>
<td>task begin/end</td>
<td></td>
<td></td>
</tr>
<tr>
<td>block begin/end</td>
<td>task updated</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>memory map updated</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Query &amp; Set</td>
<td>memory read/write</td>
<td>query symbol database</td>
<td>query symbol database</td>
</tr>
<tr>
<td></td>
<td>memory r/w with pgd</td>
<td>get current context</td>
<td>interpret Java object</td>
</tr>
<tr>
<td></td>
<td>register read/write</td>
<td>get task list</td>
<td>get/set DVM state</td>
</tr>
<tr>
<td></td>
<td>taint set/check</td>
<td></td>
<td>taint set/check objects</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>disable JIT</td>
</tr>
</tbody>
</table>
Dalvik Instruction Tracer (Example)

1. void opcode_callback(uint32_t opcode) {
   2.   printf("[\%x] %s\n", GET_RPC, opcodeToStr(opcode));
   3. }

5. void module_callback(int pid) {
   6.   if (bInitialized || (getIBase(pid) == 0))
   7.       return;
   8.
   9.       getModAddr("dfk@classes.dex", &startAddr, &endAddr);
10.     addDisableJITRange(pid, startAddr, endAddr);
11.     disableJITInit(getGetCodeAddrAddress(pid));
12.     addMterpOpcodesRange(pid, startAddr, endAddr);
13.     dalvikMterpInit(getIBase(pid));
14.     registerDalvikInsnBeginCb(&opcode_callback);
15.     bInitialized = 1;
16. }

19. void _init() {
20.   setTargetByName("com.andhuhu.fengyinchuanshuo");
21.   registerTargetModulesUpdatedCb(&module_callback);
22. }

Plugins

• **API Tracer**
  - System calls
    - *open, close, read, write*, includes parameters and return values
  - Native library calls
  - Java API calls
    - Java Strings converted to C Strings

• **Native and Dalvik Instruction Tracers**

• **Taint Tracker**
  - Taints ARM instructions
  - One bit per byte
  - Data movement & Arithmetic instructions including barrel shifter
  - Does not support control flow tainting
Roadmap

- External instrumentation
  - Linux context
  - Dalvik context
- Extensible: plugin-support / event-based interface

- Evaluation
  - Performance
  - Usage
Implementation

• Configuration
  – QEMU 0.10.50 – part of Gingerbread SDK
  – Gingerbread
    • “user-eng”
    • No changes to source
  – Linux 2.6.29, QEMU kernel branch
Performance Evaluation

- **Seven free benchmark Apps**
  - AnTuTu Benchmark
  - (ABenchMark) by AnTuTu
  - CaffeineMark by Ravi Reddy
  - CF-Bench by Chainfire
  - Mobile processor benchmark (Multicore) by Andrei Karpushonak
  - Benchmark by Softweg
  - Linpack by GreeneComputing

- **Six tests repeated five times each**
  - Baseline
  - NO-JIT Baseline – uses a build with JIT disabled at runtime
  - Context Only
  - API Tracer
  - Dalvik Instruction Trace
  - Taint Tracker
Select Performance Results

Results are not perfect

APITracer vs. NOJIT

Dynamic Symbol Retrieval Overhead
Usage Evaluation

• Use DroidScope to analyze real world malware
  – API Tracer
  – Dalvik Instruction Tracer + dexdump
  – Taint Tracker – taint IMEI/IMSI @ move_result_object after getIMEI/getIMSI

• Analyze included exploits
  – Removed patches in Gingerbread
  – Intercept system calls
  – Native instruction tracer
Droid Kung Fu

• Three encrypted payloads
  – ratc (Rage Against The Cage)
  – killall (ratc wrapper)
  – gjsvro (udev exploit)

• Three execution methods
  – piped commands to a shell (default execution path)
  – Runtime.exec() Java API (instrumented path)
  – JNI to native library terminal emulator (instrumented path)
  – Instrumented return values for isVersion221 and getPermission methods
Droid Kung Fu: TaintTracker

getDeviceId()

String @ 0x4052e80
"123456789012345"

UrlEncodedFormEntity.<init>

String @ 0x4056a48
"imei=123456789012345&ostype="

AbstractHttpClient.execute()

byte[ ] @ 405967c0 / void* @ 405967d0
"POST /search/sayhi.php HTTP/1.1..."

sys_write(34, 0x405967d0, 397)
DroidDream

• Same payloads as DroidKungFu
• Two processes
  – Normal droiddream process clears logcat
  – droiddream:remote is malicious
• xor-encrypts private information before leaking
• Instrumented sys_connect and sys_write
DroidDream: crypt trace

[43328f40] aget-byte v2(0x01), v4(0x405232a8), v0(186)
   Getting Tainted Memory: 40523372(2401372)
   Adding M@410accec(42c5cec) len = 4
[43328f44] sget-object v3(0x0000005e), KEYVALUE// field@0003
[43328f48] aget-byte v3(0x88), v3(0x4051e288), v1(58)
[43328f4c] xor-int/2addr v2(62), v3(41)
   Getting Tainted Memory: 410accec(42c5cec)
   Adding M@410accec(42c5cec) len = 4
[43328f4e] int-to-byte v2(0x17), v2(23)
   Getting Tainted Memory: 410accec(42c5cec)
   Adding M@410accec(42c5cec) len = 4
[43328f50] aput-byte v2(0x17), v4(0x405232a8), v0(186)
   Getting Tainted Memory: 410accec(42c5cec)
   Adding M@40523372(2401372) len = 1
Summary

• DroidScope
  – Dynamic binary instrumentation for Android
  – Built on Android Emulator in SDK
  – External Introspection & Instrumentation support
  – Four plugins
    • API Tracer
    • Native Instruction Tracer
    • Dalvik Instruction Tracers
    • TaintTracker
  – Partial JIT support
Related Works

• Static Analysis
  – ded, Dexpler, soot
  – Woodpecker, DroidMoss

• Dynamic Analysis
  – TaintDroid
  – DroidRanger
  – PIN, Valgrind, DynamoRIO
  – Anubis, TEMU, Ether, PinOS

• Introspection
  – Virtuoso
  – VMWatcher
Challenges

- **JIT**
  - Full JIT support
  - Flushing JIT cache
- **Emulation detection**
  - Real Sensors: GPS, Microphone, etc.
  - Bouncer
- **Timing assumptions, timeouts, events**
- **Closed source systems, e.g. iOS**
Questions?

Q0. Where can I get DroidScope?