DroidScope:

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

Lok Yan

Heng Yin

August 10, 2012

L.C.Smith College of Engineering and Computer Science

Android

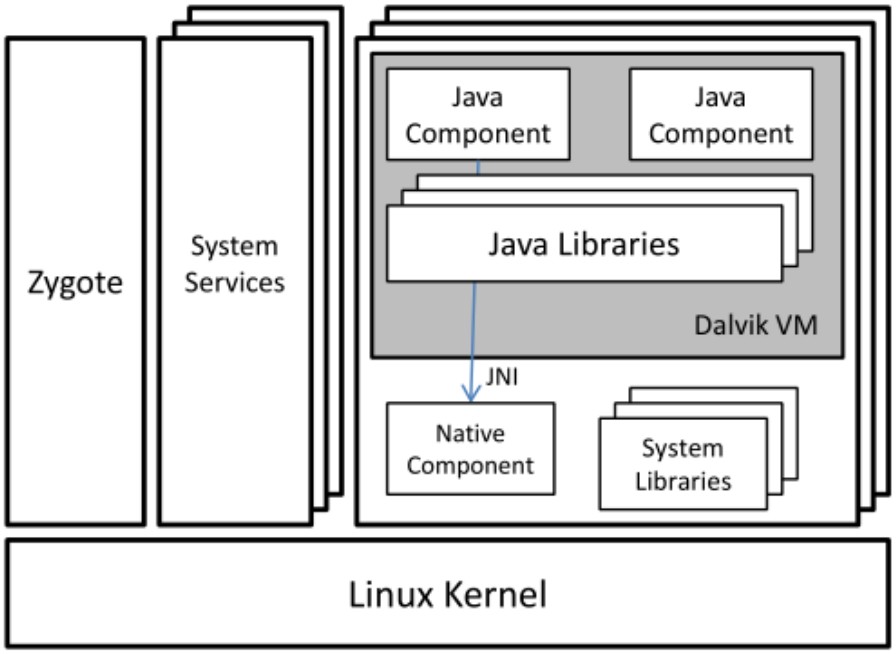
Java Components

System Services

Native Components

Apps

Android



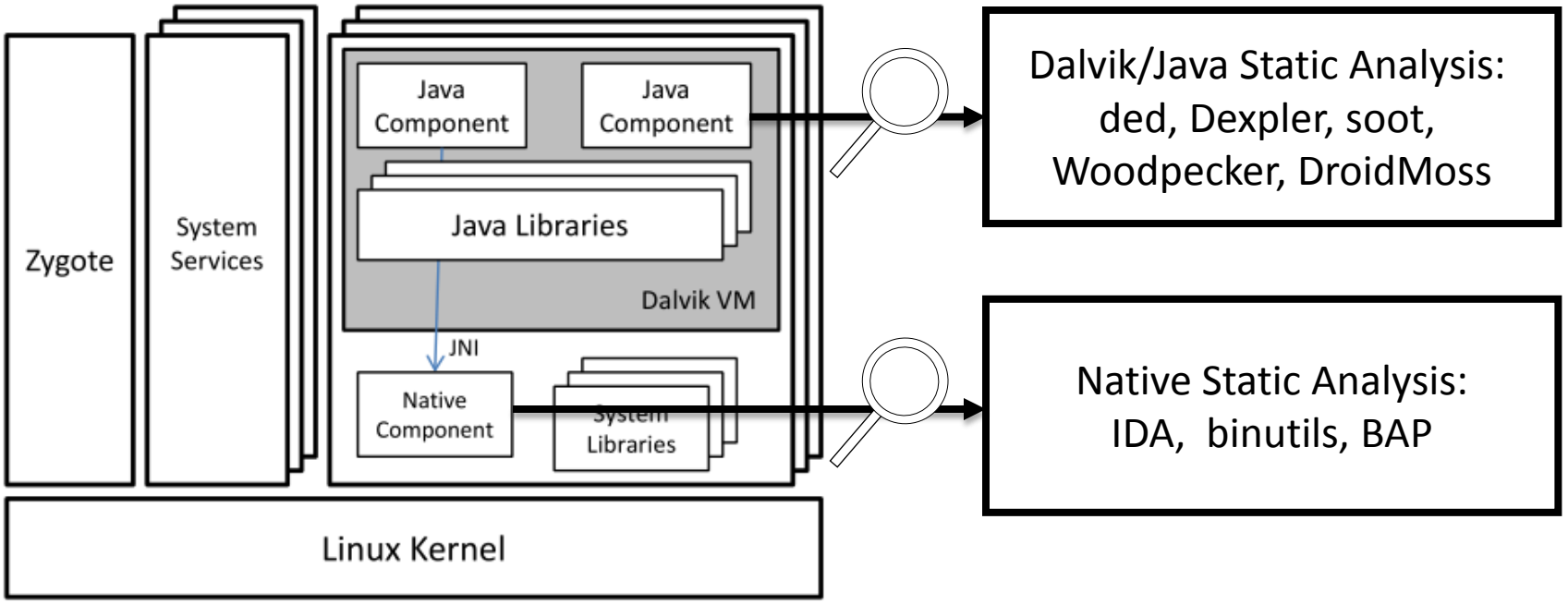
Java Components

Native Components

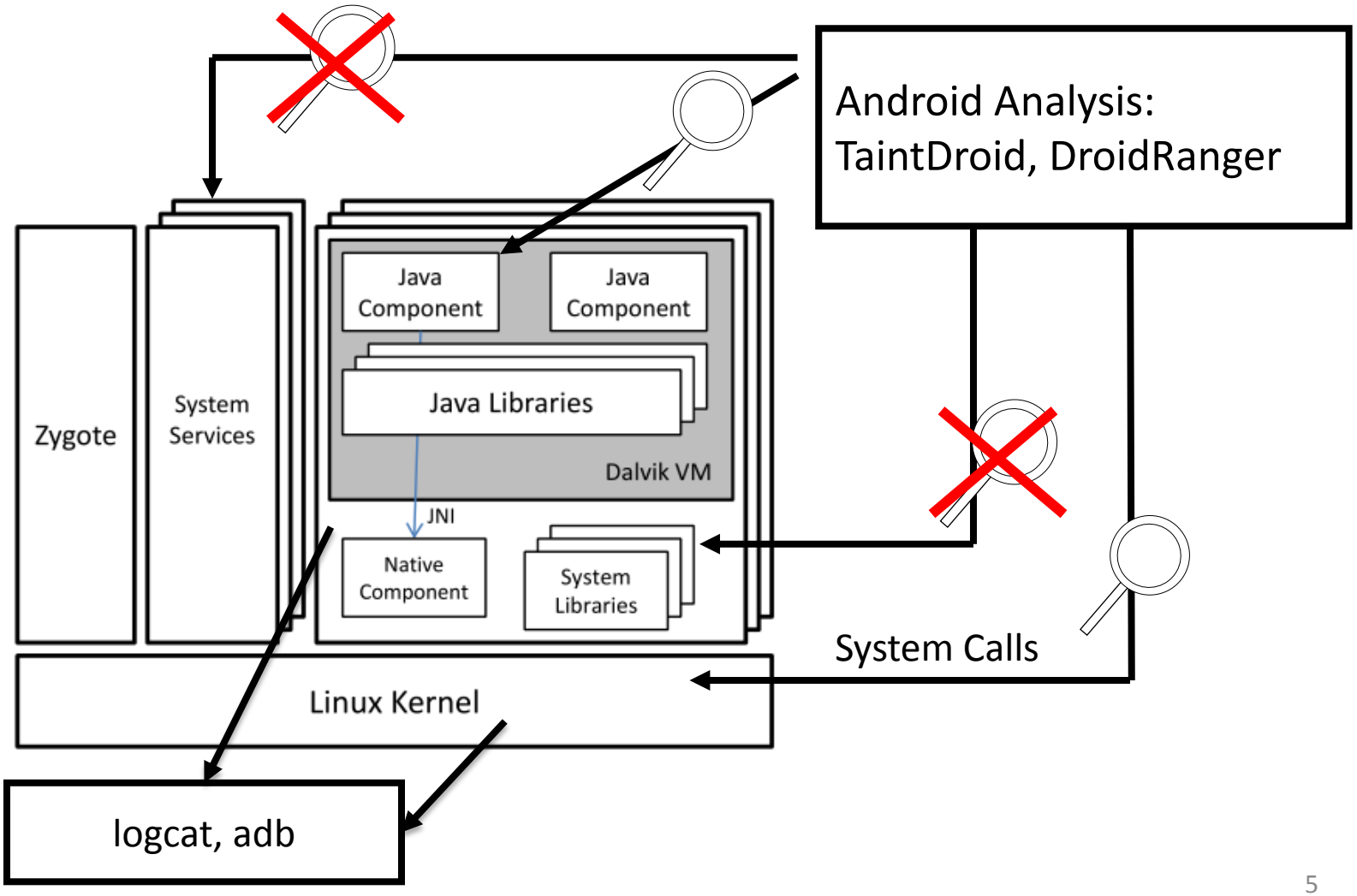
System Services

Apps

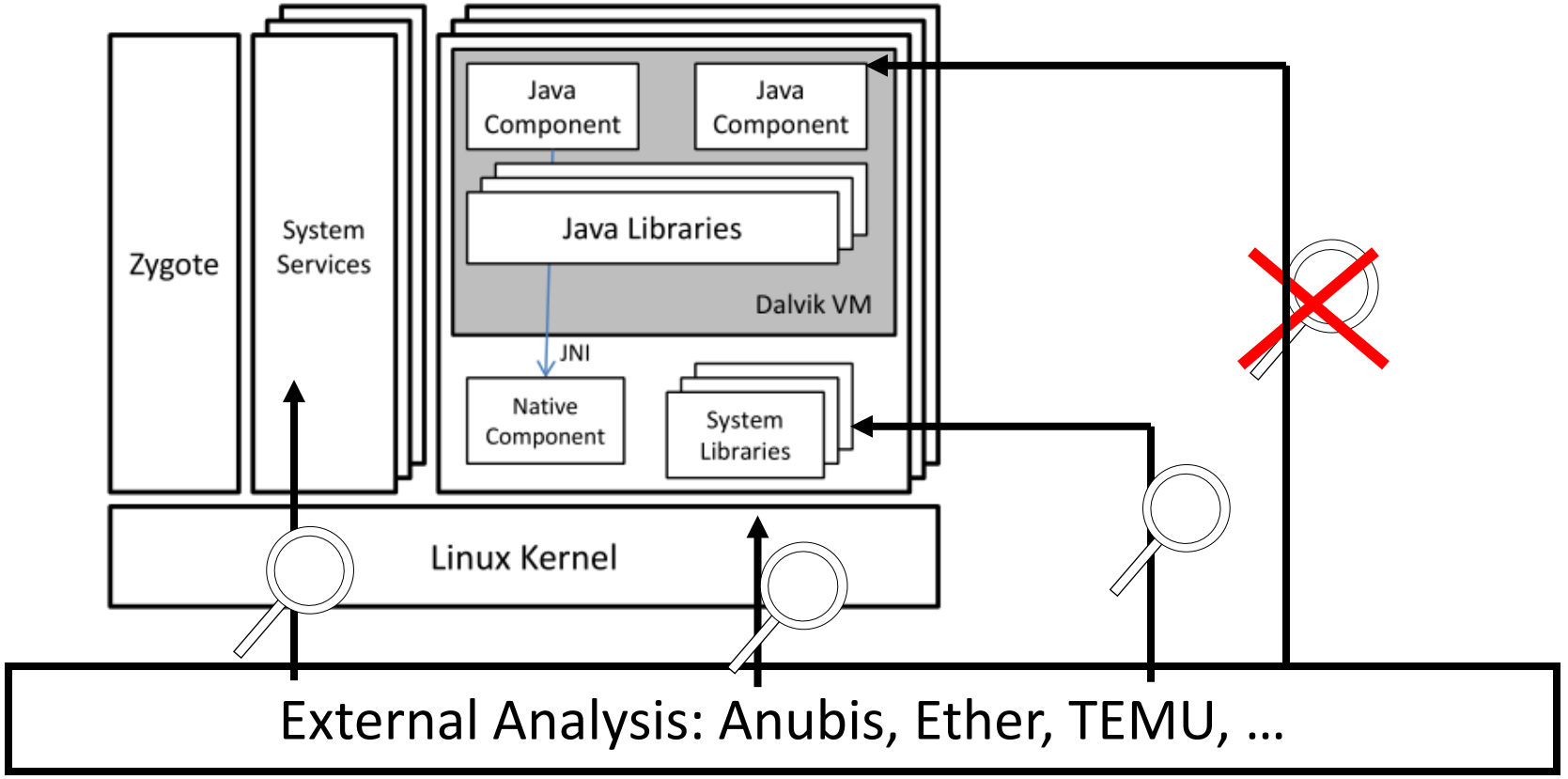
Motivation: Static Analysis



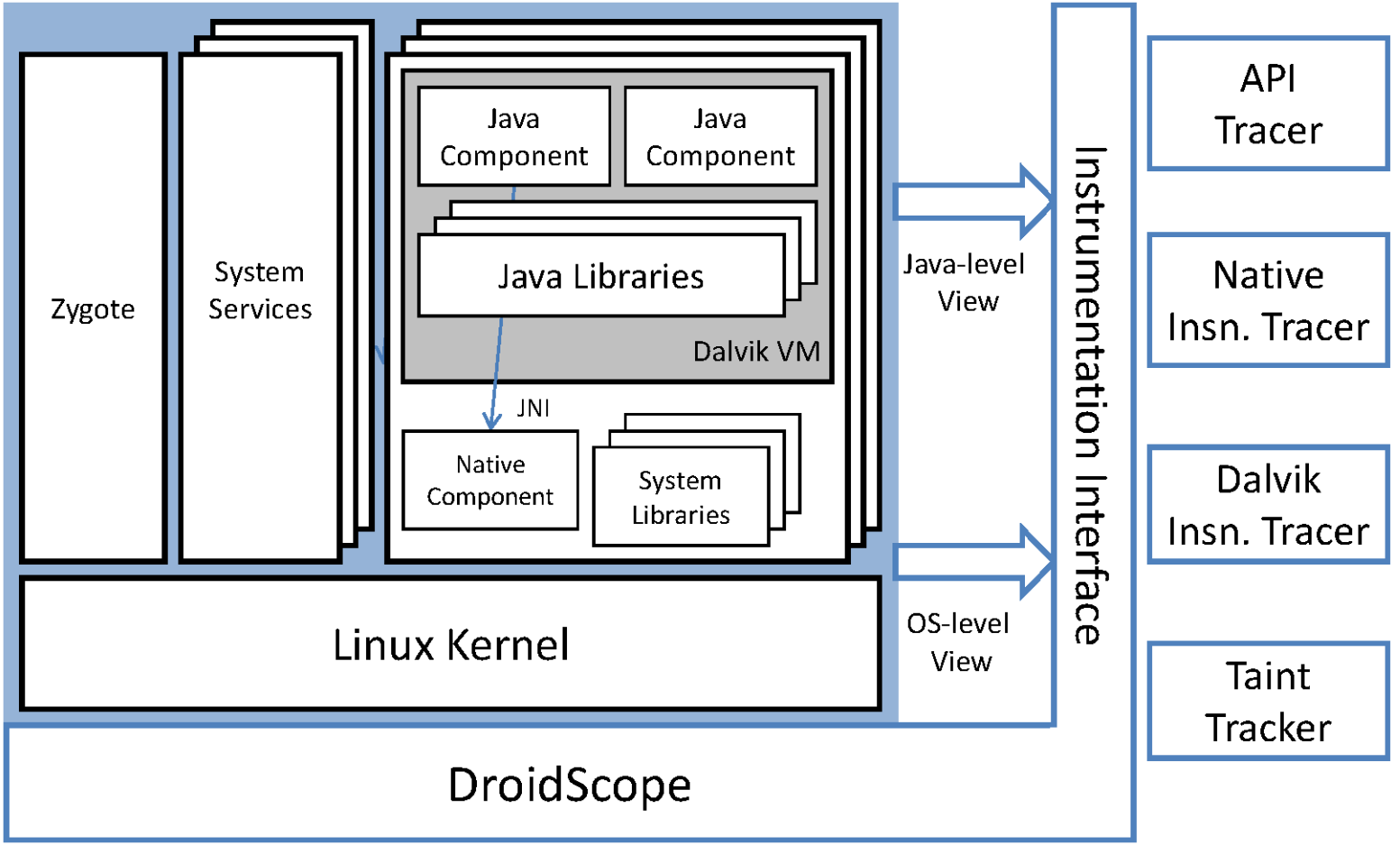
Motivation: Dynamic Analysis



Motivation: Dynamic Analysis



DroidScope Overview



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*

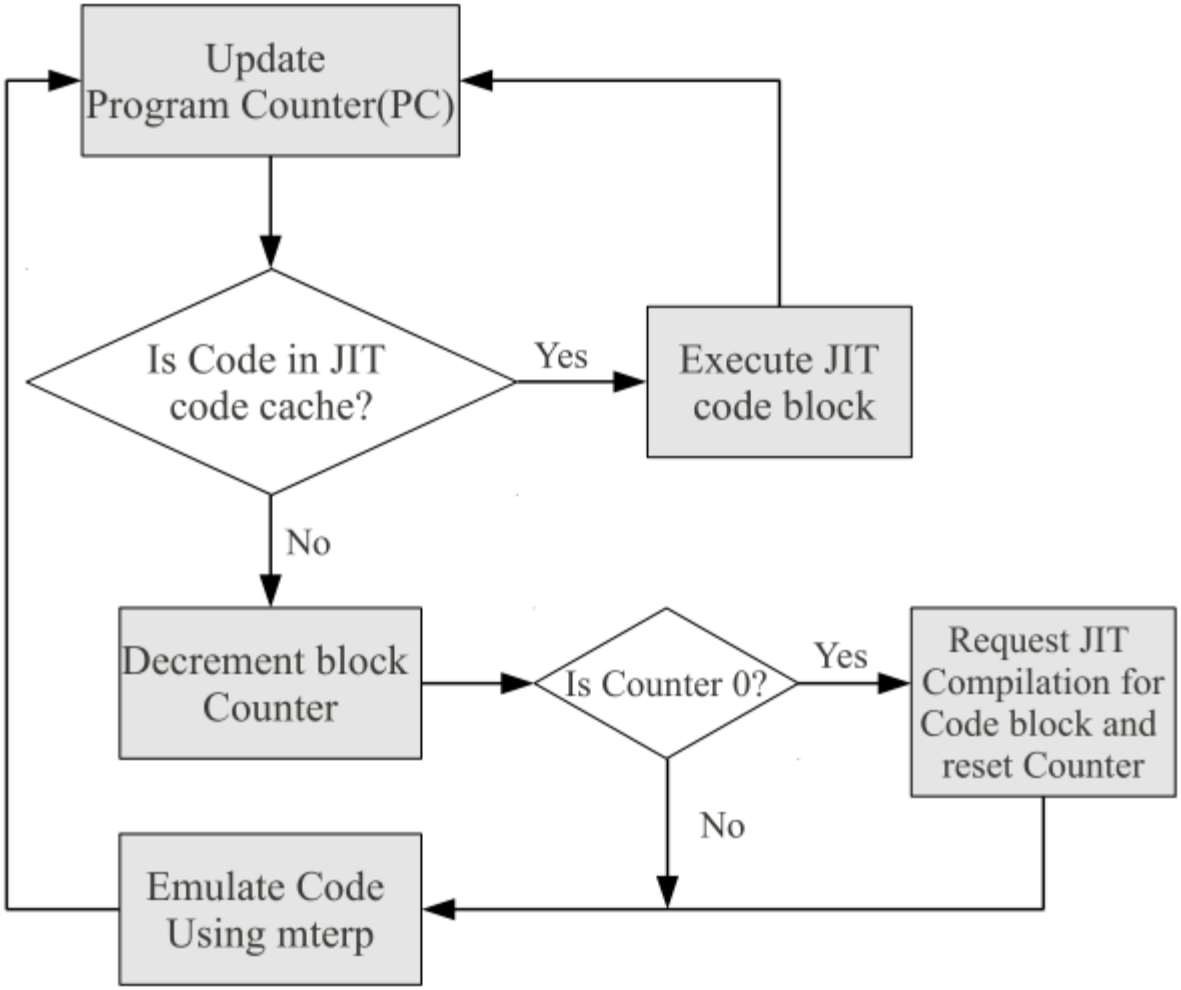
Java/Dalvik View

- 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



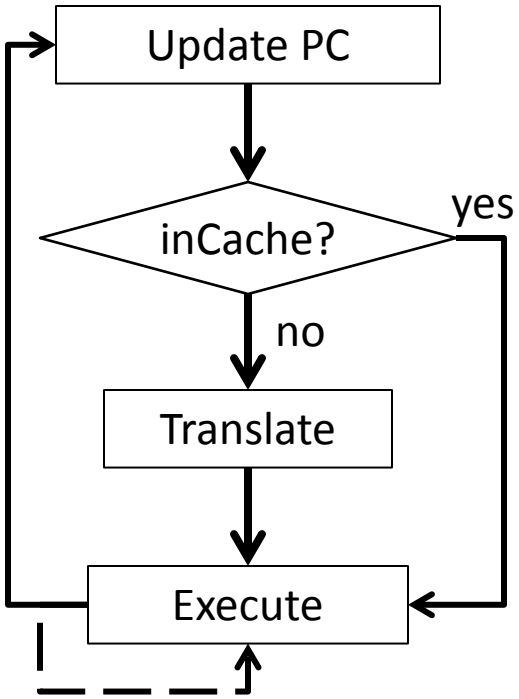
Roadmap

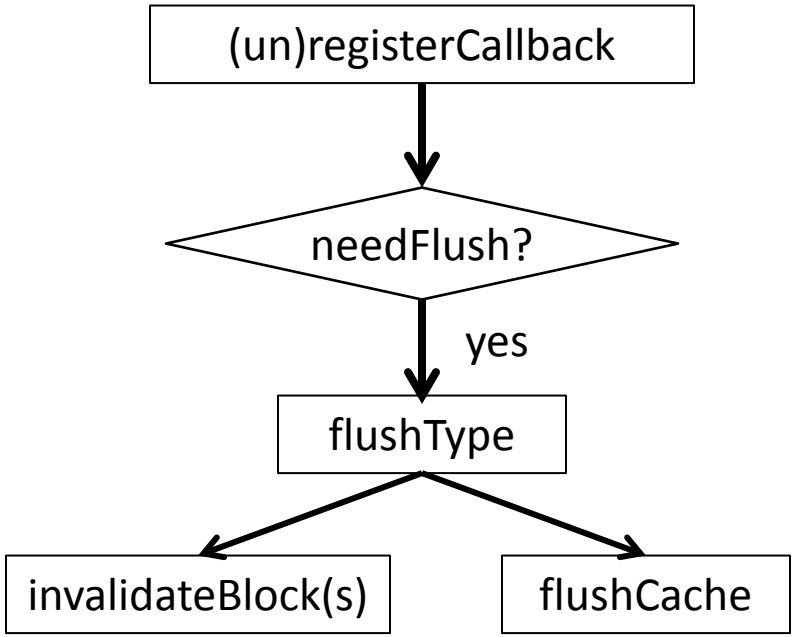
- ✓ External instrumentation
  - Linux context
  - Dalvik context
- ➢ Extensible: plugin-support / event-based interface
- Evaluation
  - Performance
  - Usage

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





Instrumentation

	NativeAPI	LinuxAPI	DalvikAPI
Events	instruction begin/end	context switch	Dalvik instruction begin
	register read/write	system call	method begin
	memory read/write	task begin/end	
	block begin/end	task updated	
		memory map updated	
Query & Set	memory read/write	query symbol database	query symbol database
	memory r/w with pgd	get current context	interpret Java object
	register read/write	get task list	get/set DVM state
	taint set/check		taint set/check objects
			disable JIT

Dalvik Instruction Tracer (Example)

```
1. void opcode_callback(uint32_t opcode) {
2.   printf("[%x] %s\n", GET_RPC, opcodeToStr(opcode));
3. }
4.
5. void module_callback(int pid) {
6.   if (bInitialized || (getIBase(pid) == 0))
7.     return;
8.
9.
10.  getModAddr(“dfk@classes.dex”, &startAddr, &endAddr);
11.  addDisableJITRange(pid, startAddr, endAddr);
12.  disableJITInit(getGetCodeAddrAddress(pid));
13.  addMterpOpcodesRange(pid, startAddr, endAddr);
14.  dalvikMterpInit(getIBase(pid));
15.  registerDalvikInsnBeginCb(&opcode_callback);
16.  bInitialized = 1;
17. }
18.
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

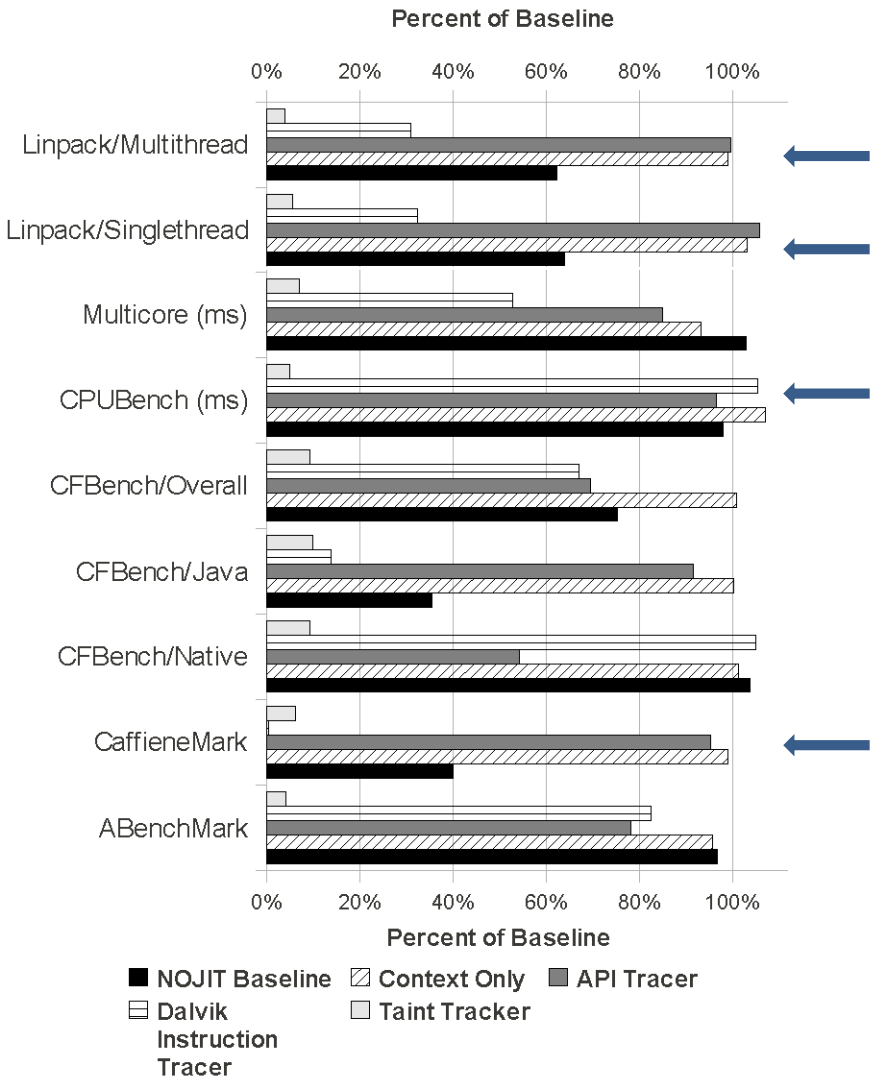
Implementation

- 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



APITracer vs. NOJIT

Results are not perfect

Dynamic Symbol Retrieval Overhead

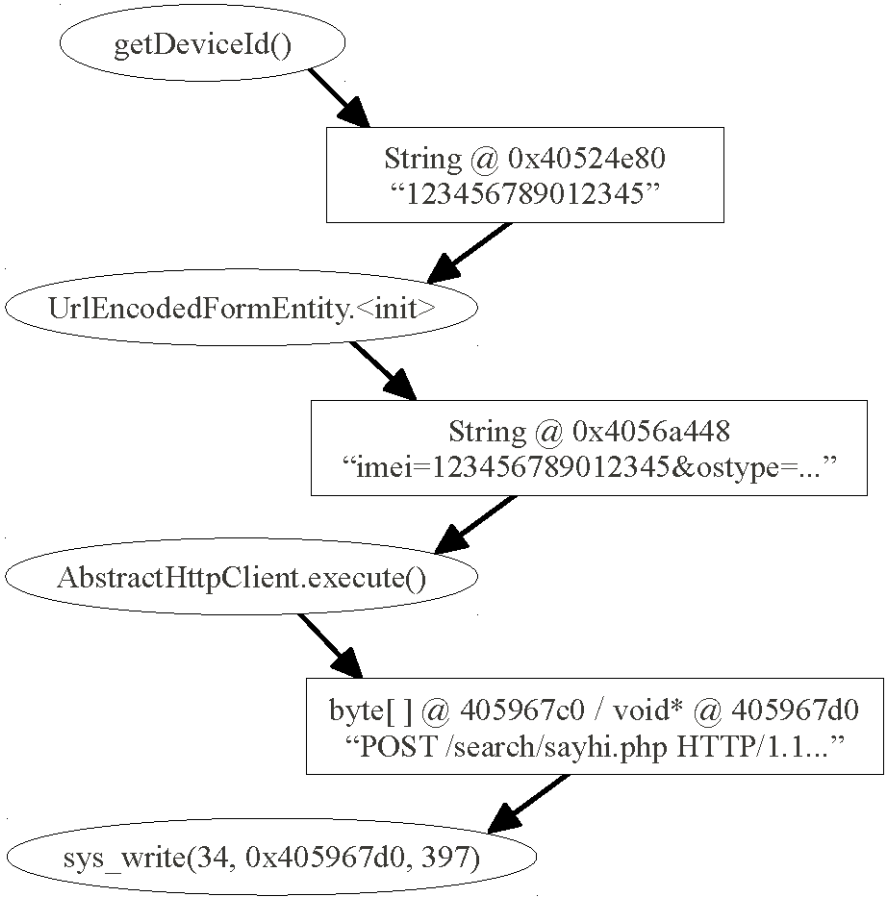
Usage Evaluation

- 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

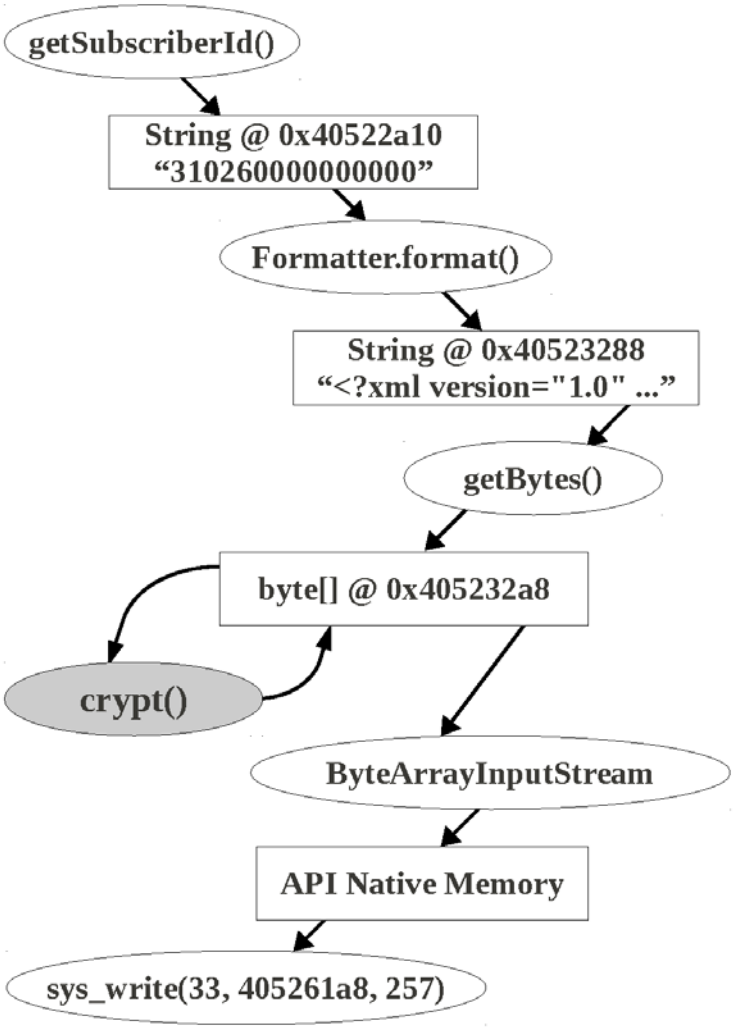
Droid Kung Fu: TaintTracker



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*

Droid Dream: TaintTracker



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?