Mobile Security Introduction

Department of Computer Science & Technology
Tsinghua University

Acknowledgement

- This lecture is extended and modified from lecture notes by:
  - Dr. Cliff Zou: CAP6135/CIS3360 courses
  - Dr. Yan Chen: EECS350 course
  - Dr. Nickolai Zeldovich: 6.858 course
  - Dr. Dang Song: CS161 course
  - Dr. Marco Cova 20009/20010 courses
  - Dr. Ninghui Li: CS426/CS526 courses

Contents

Introduction

- Android Markets are not in a position to provide security in more than a superficial way
- To broadly characterize the security of applications in the Android Market

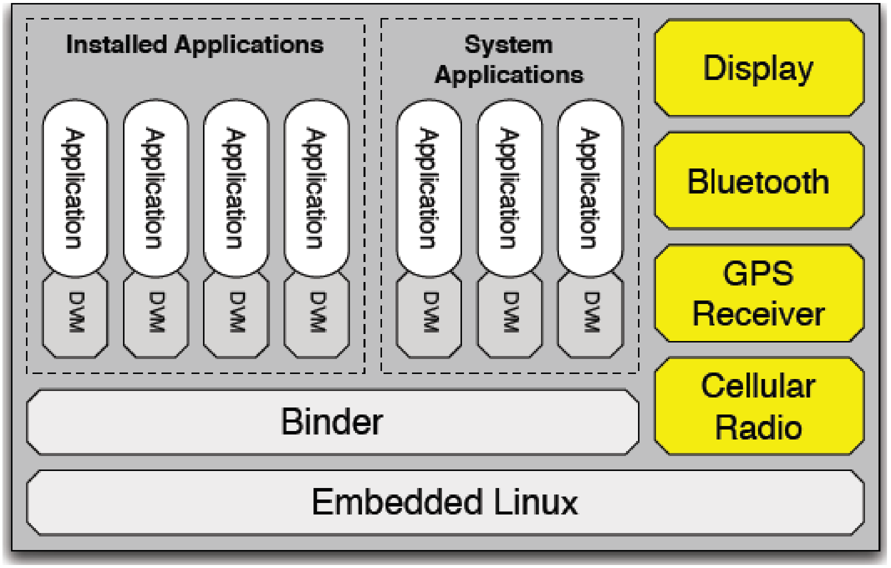


Figure 1: The Android system architecture

Introduction

- Wide misuse of privacy sensitive information
  - “Cookie-esque” tracking
- Found no evidence of telephony misuse
- Ad and analytic network libraries => 51% applications
  - AdMob => 29.09%
  - Google Ads => 18.72%
  - Many applications include more than one ad library
- Failed to securely use Android APIs

Background

- Dalvik Virtual Machine
  - JVM => .class
  - DVM => .dex
- Dalvik dx compiler

Constant Pool:
-References to other classes
-Method
-Numeri

Class Definition:
-Access flags

Data:
-Method code
-Info related to methods
-Variables

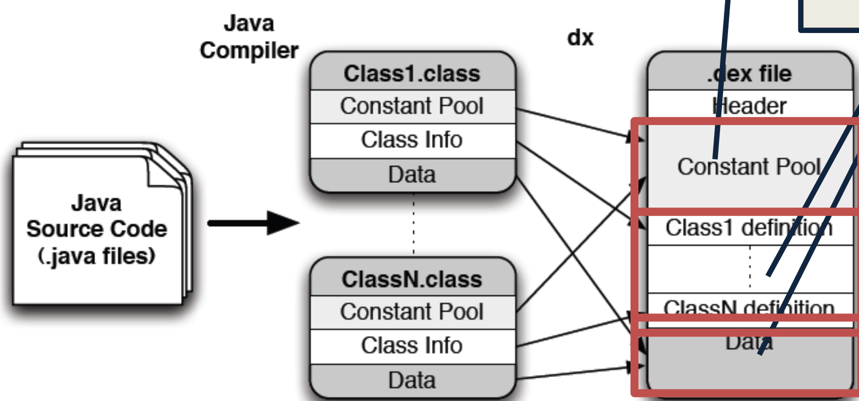


Figure 2: Compilation process for DVM applications

Background

- Register architecture
  - DVM: register-based
    - 2^16 available registers
  - JVM: stack-based
    - 200 opcodes

Background

- Instruction set
  - DVM
    - 218 opcodes
    - Longer instructions
    - Fewer instructions
    - 30% fewer instructions, but 35% larger code size (bytes)
  - JVM
    - 200 opcodes

Background

```
.method static constructor <clinit>()V
    .registers 6

    .prologue
    const/4 v5, 0x1

    const/4 v4, 0x0

    .line 41
    new-instance v0, Ljava/lang/Boolean;

    invoke-direct {v0, v4}, Ljava/lang/Boolean;-><init>(Z)V

    sput-object v0, Lcom/google/common/io/protocol/ProtoBuf;->FALSE:Ljava/lang/Boolean;

    .line 42
    new-instance v0, Ljava/lang/Boolean;

    invoke-direct {v0, v5}, Ljava/lang/Boolean;-><init>(Z)V

    sput-object v0, Lcom/google/common/io/protocol/ProtoBuf;->TRUE:Ljava/lang/Boolean;

    .line 59
    const/16 v0, 0x10

    new-array v0, v0, [Ljava/lang/Long;

                                                                          49,0-1
```

Background

- Constant pool structure
  - DVM
    - Single pool
    - dx eliminates some constants by inlining their values directly into the bytecode
  - JVM
    - Multiple

Background

- Ambiguous primitive types
  - DVM
    - int/float, long/double use the same opcodes
  - JVM
    - Different
- Null references
  - DVM
    - Not specify a null type
    - Use zero value

Background

- Comparison of object references
  - DVM
    - Comparison between two integers
    - Comparison of integer and zero
  - JVM
    - if_acmpeq / if_acmpne
    - ifnull / ifnonnull

Background

- Storage of primitive types in arrays
  – DVM
    - Ambiguous opcodes
    - aget for int/float, aget-wide for long/double

The ded decompiler

- To decompile the Java source rather than to operate on the DEX opcodes
  - Leverage existing tools for code analysis
  - Require access to source code to identify false-positives resulting from automated code analysis

The ded decompiler

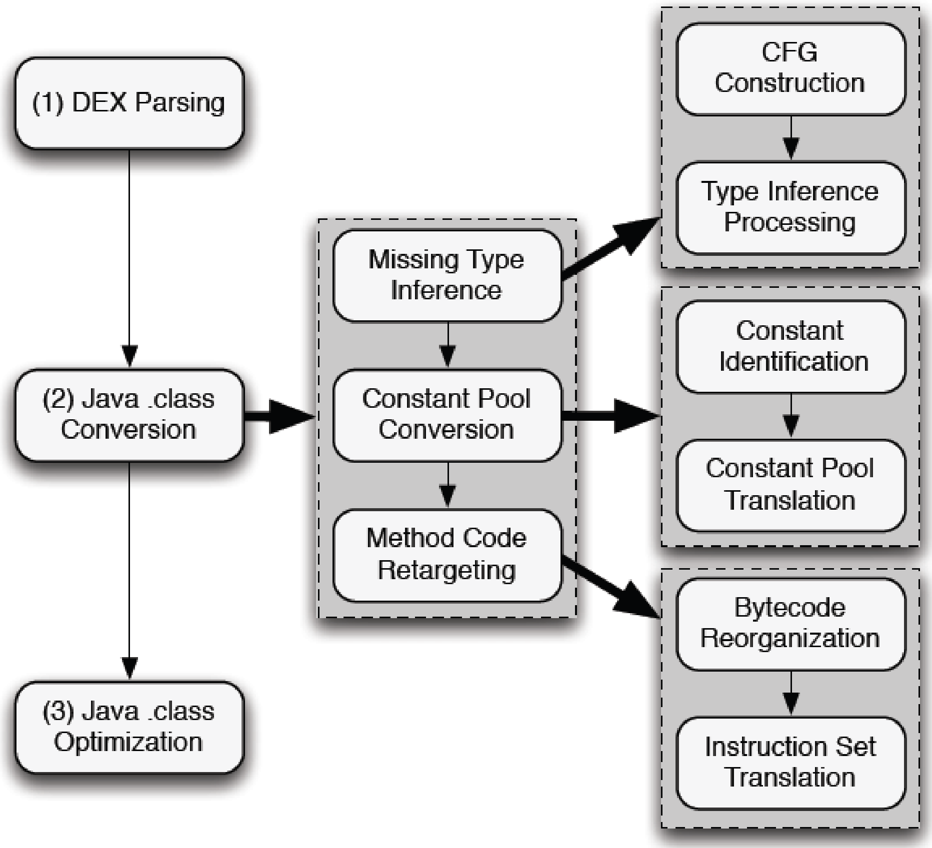


Figure 3: Dalvik bytecode retargeting

The ded decompiler

- Application Retargeting
  - Type Inference
    - Constant and variable declaration only specifies 32 or 64 bits
    - Comparison operators do not distinguish between integer and object reference comparison
    - Inference must be *path-sensitive*

The ded decompiler

- Application Retargeting (cont.)
  - To infer a register's type
    - Compare with a known type
    - add-int like instruction only operate on specific types
    - Use as return value or parameters of methods (via method signature)
    - Branch
      - Push onto an inference stack
      - DFS

The ded decompiler

- Constant Pool Conversion
  - .dex file vs. .class file
    - Single constant pool vs. multiple constant pool
    - Dalvik bytecode places primitive type constant directly in bytecode

The ded decompiler

- Method Code Retargeting
  - Address multidimensional arrays
  - Bytecode translation
    - ded maps each referenced register to a Java local variable table index
    - Instruction traslation
      - One Dalvik instruction -> multiple Java instructions
    - ded defines exception tables that describe try/catch/finally blocks

The ded decompiler

- Example:

Dalvik	Java
`add-int` d_0, s_0, s_1	`iload` s'_0
	`iload` s'_1
	`iadd`
	`istore` d'_0

The ded decompiler

- Optimization and Decompilation
  - Soot
  - While the Java bytecode generated by ded is legal, the source code failure rate is almost entirely due to Soot's inability

The ded decompiler

- Source Code Recovery Validation
  - decompilation time: 497.7 hours
  - 99.97% of total time -> Soot

Table 1: Studied Applications (from Android Market)

Category	Total Classes	Retargeted Classes	Decompiled Classes	LOC
Comics	5627	99.54%	94.72%	415625
Communication	23000	99.12%	92.32%	1832514
Demo	8012	99.90%	94.75%	830471
Entertainment	10300	99.64%	95.39%	709915
Finance	18375	99.34%	94.29%	1556392
Games (Arcade)	8508	99.27%	93.16%	766045
Games (Puzzle)	9809	99.38%	94.58%	727642
Games (Casino)	10754	99.39%	93.38%	985423
Games (Casual)	8047	99.33%	93.69%	681429
Health	11438	99.55%	94.69%	847511
Lifestyle	9548	99.69%	95.30%	778446
Multimedia	15539	99.20%	93.46%	1323805
News/Weather	14297	99.41%	94.52%	1123674
Productivity	14751	99.25%	94.87%	1443600
Reference	10596	99.69%	94.87%	887794
Shopping	15771	99.64%	96.25%	1371351
Social	23188	99.57%	95.23%	2048177
Libraries	2748	99.45%	94.18%	182655
Sports	8509	99.49%	94.44%	651881
Themes	4806	99.04%	93.30%	310203
Tools	9696	99.28%	95.29%	839866
Travel	18791	99.30%	94.47%	1419783
Total	**262110**	**99.41%**	**94.41%**	**21734202**

The ded decompiler

- Retargeting Failures
  - 0.59% of classes
  - Unresolved reference
  - Type violations by Android dex compiler
  - ded produces illegal bytecode (rare)
- Decompilation Failures
  - 5% of classes
  - Soot
  - Decompile traditonal Java program
  - 94.59%

The ded decompiler

- Future work
  - Fernflower
  - 98.04% recovery rate

Evaluating Android Security

- Analysis Specification
  - Use Fortify SCA static analysis suite
  - Control flow analysis
    - A control flow rule is an automaton

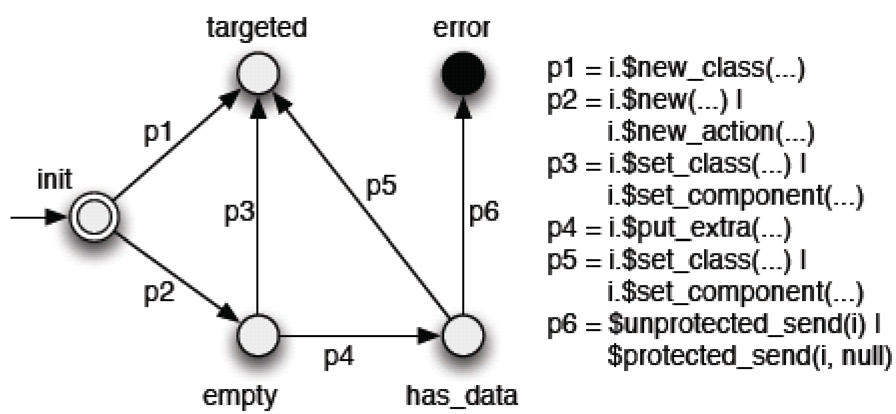


Figure 4: Example control flow specification

Evaluating Android Security

- Analysis Specification (cont.)
  - Data flow analysis
    - IMEI, IMSI, ICC-ID
    - Data flows between the sources and sinks are violations
  - Structural analysis
  - Semantic analysis
    - Ex: app does not send SMS to hard-coded targets

Evaluating Android Security

- Overview
  - Misuse of Phone Identifiers
  - Exposure of Physical Location
  - Abuse of Telephony Services
  - Eavesdropping on Audio/Video
  - Botnet Characteristics (Sockets)
  - Harvesting Installed Applications
  - Use of Advertisement Libraries
  - Dangerous Developer Libraries
  - Android-specific Vulnerabilities
  - General Java Application Vulnerabilities

Application Analysis Results

- Information Misuse
  - Phone Identifiers

Table 2: Access of Phone Identifier APIs

Identifier	# Calls	# Apps	# w/ Permission*
Phone Number	167	129	105
IMEI	378	216	184†
IMSI	38	30	27
ICC-ID	33	21	21
Total Unique	-	246	210†

* Defined as having the `READ_PHONE_STATE` permission.
† Only 1 app did not also have the `INTERNET` permission.

Table 3: Detected Data Flows to Network Sinks

	Phone Identifiers		Location Info.	
Sink	**# Flows**	**# Apps**	**# Flows**	**# Apps**
OutputStream	10	9	0	0
HttpClient Param	24	9	12	4
URL Object	59	19	49	10
Total Unique	-	33	-	13

Application Analysis Results

- Finding 1 - *Phone identifiers are frequently leaked through plaintext requests*
- Finding 2 - *Phone identifiers are used as device fingerprints*
- Finding 3 - *Phone identifiers, specifically the IMEI, are used to track individual users*
- Finding 4 - *The IMEI is tied to personally identifiable information (PII)*
- Finding 5 - *Not all phone identifier use leads to exfiltration*
- Finding 6 - *Phone identifiers are sent to advertisement and analytics servers*

Application Analysis Results

- Information Misuse (cont.)
  - Location Information
    - getLastKnownLocation()
    - LocationListener => requestLocationUpdates()

Table 4: Access of Location APIs

Identifier	# Uses	# Apps	# w/ Perm.*
getLastKnownLocation	428	204	148
LocationListener	652	469	282
requestLocationUpdates	316	146	128
Total Unique	-	505	304†

* Defined as having a LOCATION permission.

† In total, 5 apps did not also have the INTERNET permission.

Application Analysis Results

– Finding 7 - *The granularity of location reporting may not always be obvious to the user*
– Finding 8 - *Location information is sent to advertisement servers*

Application Analysis Results

- Phone Misuse
  - Telephony Services
    - A constant used for SMS destination number
    - Creation of URI objects with “tel:” prefix and the string “900”
    - URI objects with “tel:” prefix
  - Finding 9 - *Applications do not appear to be using fixed phone number services*
  - Finding 10 - *Applications do not appear to be misusing voice services*

Application Analysis Results

- Phone Misuse (cont.)
  - Background Audio/Video
    - Recording video without calling setPreviewDisplay()
    - AudioRecord.read() is not reachable from an Android activity component
    - MediaRecorder.start() is not reachable from an activity component

Application Analysis Results

- Finding 11 - *Applications do not appear to be misusing video recording*
- Finding 12 - *Applications do not appear to be misusing audio recording*

Application Analysis Results

- Phone Misuse (cont.)
  - Socket API Use
  - Finding 13 - *A small number of applications include code that uses the Socket class directly*
  - Finding 14 - *We found no evidence of malicious behavior by applications using Socket directly*

Application Analysis Results

- Phone Misuse (cont.)
  - Installed Applications
    - A set of get APIs returning the list of installed app
    - A set of query APIs that mirrors Android's runtime intent resolution
  - Finding 15 - *Applications do not appear to be harvesting information about which applications are installed on the phone*

Application Analysis Results

- Included Libraries
  - Advertisement and Analytics Libraries

Table 5: Identified Ad and Analytics Library Paths

Library Path	# Apps	Format	Obtains*
com/admob/android/ads	320	Obf.	L
com/google/ads	206	Plain	-
com/flurry/android	98	Obf.	-
com/qwapi/adclient/android	74	Plain	L, P, E
com/google/android/apps/analytics	67	Plain	-
com/adwhirl	60	Plain	L
com/mobclix/android/sdk	58	Plain	L, E‡
com/millennialmedia/android	52	Plain	-
com/zestadz/android	10	Plain	-
com/admarvel/android/ads	8	Plain	-
com/estsoft/adlocal	8	Plain	L
com/adfonic/android	5	Obf.	-
com/vdroid/ads	5	Obf.	L, E
com/greystripe/android/sdk	4	Obf.	E
com/medialets	4	Obf.	L
com/wooboo/adlib_android	4	Obf.	L, P, I†
com/adserver/adview	3	Obf.	L
com/tapjoy	3	Plain	-
com/inmobi/androidsdk	2	Plain	E‡
com/apegroup/ad	1	Plain	-
com/casee/adsdk	1	Plain	S
com/webtrends/mobile	1	Plain	L, E, S, I
Total Unique Apps	561	-	-

* L = Location; P = Phone number; E = IMEI; S = IMSI; I = ICC-ID
† In 1 app, the library included "L", while the other 3 included "P, I".
‡ Direct API use not decompiled, but wrapper *.getDeviceId()* called.

Application Analysis Results

– Finding 16 - *Ad and analytics library use of phone identifiers and location is sometimes configurable*
– Finding 17 - *Analytics library reporting frequency is often configurable*
– Finding 18 - *Ad and analytics libraries probe for permissions*

Application Analysis Results

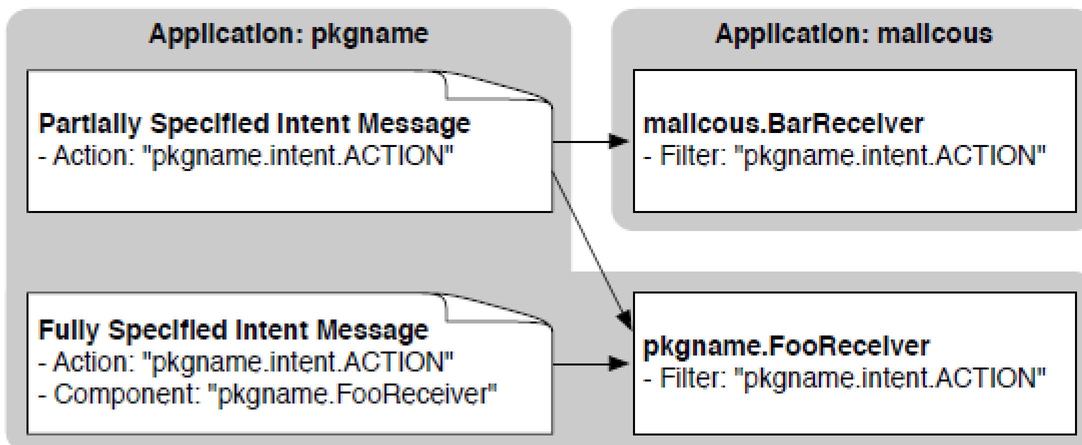
- Included Libraries (cont.)
  - Developer Tookits
  - Finding 19 - *Some developer toolkits replicate dangerous functionality*
    - jackeey.wallapaper sends identifiers to imnet.us
  - Finding 20 - *Some developer toolkits probe for permissions*
    - *checkPermission()*
  - Finding 21 - *Well-known brands sometimes commission developers that include dangerous functionality*

Application Analysis Results

- Android-specific Vulnerabilities
  - Leaking Information to Logs
    - READ_LOGS
    - Finding 22 - *Private information is written to Android's general logging interface*

Application Analysis Results

- Android-specific Vulnerabilities (cont.)
  - Leaking Information via IPC



  - Finding 23 - *Applications broadcast private information in IPC accessible to all applications*

Application Analysis Results

- Android-specific Vulnerabilities (cont.)
  - Unprotected Broadcast Receivers
    - Finding 24 - *Few applications are vulnerable to forging attacks to dynamic broadcast receivers*
  - Intent Injection Attacks
    - Finding 25 - *Some applications define intent addresses based on IPC input*

Application Analysis Results

- Android-specific Vulnerabilities (cont.)
  - Delegating Control
    - Pending intent
    - Cannot change values
    - But can fill in missing fields

    - Finding 26 - *Few applications unsafely delegate actions*
      - UI notification service
      - Alarm service
      - UI widget ←→ main application

Application Analysis Results

- Android-specific Vulnerabilities (cont.)
  - Null Checks on IPC Input
  - Finding 27 - *Applications frequently do not perform null checks on IPC input*
    - 53.7% (591 applications)

Application Analysis Results

- Android-specific Vulnerabilities (cont.)
  - Sdcard Use
    - 22.8% (251 applications)
  - JNI Use
    - 6.3% (69 applications)

Study Limitations

- Application popularity
- Data and control flows for IPC between components
- Source code recovery failures

- ProGuard
  - Obfuscate
  - Protect against readability

What This All Means

- Application certification
- Misuse of privacy sensitive information
- Cookie-esque tracking
- Ad and analytic libraries
  - Free applications!
- LOG / unprotected IPC

Conclusion

- ded decompiler
- Dangerous functionality

- Other Android potential security Problems
  - Application installation
  - Malicious JNI
  - phishing