2015 IEEE 39th Annual International Computers, Software & Applications Conference

CovDroid: A Black-Box Testing Coverage System for Android

Chao-Chun Yeh* ‡
avainyeh@itri.org.tw

Shih-Kun Huang † ‡
skhuang@cs.nctu.edu.tw

*Computational Intelligence Technology Center
Industrial Technology Research Institute
Hsinchu, Taiwan

† Information Technology Service Center
‡ Department of Computer Science
National Chiao Tung University
Hsinchu, Taiwan

***Abstract*—In android ecosystem, the Apps marketplace vendor faces huge number of Apps with irregular quality. Besides bug finding, coverage index is neglected for the current Android testing services. However it is also a challenge to measure the testing coverage without source code. In this paper, we provide a systematic approach to measure the App testing coverage for black-box testing and implement CovDroid, a black-box coverage system for android. Furthermore, we use a App with different test cases to prove our concept that coverage index can improve the App and measure test cases quality for App market or testing vendors.**

***Keyword* - Android, App Testing, Coverage Testing, Black Box testing**

I. INTRODUCTION

With the success of android based mobile device, the mobile App marketplace causes a paradigm shift in the software delivery model. Traditionally the software vendors develop program, test it with test cases and deliver tested software to the end users. However, with the new delivery model, the mobile marketplace vendor will face huge number of Apps with irregular quality that may cause vulnerabilities such as information leakage[1], premium service abuse [2], and root exploits [3]. In order to control the Apps quality, there are emerging third parties that provides testing service to help the marketplace vendor control the Apps quality.

However the current testing services [4-6] use Android testing tools Monkey [7] with random test or crowdsourcing test and then log the system behavior or provide extra SDK for developers to rewrite their App for testing[8]. The test report includes screenshots and related system evens such as crash, exception and ANR (Application Not Responding). However the information is not enough for comprehensive quality evaluation. For example coverage is an important factor for tester manager to measure the App quality. In traditional software industry, there are many coverage tools [9, 10] bundle with integrated development environment (IDE) for white-box testing. With the tools, developers can easily obtains the testing coverage from the testing cases. However in android ecosystem, the marketplace vendor faces the dilemma that the developer only provides the compiled execution file with binary format and it is difficult to measure the testing coverage without the source code. Currently the App testing vendor can provide many test cases to test the apps. It is not representative of the software quality for the most part only with the total test case number. Testing coverage should be consider as a critical index for those App testing vendors. From the mobile App delivery model, the software coverage can be tested easily in the testing services because of the scalability in the cloud environment.

In this paper, we provide a systematic approach to measure the App testing coverage for black-box testing and implement CovDroid, a black-box coverage tool for android. Furthermore, we use a third-party App with different test cases to prove our concept and explains the CovDroid benefit for App testing vendor.

II. BACKGROUND

In this section, we give a technical overview of android system and the current status of the android testing approaches. We also introduce the method to instrument coverage tag in android application.

A. Android and Android Market

Android is an open source project for smartphones and tablet computers. It is a Linux based mobile devices platform provided by Google and Open Handset Alliance (OHA). The Android software stack includes Linux kernel, middleware and build in applications. On top of the Linux kernel, there are the native libraries and android runtime written in C, and application software running on an application framework that includes Java-compatible libraries.

The Android Market (now called Google Play) is an App delivery platform. It allows developer to distribute their mobile App and end users to browse and download their interested App. In Google Play, developers can release the early version app to the selected users or group as alpha-testing or beta-testing. After those testing stages, developers can improve their App quality by the collecting feedback and issues.

B. Dalvike

Android uses Dalvik as process virtual machine until version 5.0. Developer writes android programs with Java and then

0730-3157/15 $31.00 © 2015 IEEE
DOI 10.1109/COMPSAC.2015.125

compiled to bytecode for the Java virtual machine. For hardware constrain such memory and process speed, the compiled bytecode translates into the compact Dalvik executable format (dex) and optimized version (odex). Since Android 4.4, Google provides an experimental new runtime environment with ahead-of-time (AOT) compilation. It compile the bytecode on the installation time instead of starting time.

C. Android Testing

For Android developer, they can test their App by Google Android Testing Framework including JUnit, test suites and monkeyrunner. The Android JUnit extensions supply component-specific test case classes providing extra helper methods for making mock objects for testing. Based on JUnit, Developers can use Android test suites within test packages to a class without calling Android API. For GUI testing, developer can use monkeyrunner to test Apps by sending pseudo-random events to the target testing device. For the whole testing process, developer can use the Android SDK tools including those testing tools and obtain testing coverage by extra coverage tool like EmmaMaven[11] with the source code.

However, for market vendors and third party testing vendors, they collect Apps from developers without source code. Currently, they only can test Apps by monkey test and then capture screens as the testing report.

D. Instrumentation and smali

In computer system, instrumentation is a design to diagnose errors, monitor performance and measure the level of a system behavior[12-15]. In Android system, instrumentation are used for test suits to write another app with testing classes such as ServiceTestCase, ActivityUnitTestCase, ApplicationTestCase, and ProviderTestCase and to test the target App[16]. It designs within Android framework and developers have to turn on the instrumentation mode in the AndroidManifest.xml file. Without Android famework,there are the devik level instrumentation tools such smali[17] or ASMDEX[18]. They are the assembler/disassembler tools for the dex format and based on Jasmin's/dedexer's syntax, with the instrumentation functions such as annotations, debug info and line info, etc.

III. THE COVDROID

In this section, we give an overview of our black-box testing coverage toolkit –CovDroid, our proposed approach to measure the coverage for black-box testing, and explain the essential steps involved in testing an App with the toolkit. The modules of CovDroid are illustrated in Figure 1.

First, the testing app is disassembled into Jasmin format [19], instrumented coverage-related information and finally assembled into a new testing app. The new testing app is installed automatically in the android device and then testing script from the tester or other automatic test system will interacts with the testing app. When the testing app is executed, the execution monitor will inspect the execution status as execution log files. After collecting the execution log, the coverage report manager calculates the coverage and effective index for the test case.

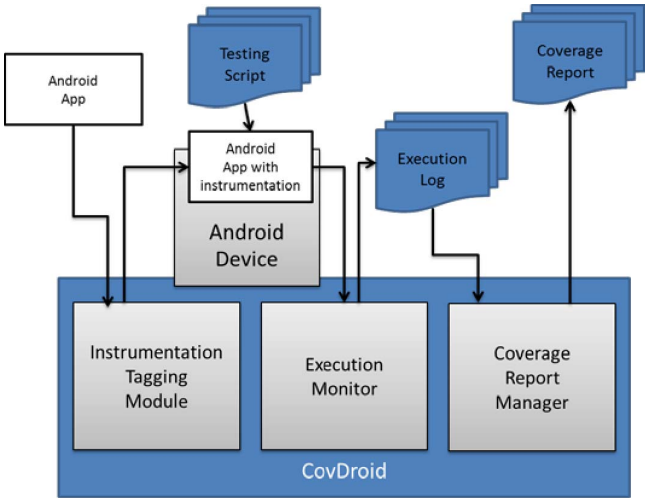

Figure 1: CovDroid Architecture

The CovDroid is developed in a virtual machine including 4 vCPU and 16GB vRAM on a 2.7Ghz i7 CPU with 32GB RAM physical machine, Ubuntu 12.04 64-bit desktop edition for the host OS. The CovDroid is mainly written in Python and integrates third-party open source library[20] that can parse smali files of APKs into tree based structure and inject monitor code to smali tree. It includes three modules:

- Instrumentation Tagging Module
  The Instrumentation tagging module disassemble dex format into Jasmin syntax file with smali[21] and then parse the files to find the class methods for instrumentation. We use class level coverage instead of branch coverage because of the performance consideration and the bug report readability. From the code style guidelines[22],it suggests a method should be under 40 lines or so. Instrumentation tagging information contains three fields such as class name, method name and an identifier. The table1 shows that in method beginning part, we adjust the register number, add two constant strings such as v0 and v1 and then invoke log API (invoke-static) to notify the execution status,. After tagging, the module rebuilds a new testing app from the smali code with backsmali.

Table 1: Instrumentation tagging example

```
.class public Litri/icl/w200/inforleak/InformLeakActivity;                          #class field
.method public GetAccount(Landroid/view/View;)V                                    #method field
   …
   …
   const-string v0, "CovDroid"                                                     #identifier field
   const-string v1, "Litri/icl/w200/inforleak/InformLeakActivity;->GetAccount(Landroid/view/View;)V"   # execution class/method
   invoke-static {/
v0, v1
/}, Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;)I
```

Table 2: Log fields and example

Fields	Example
Timestamp	05-14 21:13:28.159
Identifier	CovDroid
Process ID	215
Execution class/method	Litri/icl/w200/inforleak/Inform LeakActivity ;->showMsg(Ljava/lang/CharSequence;)V

- Execution Monitor

  The execution monitor uses Android Debug Bridge[23] tool to generate the execution status such as timestamp, identifier, process id and execution class/method(see Figure2). The status can provide coverage report manager to calculate the test case coverage and test case effectiveness index. Table 2 lists the fields and an example.

```
D/smali   (  215): Litri/icl/w200/inforleak/InformLeakActivity;-><init>()V
D/smali   (  215): Litri/icl/w200/inforleak/InformLeakActivity;->onCreate(Landro
id/os/Bundle;)V
D/smali   (  215): Litri/icl/w200/inforleak/InformLeakActivity;->GetIMSI(Landroi
d/view/View;)V
D/smali   (  215): Litri/icl/w200/inforleak/InformLeakActivity;->md5(Ljava/lang/
String;)Ljava/lang/String;
D/smali   (  215): Litri/icl/w200/inforleak/InformLeakActivity;->postData(Ljava/
lang/String;)V
D/smali   (  215): Litri/icl/w200/inforleak/InformLeakActivity;->showMsg(Ljava/l
ang/CharSequence;)V
```

Figure 2: Execution Method Information

- Coverage Report Manager

  After the test case execution, the coverage report manager counts the total methods number from smali file generated by instrumentation tagging module and collects execution log and calculates the test case coverage index, test case effectiveness, test case similarity[24] index as follow:

$$\text{Coverage Index} = \frac{\text{the executed method number}}{\text{total methods}}$$

$$\text{Effectiveness Index} = \frac{\textit{the executed method number}}{\text{Execution Time}}$$

Test Case Coverage Similarity

$$= \frac{\text{Test Case A Hits Block(i)} \cdot \text{Test Case B Hits Block(i)}}{|\text{Test Case A Block Hits}||\text{Test Case B Block Hits}|}$$

The test manager can estimate the test case quantity, quality and similarity based on the three indexes. With the test case coverage index, the test manager can decide how many test cases should be involved with the target testing app to archive higher coverage rate. The test case quality can be represented as time cost with the test case effectiveness index. Besides providing those indexes, the coverage report manager list hot spot method (total hits rate over 95%) for performance turning and weakness method (total hit rate under 25%) for the further testing process.

IV. CASE STUDY

To prove the concept of CovDroid, we use a third party utility App- informLeak (see Figure 3) for evaluation. The App is design to gather device and user information such as device information (IMSI, IMEI,GPS and Camera), user profile (account and phone number) and activities (SMS and brower history) for further personal behavior mining and device management. Table 3 lists the detail description for each button.

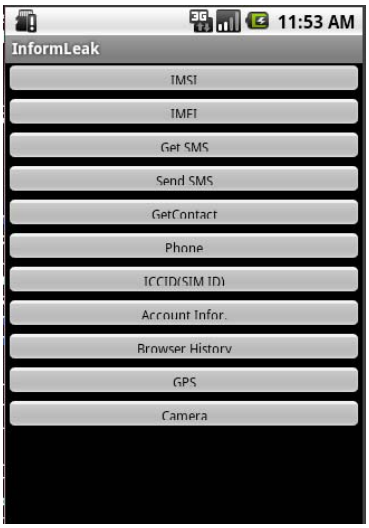

Figure 3: InformLeak App

Table 3: Button Name & Description

Button Name	Description
IMSI	Get the device International Mobile Subscriber Identity
IMEI	Get the device international mobile equipment identity number
Get SMS	Get the SMS list
Send SMS	Send the SMS
Get Contact	Get the user contact list
Phone	Get the Phone Number
ICCD(SIM ID)	Get the SIM card unique serial number
Account Infor	Get the user account information
Browser History	Get the browser history
GPS	Get the GPS value
Camera	Get the camera roll list

We use monkey to generate two test cases that contain pseudo-random user event sequences. After the instrumentation tagging module processing, the testing app increases 2.34% in size and instruments 34 methods in 11 classes. We run two test cases with the inforLeak app in android device. Testing result is shown in Table 4. With coverage index, we can found that test case 1 (73.52%) get higher coverage rate than test case 2 (58.82%) and obtain un-coverage methods from execution log. However based on effectiveness index, test case 2 (0.133) is more effective than test case 1 (0.083). If resource is limited, we or the testing vendors can adjust priority for different test cases based on their effectiveness index value. In addition to those benefits, CovDroid can provide extra information like executed method name for fault location or App performance turning

Table 4: Testing Result

	Test Case 1	Test Case 2
Time(sec)	300	150
Total Execution methods	25	20
Coverage Index	73.52%	58.82%
Effectiveness index	0.083	0.133

Figure 4 shows method block hit number of the two test cases (see Appendix A). From the hit sequences, we use can calculate the test case coverage similarity (59.68%). If two cases have high similarity, we can schedule only one for testing in resource limitation environment. For performance index, the CovDroid suggests that method-block-15, method-block-18 and method-block-17 are hot spot areas because their hit rate is over 95% total hit rate (32.3). For weakness method, the CovDroid shows that method-block-9, method-block-12, method-block-16, method-block-21, method-block-30, method-block-33 and method-block-31 are most untested methods. With the coverage index and effectiveness index, the testing vendors can provide quality of service for different developer need.

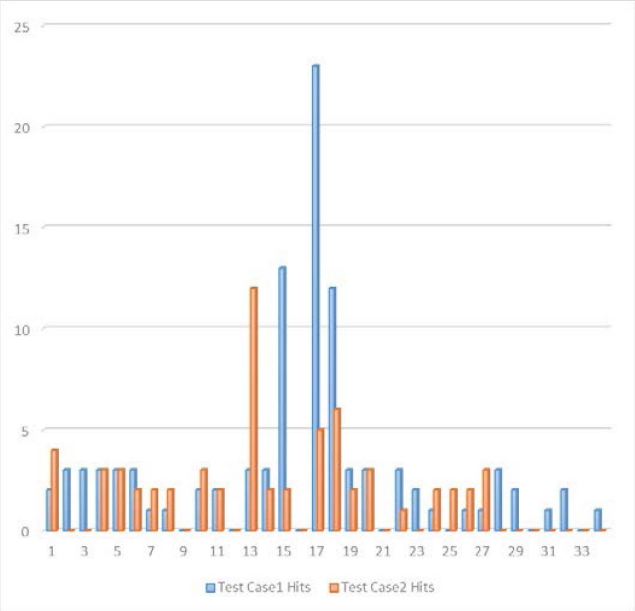

Figure 4: Method Block Hit Number

V. Related Work

Emma[25] is a Java based code coverage tool that can instrument classes to obtain different coverage types such as class, method, line and basic block. However the Emma limitation is that it can only handle java archive file format (jar) and doesn't support dalvik executable format (dex).

Dalvik Debug Monitor Server[26] (DDMS) is integrated into Eclipse and can profile the execution method information. However the method is focus on the system level API rather than App's functional block [27].

Mahmood, Riyadh, et al.[28] focus on automatic Android apps testing for the security and robustness issue and describe an Android-specific program analysis technique capable of generating a large number of test cases for fuzzing an app. They decompile the Android application package file to obtain the source files and use EMMA to monitor and report the testing coverage.

Android Maven Plugin[11] provides line coverage for developer with source code. It combines the compiled java classes and Maven bytecode with instrumentation. The experienced developer use the tool to measure their App qualify in developing stage with some performance degradation.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a systematic approach to measure the App testing coverage for black-box testing and implement CovDroid, a black-box coverage tool for android. We use a case study to prove our concept that coverage is an essential element for testing and market vendor to guide the App quality and based the CovDroid, the App market and testing vendor can provide more developers to improve their App quality with the method information. While CovDroid demonstrates the results, it is still a prototype and we will focus on the following aspects:

1. Hybird more coverage criteria: the current CovDroid only supports method coverage. We plan to extend coverage criteria to hybrid branch coverage and method coverage for selecting the proper one at run time.

2. Real time monitor: the CovDroid calculate coverage index and effectiveness index in offline mode. We will redesign the execution monitor and coverage report manager to provide real time monitor for tester to adjust test case priority dynamically.

3. Mapping the coverage information to user interface: the CovDroid coverage report is based on jasmin structure. It is complicated for tester with user interface control to identify the relationship between internal jasmine and external UI widgets.

ACKNOWLEDGMENT

This work is a partial result of Project No. E301AR9800 conducted by Industrial Technology Research Institute under sponsorship of the Ministry of Economic Affairs, Taiwan, R.O.C.

REFERENCES

[1] *HTC Vulnerability*. Available: http://www.cvedetails.com/cve/CVE-2011-3975/

[2] *HTC IQRD service Vulnerability*. Available: http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-1939

[3] Available: http://www.cvedetails.com/cve/CVE-2011-1823/

[4] "The vold volume manager daemon Vulnerability."

[5] *Testflight service*. Available: https://testflightapp.com/

[6] *Android Testing service*. Available: http://www.testin.cn

[7] *UI/Application Exerciser Monkey*. Available: http://developer.android.com/tools/help/monkey.html

[8] *Leanplum Android Testing*. Available: https://www.leanplum.com/docs#/setup/android

[9] M. Shahid and S. Ibrahim, "An Evaluation of Test Coverage Tools in Software Testing," in *2011 International Conference on Telecommunication Technology and Applications Proc. of CSIT*, 2011.

[10] Q. Yang, J. J. Li, and D. M. Weiss, "A survey of coverage-based testing tools," *The Computer Journal,* vol. 52, pp. 589-597, 2009.

[11] *maven-android-plugin*. Available: https://code.google.com/p/maven-android-plugin/wiki/EmmaMaven

[12] *Source code instrumentation overview*. Available: http://www-01.ibm.com/support/knowledgecenter/SSSHUF_8.0.0/com.ibm.rational.testrt.doc/topics/cinstruovw.html

[13] R. E. Filman and K. Havelund, "Source-code instrumentation and quantification of events," 2002.

[14] Z. Wang, A. Sanchez, and A. Herkersdorf, "Scisim: a software performance estimation framework using source code instrumentation," in *Proceedings of the 7th international workshop on Software and performance*, 2008, pp. 33-42.

[15] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney*, et al.*, "Pin: building customized program analysis tools with dynamic instrumentation," *ACM Sigplan Notices,* vol. 40, pp. 190-200, 2005.

[16] "Google API- Instrumentation."

[17] *smali- an assembler/disassembler for Android's dex format*. Available: https://code.google.com/p/smali/

[18] *Introduction to the ASMDEX Bytecode Framework*. Available: http://asm.ow2.org/doc/tutorial-asmdex.html

[19] *JASMIN- an assembler for the Java Virtual Machine*. Available: http://jasmin.sourceforge.net/

[20] K. Yang. *An APK instrumentation library and DroidBox APIMonitor*. Available: https://github.com/kelwin/apkil

[21] *An assembler/disassembler for Android's dex format*. Available: https://code.google.com/p/smali/

[22] *Android Code Style Guidelines for Contributors*. Available: https://source.android.com/source/code-style.html

[23] *Android Debug Bridge*. Available: http://developer.android.com/tools/help/adb.html

[24] *Cosine similarity*. Available: http://en.wikipedia.org/wiki/Cosine_similarity

[25] *EMMA: a free Java code coverage tool*. Available: http://emma.sourceforge.net/

[26] *Dalvik Debug Monitor Server*. Available: http://developer.android.com/tools/debugging/ddms.html

[27] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 627-638.

[28] R. Mahmood, N. Esfahani, T. Kacem, N. Mirzaei, S. Malek, and A. Stavrou, "A whitebox approach for automated security testing of Android applications on the cloud," in *Automation of Software Test (AST), 2012 7th International Workshop on*, 2012, pp. 22-28.

APPENDIX-A METHOD BLOCK HIT NUMBER

Method Block Number	Test Case1 Hit number	Test Case2 Hit number
1	2	4
2	3	0
3	3	0
4	3	3
5	3	3
6	3	2
7	1	2
8	1	2
9	0	0
10	2	3
11	2	2
12	0	0
13	3	12
14	3	2
15	13	2
16	0	0
17	23	5
18	12	6
19	3	2
20	3	3
21	0	0
22	3	1
23	2	0
24	1	2
25	0	2
26	1	2
27	1	3
28	3	0
29	2	0
30	0	0
31	1	0
32	2	0
33	0	0
34	1	0
Total Hits	100	63