ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 4, April 2017

Inter-Procedural, Light Weight and Call-Back Aware Resource Leak Detection in Android Apps

Roopa .S. Kumar, Smitha .J.C

M.Tech Student, Dept. of C.S.E, APJ Abdul Kalam Technological University, Kerala, India

Assistant Professor, Dept. of C.S.E, APJ Abdul Kalam Technological University, Kerala, India

ABSTRACT: Android devices include resources such as Camera, Media Player and Sensors. These resources need programmers to explicitly request and release them. Serious problems such as performance degradation or system crash can occur due to missing of release operations. This type of defects is called is a leak. A large number of existing works on testing and analysing android apps, still there remains lot of challenging problems. In this paper, a modified relda model is proposed which is a light weight, precise and static resource leak detection tool. A resource table is collected and which includes the android reference requires developers release manually. Based on this table a general approach is designed to automatically detect resource leaks. A function call back graph is created for each android application to make precise the inter procedural analysis which handles function calls of user defined methods and thecallbacks invoked by android framework at the same time. Resource leak is a type of bugs which is found in android apps ,but developers don't have enough time to detect and fix them. In this paper, a system is proposed that can detect resource leaks in Android Apps automatically.

KEYWORDS: Android apps; resource leak; inter-procedure; function calls

I. INTRODUCTION

Nowadays, Android smartphones are becoming popular. A recent study shows that Android share reaches 82.8 %in smartphone markets. However, the quality of android apps is still worrisome, because a majority of apps are developed by relatively small teams, which may not afford expensive testing. Android phones come up with many components embedded in them. The components can be divided into two categories. They are traditional resources and exotic resources. Traditional resources which can be found in desktop, like CPU, memory and screen. Exotic resources are resources such as GPS, camera and different types of sensors. The exotic resources which will consume much energy in Android phones and they drain the phone's battery at a high rate .In addition to this, explicit user management is needed for this resources.The developers need to release and request resources manually. Absence of these operations might lead to high energy consumption, or even system crash may occur. This type of leaks is called resource leaks. Resource management is nowadays a challenging task to developers. There are many reasons for that. First reason is that unlike memory is allocated and recycled by virtual machine, these resources require programmers to explicitly turn them on and off. The execution path of an android app is often complicated due to large number of callbacks and Android's event driven nature. Second reason is that developers have to often push their apps to markets in a short time, and also focussing onto user friendliness and functionality of their apps. The developers often get complaint about the performance related problems from the users, and also these complaints contain insufficient information for the localization of the bugs. Third reason is that, programmers might misunderstand the Android API specifications. Resource leak is a common type of bugs which is found in android apps, but the developers often do not have enough time to detect and fix them. Detecting a resource leak in the app is to find a reachable path that requests but does not release resources.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 4, April 2017

II. RELATED WORK

Casper .S. Jensen et.al [1]in his paper has proposed about the Automated software testing aims to detect errors by producing test inputs that cover as much of the application sourcecode as possible. Applications for mobile devices are typically event-driven, which raises the challenge of automaticallyproducing event sequences that result in high coverage.Some existing approaches use random or model-basedtesting that largely treats the application as a black box.Other approaches use symbolic execution, either startingfrom the entry points of the applications or on specific eventsequences. A common limitation of the existing approachesis that they often fail to reach the parts of the applicationcode that require more complex event sequences. Proposed a two-phase technique for automatically findingevent sequences that reach a given target line in theapplication code. The first phase performs concolic execution
to build summaries of the individual event handlers of the application. The second phase builds event sequences backward from the target, using the summaries together with a UI model of the application. Our experiments on a collection of open source Android applications show that this technique can successfully produce event sequences that reach challenging targets.

Tanzirul Azim et.al[2] has proposed about the systematic exploration of Android apps is an enabler for a variety of app analysis and testing tasks. Performing the exploration while apps run on actual phones is essential for exploring the full range of app capabilities. However, exploring real-world apps on real phones is challenging due to non-determinism, non-standard control flow, scalabilityand overhead constraints. Relying on end-users to conduct the exploration might not be very effective: we performeda 7-user study on popular Android apps, and found that the combined 7-user coverage was 30.08% of the app screensand 6.46% of the app methods. Prior approaches for automated exploration of Android apps have run apps in an emulator or focused on small apps whose source code was available. To address these problems, we present A3E, anapproach and tool that allows substantial Android apps to be explored systematically while running on actual phones, yet without requiring access to the app's source code. The key insight of our approach is to use a static, taint-style, dataflow analysis on the app bytecode in a novel way, to construct a high-level control flow graph that captures legal transitions among activities (app screens).We then use this graph to develop an exploration strategy named Targeted Exploration that permits fast, direct exploration of activities, including activities that would be difficult to reach during normal use. We also developed a strategy named Depth-first Exploration that mimics user actions for exploring activities and their constituents in a slower, but more systematic way. To measure the effectiveness of our techniques, we use two metrics: activity coverage (number of screens explored) and method coverage.

Ronalda Garzia et.al [3]in his paper has proposed about the typestate reflects how the legal operations on imperative objects can change at runtime as their internal state changes. A typestate checker can statically ensure, for instance, that an object method is only called when the object is in a state for which the operation is well-defined. Prior work has shown how modular typestate checking can be achieved thanks to access permissions and state guarantees. However, typestate was not treated as a primitive language concept: typestate checkers are an additional verification layer on top of an existing language. In contrast, a typestate-oriented programming language directly supports expressing typestates. For example, in the Plaid programming language, the typestate of an object directly corresponds to its class, and that class can change dynamically. Plaid objects have not just typestate-dependent interfaces, but also typestate-dependent behaviors and runtime representations. This paper lays foundations for typestate-oriented programming by formalizing a nominal object-oriented language with mutable state that integrates typestate change and typestate checking as primitive concepts. We first describe a statically-typed language, called Featherweight Typestate (FT), where the types of object references are augmented with access permissions and state guarantees. We describe a novel flow-sensitive permission-based type system for FT. Because static typestate checking is still too rigid for some applications, we then extend this language into a gradually-typed language, called Gradual Featherweight Typestate (GFT). This language extends the notion of gradual typing to account for typestate: gradual typestate checking seamlessly combines static and dynamic checking by automatically inserting runtime checks into programs. The gradual type system of GFT allows programmers to write dynamically safe code even whenthe static type checker can only partly verify it.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 4, April 2017

Steven Arzt et.al[4] in his paper has proposed about the today's smartphones are a ubiquitous source of private and confidential data. At the same time, smartphone users are plagued by carelessly programmed apps that leak important data by accident,and by malicious apps that exploit their given privileges to copy such data intentionally. While existing static taint-analysis approaches have the potential of detecting such data leaks ahead of time, all approaches for Android use a number of coarse-grain approximations that can yield high numbers of missed leaks and false alarms.
In this work we thus present FLOWDROID, a novel and highly precise static taint analysis for Android applications. A precise model of Android's lifecycle allows the analysis to properly handle callbacks invoked by the Android framework, while context, flow, field and object-sensitivity allows the analysis to reduce the number of false alarms. Novel on-demand algorithms help FLOWDROID maintain high efficiency and precision at the same time. We also propose DROIDBENCH, an open test suite for evaluating the effectiveness and accuracy of taint-analysis tools specifically for Android apps. As we show through a set of experiments using SecuriBench Micro, DROIDBENCH, and a set of well-known Androidtest applications, FLOWDROID finds a very high fraction of data leaks while keeping the rate of false positives low. On DROIDBENCH, FLOWDROID achieves 93% recall and 86% precision, greatly outperforming the commercial tools IBM AppScan Source and Fortify SCA. FLOWDROID successfully finds leaks in a subset of 500 apps from Google Play and about 1,000 malware apps from the VirusShare project.

Jiajin Huang et.al[5] in his paper has proposed about the Android smartphones are becoming increasingly popular. The open nature of Android allows users to install miscellaneous applications, including the malicious ones, from third-party marketplaces without rigorous sanity checks. A large portion of existing malwares perform stealthy operations such as sending short messages, making phone calls and HTTP connections, and installing additional malicious components. In this paper, we propose a novel technique to detect such stealthy behavior. We model stealthy behaviour as the program behavior that mismatches with user interface, which denotes the user's expectation of program behavior. We use static program analysis to attribute a top level function that is usually a user interaction function with the behavior it performs. Then we analyze the text extracted from the user interface component associated with the top level function. Semantic mismatch of the two indicates stealthy behavior. To evaluate AsDroid, we download a pool of 182 apps that are potentially problematic by looking at their permissions. Among the 182 apps, AsDroid reports stealthy behaviors in 113 apps, with 28 false positives and 11 false negatives.

Domenico Amalfitano [6] et.al in his paper has proposed about the AndroidRipper, an automated technique that tests Android apps via their Graphical User Interface (GUI). AndroidRipper is based on a user-interface driven ripper that automatically explores the app's GUI with the aim of exercising the application in a structured manner.
Matthew Amold [8] et.al in his paper has proposed about approach to address this problem by using a specialized runtime environment called Quality Virtual Machine (QVM).
Florean Gross [9] et.al in his paper has proposed about QVM efficiently detects defects by continuously monitoring the execution of the application in a production setting. QVM enables the efficient checking of violations of user-specified correctness properties, e.g., typestate safety properties, Java assertions, and heap properties pertaining to ownership.
Stephen.J.fink[10] et.al in his paper has proposed about the challenge of sound typestate verification, with acceptable precision, for real-world Java programs.
Yulei Sui [11] et.al in his paper has proposed about the static detector, Saber, for detecting mem-ory leaks in C programs. Leveraging recent advances on sparse pointer analysis, Saber is the _rst to use a full-sparse value-ow analysis for leak detection.
Guoqing Xu [12] et.al in his paper has proposed about specification-based technique called LeakChaser that cannot only capture precisely the unnecessary references leading to leaks, but also explain, with high-level semantics, why these references become unnecessary.

III. PROPOSED SYSTEM

There are three types of resources. They are Exclusive resources, Memory consuming resources and also Energy consuming resources. Exclusive resources are those resources that can be used by only one app at a time. Failing to release these resources will prevent other apps from accessing them. Memory consuming resources consumes much more memory than the general resources. Energy Consuming resources are those resources that will consume much

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 4, April 2017

more energy than the general resources. This approach should have several characteristics. They are Full automation, high efficiency with enough accuracy, scalability. Full automation in which testers need to provide target APKs and can work without any extra user interactions. High efficiency with enough accuracy static analysis of large-scale and real world android apps are time consuming. Conformation of bugs and fixing them is labor-intensive for the developers. Next is the scalability. The Android framework will always upgrade and add new resources. Thus collecting the resource request and release APIs as a static file to describe the analysis objectives.
FCG (Function Call Graph) construction

Algorithm 1 Construct the FCG
1: mainAct1 = getMainActivity(app)
2: for each invoked function F of mainAct do
3: create a new FCG node fnode for F in fcg
4: construct fcg nodes(f, fnode)
5: end for
Algorithm 1 shows the FCG construction

construct fcg nodes(F, fnode)
1: if F has been visited then
2: return
3: end if
4: for each instruction ins in F do
5: if ins is an “invoke” instruction then
6: for each invoked function F’ of ins do
7: create a new FCG node fnode’ for F’ in fcg
8: if F’’ is not an implicit callback then
9: add fnode’ into fnode.children
10: end if
11: construct fcg nodes (F’, fnode’)
12: end for
13: end if
14: end for
The above algorithm shows the FCG construction. An Android app is taken as the input. Main activity of the app is obtained by using the getMainActivity method. The algorithm starts from the main activity to obtain all reachable methods and construction of nodes and edges in FCG. Each node will be containing an attribute called children to reperesent the corresponding nodes of its invoked functions.

VFG(Value flow graph) Construction
Algorithm 2Construct VFGs
1: node map = {}
2: create a RootNode root as the entry node of vfg
3: for each block b in cfg do
4: (firstnode, lastnode) = create vfg nodes(c)
5: node map[c] = (firstnode, lastnode)
6: if b is the entry block of cfg then
7: create an edge from root to firstnode
8: end if
9: end for
10: for each block b in cfg do
11: (firstnode, lastnode) = node map[b]
12: for each block c’ in c.nexts do
13: (firstnode’, lastnode’) = node map[c’]
14: create an edge from lastnode to firstnode’
15: end for
16: end for

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 4, April 2017

17: minimize vfg.
create vfg nodes(c)
1: firstnode = lastnode = null
2: for each instruction ins in block c do
3: if ins invokes a resource request API then
4: create a SourceNode node
5: else if ins invokes a resource release API then
6: create a FreeNode node
7: else if ins invokes a resource-irrelevant function
then
8: create a CallNode node
9: else if ins is a “return” instruction then
10: create an ExitNode node
11: end if
12: if firstnode is null then
13: firstnode = lastnode = node
14: else
15: create an edge from lastnode to node
16: lastnode = node
17: end if
18: end for
19: if there is no VFG-related node in block b then
20: create a NopNode node
21: firstnode = lastnode = node
22: end if
23: return (firstnode, lastnode)

Algorithm 3 shows the VFG construction. CFG is taken as the input and VFG is generated. The variable node_map will be matching the CFG blocks and also the VFG nodes. RootNode is taken as the entry node. The method create_vfg_nodes returns the first and last nodes. The function create_vfg_nodes traverses the instructions in block c and creates new appropriate nodes.

IV. SIMULATION RESULTS

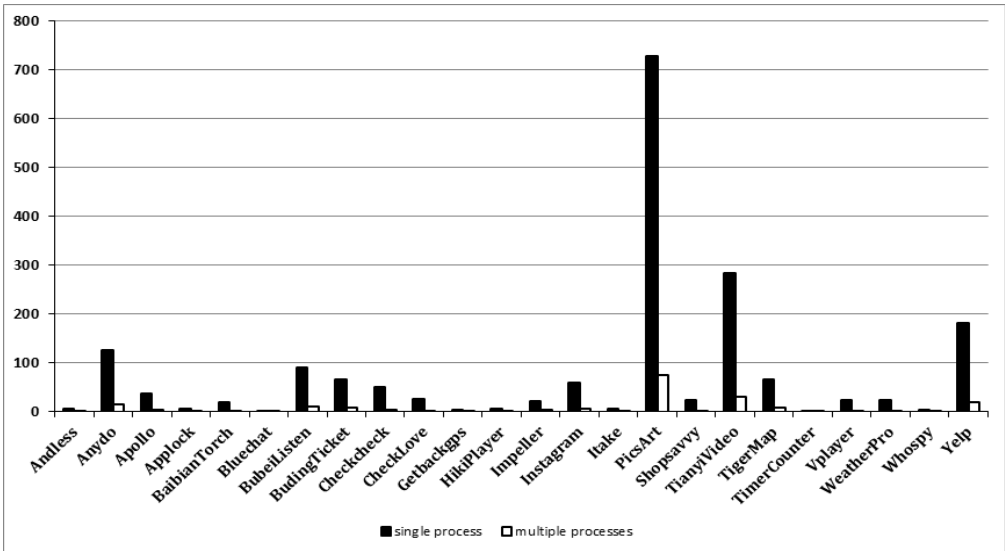

Fig 1.Resource summary time comparison between single and multiple processes.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Website: www.ijircce.com

Vol. 5, Issue 4, April 2017

This figure compares the resource summary time comparison between single and the multiple processes. The resource summary time single thread method is seven times more than the multiple thread method.

V. CONCLUSION AND FUTURE WORK

This paper deals with the analysis of resource leak detection in android apps. The definition of categories and the categories of resources is discussed in this paper. In this a CBG (callback graph) is constructed which is used to handle callbacks invoked by android framework. An VFG (value flow graph) is also constructed. In future extending the proposed static analysis process to other similar problems.

REFERENCES

1. Casper Svenning Jensen, Mukul R. Prasad, and Anders Møller.Automated testing with targeted event sequence generation. In Proceedings of International Symposium on Software Testing and Anal-
ysis, pages 67–77, 2013.
2. Tanzirul Azim and Iulian Neamtiu. Targeted and depth-first exploration for systematic testing of Android apps. In Proceedings ofthe 2013 ACM SIGPLAN International Conference on Object OrientedProgramming, Systems, Languages, and Applications, pages 641–660, 2013.
3. Ronald Garcia, ´Eric Tanter, Roger Wolff, and Jonathan Aldrich. Foundations of typestate-oriented programming. ACMTransactionon Programming Languages and Systems, 36(4):12:1–12:44, 2014.
4. Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for Android apps. In Proceedings of the 2014 ACM SIGPLAN Conference onProgramming Language Design and Implementation, page 29, 2014.
5. Jianjun Huang, Xiangyu Zhang, Lin Tan, Peng Wang, and Bin Liang. Asdroid: detecting stealthy behaviors in Android applications by user interface and program behavior contradiction. In Proceedings of the 36th International Conference on Software Engineer-ing, pages 1036–1046, 2014.
6. Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, and Atif M. Memon. Using GUI ripping for automated testing of android applications. In Proceedings ofthe 2012 IEEE/ACM International Conference on Automated SoftwareEngineering, pages 258–261, 2012.
7.Matthew Arnold, Martin T. Vechev, and Eran Yahav. QVM: anefficient runtime for detecting defects in deployed systems. In Proceedings of the 23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, pages143–162, 2008.
8. Stephen J. Fink, Eran Yahav, Nurit Dor, G. Ramalingam, and Emmanuel Geay. Effective typestate verification in the presence of aliasing. In Proceedings of the ACM/SIGSOFT InternationalSymposium on Software Testing and Analysis, pages 133–144, 2006.
9. Florian Gross, Gordon Fraser, and Andreas Zeller. Search-based system testing: high coverage, no false alarms. In Proceedings ofInternational Symposium on Software Testing and Analysis, pages 67–77, 2012.
10. Cuixiong Hu and Iulian Neamtiu. Automating GUI testing for Android applications. In Proceedings of the 6th InternationalWorkshop on Automation of Software Test, pages 77–83, 2011.
11. Yulei Sui, Ding Ye, and Jingling Xue. Static memory leak detection using full-sparse value-flow analysis. In Proceedings of InternationalSymposium on Software Testing and Analysis, pages 254–264, 2012.
12. Guoqing (Harry) Xu, Michael D. Bond, Feng Qin, and Atanas Rountev. Leakchaser: helping programmers narrow down causes of memory leaks. In Proceedings of the 32nd ACM SIGPLANConference on Programming Language Design and Implementation, pages 270–282, 2011.

BIOGRAPHY

Roopa.S.Kumar is M.Tech final year student under APJ Abdul Kalam technological university in Lourdes Matha College of science and technology. She has completed B.Tech in Lourdes Matha College from Kerala University.Her interests are Computer networks, medical image compression, Resource Leak Detection in Android Apps.

Ms Smitha J C is anAssistant Professor in Computer Science and Engineering in Lourdes Matha College of Science and Technology, Trivandrum, Kerala. Her research interests are Image Processing, Artificial Neural Network,and Computer Graphics.