Evaluating Effectiveness of Static Analysis Testing Tools On Android Applications

XXXXXXXXXXXXXXX
XXXXXXXXX
XXXXXXXXX
XXXXXXXXX
XXXXXXXXX, XX, XXX
XXXXXX@XXXXX.XXX

ABSTRACT

Mobile applications (apps) have taken the computing world by storm. On the Android platform alone, there are over a million apps available for download on over one billion Android devices. Android development can be very lucrative, but is also extremely competitive due to the number of options that users have. Users will often choose alternative apps over ones which are delivered late or are buggy. Developers are under an immense amount of pressure to create apps quickly, and make them as high quality as possible. Static analysis testing tools are often relied upon to automatically perform a brunt of the testing to quickly and cheaply discover possible errors in the app. Unfortunately, all static analysis tools are not created equal.

In the following paper, we investigate the effectiveness of three leading static analysis testing tools for Android: FindBugs, PMD and Android Lint. We first created a defect oracle of fifty unique errors from eight defect categories and then ran the tools against this oracle comparing the results of our examined tools in terms of precision, recall, and F-measure. Our primary contributions of this work are an unbiased analysis of these three leading testing tools, along with a publicly accessible defect oracle which may be used for future evaluations and research.

While none of the three tools led in all evaluated categories, FindBugs performed at the most consistently high level and led in the categories of Recall and F-measure.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

Keywords

Android, Testing, Static Analysis

1. INTRODUCTION

Mobile apps have changed the way that we live and work allowing for an unprecedented level of power at almost any time in any location. There are currently over one million Android apps [4] on over one billion devices [20] which give developers a wide range of customers, but also give customers a diverse range of options. For many users, a deciding factor for the app they will choose is its quality, which they may witness firsthand or may read from reviews of the app [13].

Developers are under an immense amount of pressure to deliver apps and their updates in a prompt, cost effective and high quality manner. Static analysis tools are routinely used to quickly, and automatically test apps for a variety of problems including defects [11] and security vulnerabilities [21, 23]. Static analysis tools evaluate a system and its components without actually executing the software [1].

Android developers often use static analysis tools to assist with defect detection. Unfortunately, not all static analysis tools are created equally. *The goal of our research is to evaluate static analysis tools that are used for testing Android applications.* We evaluated three leading Android static analysis testing tools: FindBugs [3], PMD [18], and Android Lint [2] assessing them in a variety of areas including precision, recall and F-measure. In our comparison of the tools, we sought to answer the following research questions:

RQ1: *What categories of defects are found by static analysis tools?*
We categorized defects into eight different groups including 'compatibility', 'concurrency', 'correctness', 'maintainability', 'performance', 'reliability', 'security', and 'usability.' We found that PMD discovered defects in 6/8 of the groups, while FindBugs identified defects in 5/8 and Android Lint only found defects in 2/8 groups.

RQ2: *Which static analysis tool is the most effective at finding defects?*
We evaluated each tool against the known bugs in our oracle in terms of precision, accuracy and F-measure. While no tool led in all three areas, FindBugs discovered defects at the most consistently high level in each category.

The remainder of this paper is organized as follows: Section 2 discusses previous work with static analysis tools. Section 3 provides the methodology we used to conduct our analysis. The results of our analysis are provided and discussed in Section 4, while Section 5 provides information about our public data set which may be accessed and used by other researchers. Section 6 discusses limitations of our study and future work to be conducted while Section 7 provides a conclusion of our study.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'16, April 4-8, 2016, Pisa, Italy.
Copyright 2016 ACM 978-1-4503-3739-7/16/04...$15.00.
http://dx.doi.org/xx.xxxx/xxxxxxx.xxxxxxx ...$15.00.

2. RELATED WORK

The importance of static analysis tools have been demonstrated in innumerable ways with research being dedicated to improving their performance while reducing their false positives [12]. Static analysis tools have been developed for a variety of purposes for a range of programming languages and platforms including defect detection [11] and security analysis [8, 19, 21, 23]. There are also a wide range of testing tools developed for Android apps including GUI testing [10], tools which relied upon symbolic analysis [16], and those that found bugs using pattern analysis [15]. There are even static analysis testing tools that search for energy leaks in apps [9, 24], and examine for cache performance [6] and those that measure defect density [17].

Studies have compared the effectiveness of various static analysis tools. Rutar et al. [18] evaluated several leading static analysis tools on Java applications. They found that no single static analysis tool should ever be deemed to be perfect and made the argument for the concurrent use of several tools to ensure maximum testing coverage. While this work was profound, it did not evaluate the tools against one another, or create an oracle to measure the precision, recall or F-measure of the tools.

Research has also examined how static analysis testing tools are used in the real world. Ayewah et al. [5] reported on their experiences running FindBugs against Sun's JDK and at Google where it was integrated into their standard development process. Some of their reported findings included the lack of perceived value that many developers had in running static analysis testing tools, and that developers pay the most attention to high priority issues. They also found that users are typically less likely to repair defects found in older code.

Thung et al. [22] examined the extent that field defects were discovered using several leading static analysis tools including FindBugs, PMD, and JLint. They found that while many field defects could be identified by these tools, a significant portion of issues were not identified. This work also characterized many of the missed defects.

Previous research has demonstrated the importance of having high quality apps and their direct impact on user ratings. Khalid et al. [14] examined the relationship of code quality (using FindBugs) and the app's user rating. They found that "Bad Practice", "Internationalization" and "Performance" error categories were much more common in lower rated compared to high-rated apps. The work also found a direct correlation between these defects and complaints in the app's reviews. They determined that app developers could raise their use rating by resolving these specific types of defects using static analysis tools.

3. METHODOLOGY

The goal of our research is to evaluate the quality of three leading static analysis tools for Android. Effective static analysis defect detectors are important so developers may be sure that their tools are discovering a significant portion of the actual defects in the app, but are not returning a high level of false positives. In order to evaluate the static analysis tools, we first created a defect oracle containing various defect patterns, ran each tool on three different Android applications, matched discovered defects with the defect patterns from the oracle, and then calculated statistics to determine the effectiveness of each tool. In the following sections, we will describe the tools, our defect oracle, and how the tools were ran. The purpose is to not only provide a background of how we attained our results, but to allow future researchers to re-run our analysis should they choose to do so.

3.1 Selected Tools

We selected three leading static analysis tools for our comparison: FindBugs, PMD, and Android Lint. We chose these tools based upon their popularity and their use in previous work [14, 22]. The tools are briefly described below:

FindBugs

A popular open source static analysis tool that examines files for detects using approximately 300 pre-defined bug patterns. Each FindBugs defect pattern is categorized based on correctness, bad practice, performance, and internationalization. FindBugs also rates the reported bugs based on its confidence and rank level. The confidence is based how certain - low, medium, or high - the tool is that the identified defect will result in a failure. The rank level - of "concern", "troubling", "scary", or "scariest" - is how much of a problem the identified defect is likely to be. For example, an unused variable may be rated as troubling, while a memory leak may be rated as scary. In our analysis, we used FindBugs in the Android Studio IDE.

PMD

This tool detects potential defects in software written in Java. When performing the code analysis, the tool compares syntax patterns against predefined PMD rulesets. These rulesets define various categories of bad practices and potential bugs. Each ruleset is classified as one of the following categories: efficiency, maintainability, reliability, and usability. Each identified defect is rated based on the severity - "minor", "major", "critical", or "blocker". PMD reports all detected warnings and unlike FindBugs, does not rate them based on confidence. This leads to a large number of warnings that may contain false positives.

Android Lint

A static analysis tool that is dedicated to Android development. This tool is provided by the Android SDK and is the default inspection tool used in the Android Studio IDE. In addition to Java code analysis, the tool also analyzes compatibility of the code with the various versions of the Android API. The tool reports issues with the structural quality of the source code and each issue is reported with a message and a severity level used for prioritization. Android Lint may be ran from the command line or from Android studio. In our analysis, we ran Android Lint from the command line.

3.2 Defect Oracle

An important goal of our work was to create an unbiased, high quality defect oracle that could also be used for future tool evaluations. The defect oracle is a repository of defects used to analyze the effectiveness of the three static analysis tools. Each tool contains a set of defect patterns that are used during code inspections to identify defects and each of these defect patterns are unique to the tool itself and cannot be directly compared to other tools. In order to provide a common set of potential defects the following procedure was performed:

Step 1: Identify the common defect patterns for each tool: Since there are hundreds of possible defect patterns among the tools, we limited our research to only use high-priority patterns. This resulted in a total of fifty nine defect patterns compiled from all three static analysis tools. We analyzed each of the fifty nine pattern descriptions and determined which patterns were duplicates. From this analysis, we were able to identify fifty unique defect patterns, which were added to the defect oracle.

Step 2: Create a unique defect identifier that maps to each tool's defect pattern: Each defect pattern was assigned a unique defect identifier. If the defect pattern was determined to be a duplicate between multiple tools, we mapped the tool's defect pattern description to the unique defect identifier in the defect oracle. This step was essential for tracking defects discovered by multiple tools with different defect naming conventions.

Step 3: Analyze, describe, and categorize each defect: We created a common defect description for each defect identifier and sorted them into the following categories: compatibility, concurrency, correctness, maintainability, performance, reliability, security, and usability.

Step 4: Determine if the item is actually a defect.: We manually examined each of the potential defects in the oracle and ranked them 1-5 based on the likelihood of resulting in a failure; 1 most likely a defect, 5 least likely a defect. Table 1 shows an example subset of the ranked defects in the oracle. The 'Reliability' defect is ranked low (5) since it most likely wouldn't cause an issue, but it still may be a security concern. The 'Correctness' defect has a high (1) priority due to our certainty that the item is a real defect and would very likely lead to a problem if encountered by the application. This defect describes a guaranteed null pointer de-reference which would generate a null pointer exception when the code is executed. We ranked defects by first ranking them independently, and then discussed any differences until an agreement could be made. While we acknowledge that this is an imperfect process, we feel that it is more than sufficient for our oracle and that virtually any ranking system, whether done by tool or manual process, is going to carry at least some subjectivity.

Table 1: Example Ranked Defects

Bug Category	Our Bug Description	Priority
Reliability	Incorrect switch statement	5
Performance	Calls garbage collection explicitly	3
Concurrency	Unsynchronized lazy initialization	3
Correctness	Null pointer de-reference	1

We have made the oracle available for public use on our project website: http://hiddenToKeepAnonymous. .

3.3 Running the Tools

We utilized an open source software online catalog called F-Droid[1] to clone the git repositories of three popular mobile applications: aCal, android-chess, and WordPress. These three applications were chosen because they come from three different domains, greatly vary in size, and were easily imported into the Android Studio IDE.

Each of the three examined tools have different settings which may be altered depending on the desired precision and recall for each tool. In each case, we altered the default settings to create the best precision and recall for each tool. These tool settings are available on our project website.

We next ran each tool on all three applications and extracted the discovered defects upon completion. Each discovered defect was then mapped to the unique defect identifier in the oracle. Table 3 shows a portion of the mapping of each tool's discovered defects to the defects in the oracle. We will next describe any customized settings made to the tools.

[1]http://f-droid.org/

FindBugs

This popular tool checks for potential defects by using different bug patterns to analyze Java bytecode. Similar to the study conducted by Khalid et al. [14], we ran FindBugs using its recommended settings which discovers defects that are "high" and "medium" priority. We ignored "low" priority warnings since they contain a significant amount of false positives and is not recommended for analysis [3]. Since we were examining duplicated binary of the original code, we ignored all style and naming convention warnings.

PMD

This tool looks for potential defects by using different bug patterns to analyzing Java source code. When using PMD, we enabled any detectors turned off by default.

Android Lint

The third tool is a dedicated Android development static analysis tool and is provided to developers as the default static analysis tool packaged with the Android Studio IDE. Like PMD, Android Lint is not as customizable as FindBugs and only allowed us to enable detectors turned off by default.

4. RESULTS

After running the examined tools against our created oracle, we were able to answer our research questions.

RQ1: What categories of defects are found by static analysis tools? After running each tool, the reported defects were analyzed and sorted. In total, fifty unique defects were classified into eight categories: compatibility, concurrency, correctness, maintainability, performance, reliability, security, and usability. Table 2 shows the categorized breakdown of each tool's results.

Table 2: Identified Defects by Category

	FindBugs	PMD	Android Lint
Compatibility	0	0	7
Concurrency	5	0	0
Correctness	9	2	0
Maintainability	0	3	0
Performance	5	2	2
Reliability	1	3	0
Security	8	2	0
Usability	0	1	0
Total	**28**	**13**	**9**

We found that while no tool identified defects in all 8 categories, PMD found defects in 6/8 categories while FindBugs identified defects in 5/8, with Android Lint finding only 2/8. However, FindBugs did discover the most defects with 28 compared to only 13 for PMD and 9 for Android Lint. Based on these results, we can conclude that FindBugs and PMD are both capable of discovering defects in essentially the same number of categories.

FindBugs

FindBugs was able to detect the most high priority potential defects and find the most potential defects in the following categories: concurrency, correctness, and security. Since FindBugs analyzes the compiled Java bytecode, it may not have the ability to detect potential maintainability and usability defects. This is because these categories might require knowledge about how the Java source code

Table 3: Tool Description Mapping

DefectID	Bug Category	Bug Description	Priority	Tool Priority	App(s)	PMD	FB	AL
1	Compatibility	Right-to-left text compatibility issue from API 17 to specified API 14	1	Error	WP			x
2	Maintainability	Empty catch block	5	Critical	WP, aCHess	x		
3	Maintainability	Empty if statement	5	Critical	WP, aChess, aCal	x		
4	Reliability	Switch statement missing break	3	Critical	WP, aChess	x		
5	Security	Reference to array stored, instead of using deep copy	2	Critical	WP, aCal	x	x	

is actually written, which is not accessible to FindBugs.

PMD

PMD found the most performance and reliability potential defects and was the only tool to find potential maintainability and usability defects. Another interesting observation about PMD was that it found more matches to the defect patterns than both FindBugs and Android Lint in all three apps tested; however, the overwhelming majority were not classified as high priority and were deemed out-of-scope for this study. We suspect this is due to PMD analyzing the actual Java source code as written and is reporting potential defects related to coding style.

Android Lint

Android Lint showed strength in detecting potential defects related to Android specific faults, as expected according to the documentation, but was extremely weak in detecting potential defects in the common Java code. 78% of the detected defects were specific to compatibility with previous versions of the Android API, while the remaining 22% were related to garbage collection of allocated memory. Although garbage collection is common to all Java applications and is not specific to Android development, one could argue that development for a mobile environment should be highly concerned with resource usage.

We next addressed our second research question: *RQ2: Which static analysis tool is the most effective at finding defects?* We sought to determine if a single static analysis tool was the most effective at finding defects using the metrics of precision, recall and accuracy.

1. **Precision:** Ratio of the clone pair which a tool reports that are true clones, not false positives.
2. **Recall:** Ratio of the clone pairs in a system that a tool is able to detect.
3. **F-measure:** Considers precision *and* recall to measure the accuracy of a system. It is calculated as $2\times(\frac{precision\times recall}{precision+recall})$. F-measure is often referred as F1 or F-score.

The results in Figure 1 indicate that each tool had a relatively high level of precision with Android Lint leading with 1.0, FindBugs with .969 and Android Lint with .833. There was a wide separation in terms of recall with FindBugs leading with .672, PMD .326 and Android Lint following with .196. FindBugs also led with F-measure having .794, PMD .469, and Android Lint .328.

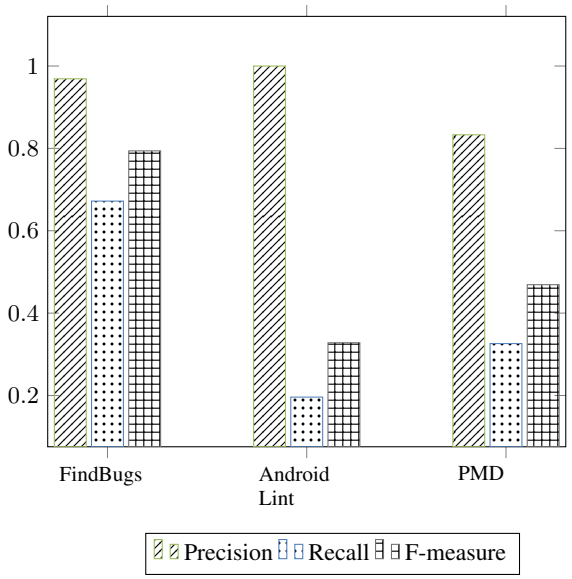

Figure 1: Tool Statistics

Defect ID	Source Location	Tool(s)	Category	Se
1	VComponent.java::287	FindBugs	Correctness	1
	TodoEdit.java::387	FindBugs		
	EventView.java::330	FindBugs		
2	JournalView.java::262	FindBugs	Correctness	1
	TodoView.java::339	FindBugs		

Figure 2: Portion of aCal Oracle from Website

5. PUBLIC DATASET

In order to assist future evaluations of static testing tools on Android apps, we have made our oracle available on our project website in a clearly defined, easy to use manner. Our data is broken up into into two primary groups, defects within each app, and defects at an aggregate level. We present all data at an aggregate level, along with it being separated in groups based upon each app. At the aggregate level, we listed all 50 defects with their bug category, bug description, tool priority, which static analysis tools discovered them, and reported bug descriptions from the tool (if found). We also broke up the discovered defects into groups based upon each app. Figure 2 shows a portion of the noted defects for aCal. We recorded the source defect, category, severity and bug description.

This data may be useful for future researchers in a variety of ways. The oracle may be used as a small benchmark to evaluate new tools in not only terms of precision, recall and F-measure, but also to see what categories and priority levels each tool is capable of discovering. Finally, this represents an unbiased oracle since it

was not created to show the effectiveness of one tool over another. This means that other researchers should feel confident when using it to evaluate their tools.

6. LIMITATIONS & FUTURE WORK

The goal of this study is to provide the research community with an analysis of three leading static testing tools for Android applications. Our results are based on a defect oracle that was constructed by running three different tools on three different applications. The size of our oracle imposes a threat to the values of the metrics to calculate the efficiency of a tool, as a defect oracle constructed from a relatively larger pool of applications could have fine-tuned our result data set.

In this research we only took high priority bugs into consideration based on each tool's configuration. This approach was used due to the large number of defect patterns defined for each tool. In future work, considering all priority levels would give a finer result set with accurate values of precision, recall and F-measure.

In the future, the defect oracle may be expanded beyond the fifty high priority defect patterns, which would allow for more fine-tuned effectiveness statistics. Once a sufficient amount of defect patterns have been prioritized and categorized in the defect oracle, the three static analysis tools should be run against a larger number of mobile applications to verify potential defects are being identified. There are always new testing tools which may be evaluated. Three other leading tools which could be analyzed are MonkeyRunner[2], Ranorex[3], and Appium[4].

Although static analysis tools have demonstrated their value on numerous occasions, it is unreasonable to expect that any tool will ever be flawless and that no static analysis tool is perfect and generally inherently contain limitations [7]. No static analysis detection tools can be expected to perfectly discover every bug in every scenario, and many limitations in FindBugs, and PMD have been noted in previous work [22]. For example, FindBugs is only capable of discovering pre-defined bug patterns, so any defects that have not already been defined for the tool will not be reported [3]. The goal of our work was to merely evaluate and compare the effectiveness of these tools, and was not to make a case either for or against using these tools in Android projects.

7. CONCLUSION

The ability to effectively automatically identify potential software defects in mobile applications using static analysis tools is paramount for improving the overall quality of the app; therefore, improving the likelihood of attracting and retaining users. We analyzed three leading Android static analysis testing tools and compared them in a variety of ways using a new, and publicly accessible defect oracle. While none of the three evaluated tools led in all three categories of precision, recall and F-measure, FindBugs performed at the most consistently high level and led in the categories of Recall and F- Measure in comparison to PMD and Android Lint. PMD and FindBugs were able to discover the most bug categories. We also created a robust, and unbiased public defect oracle which we hope will be used by future researchers.

8. REFERENCES

[1] Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, Dec 1990.

[2] Android lint. http://tools.android.com/tips/lint, 2015.

[3] Findbugs. http://findbugs.sourceforge.net, 2015.

[4] Number of available android applications. http://www.appbrain.com/stats/number-of-android-apps, 2015.

[5] N. Ayewah, D. Hovemeyer, J. Morgenthaler, J. Penix, and W. Pugh. Using static analysis to find bugs. *Software, IEEE*, 25(5):22–29, Sept 2008.

[6] A. Banerjee, S. Chattopadhyay, and A. Roychoudhury. Static analysis driven cache performance testing. In *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th*, pages 319–329. IEEE, 2013.

[7] B. Chess and G. McGraw. Static analysis for security. *IEEE Security & Privacy*, (6):76–79, 2004.

[8] Y. Feng, S. Anand, I. Dillig, and A. Aiken. Apposcopy: Semantics-based detection of android malware through static analysis. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 576–587, New York, NY, USA, 2014. ACM.

[9] http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play store/, 2015.

[10] C. Hu and I. Neamtiu. Automating gui testing for android applications. In *Proceedings of the 6th International Workshop on Automation of Software Test*, AST '11, pages 77–83, New York, NY, USA, 2011. ACM.

[11] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge. Why don't software developers use static analysis tools to find bugs? In *Software Engineering (ICSE), 2013 35th International Conference on*, pages 672–681. IEEE, 2013.

[12] S. Joshi, S. K. Lahiri, and A. Lal. Reducing false alarms for static analysis of concurrent programs, July 29 2014. US Patent 8,793,664.

[13] H. Khalid, M. Nagappan, and A. Hassan. Examining the relationship between findbugs warnings and end user ratings: A case study on 10,000 android apps.

[14] H. Khalid, M. Nagappan, and A. Hassan. Examining the relationship between findbugs warnings and end user ratings: A case study on 10,000 android apps. *Software, IEEE*, PP(99):1–1, 2015.

[15] G. Liang, J. Wang, S. Li, and R. Chang. Patbugs: A pattern-based bug detector for cross-platform mobile applications. In *Mobile Services (MS), 2014 IEEE International Conference on*, pages 84–91, June 2014.

[16] N. Mirzaei, S. Malek, C. S. Păsăreanu, N. Esfahani, and R. Mahmood. Testing android apps through symbolic execution. *SIGSOFT Softw. Eng. Notes*, 37(6):1–5, Nov. 2012.

[17] N. Nagappan and T. Ball. Static analysis tools as early indicators of pre-release defect density. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 580–586, New York, NY, USA, 2005. ACM.

[18] N. Rutar, C. B. Almazan, and J. S. Foster. A comparison of bug finding tools for java. In *Proceedings of the 15th International Symposium on Software Reliability Engineering*, ISSRE '04, pages 245–256, Washington, DC, USA, 2004. IEEE Computer Society.

[19] S. Schmeelk, J. Yang, and A. Aho. Android malware static analysis techniques. In *Proceedings of the 10th Annual Cyber and Information Security Research Conference*, CISR '15, pages 5:1–5:8, New York, NY, USA, 2015. ACM.

[2] http://developer.android.com/tools/help/MonkeyRunner.html

[3] http://www.ranorex.com

[4] http://appium.io

[20] J. Scott. http://www.computerweekly.com/news/2240212085/Android-set-to-reach-one-billion-users-in-2014, 2014.

[21] D. Song, J. Zhao, M. Burke, D. Sbîrlea, D. Wallach, and V. Sarkar. Finding tizen security bugs through whole-system static analysis. *arXiv preprint arXiv:1504.05967*, 2015.

[22] F. Thung, Lucia, D. Lo, L. Jiang, F. Rahman, and P. T. Devanbu. To what extent could we detect field defects? an empirical study of false negatives in static bug finding tools. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ASE 2012, pages 50–59, New York, NY, USA, 2012. ACM.

[23] M. S. Ware and C. J. Fox. Securing java code: Heuristics and an evaluation of static analysis tools. In *Proceedings of the 2008 Workshop on Static Analysis*, SAW '08, pages 12–21, New York, NY, USA, 2008. ACM.

[24] L. Zhang, M. S. Gordon, R. P. Dick, Z. M. Mao, P. Dinda, and L. Yang. Adel: An automatic detector of energy leaks for smartphone applications. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '12, pages 363–372, New York, NY, USA, 2012. ACM.