2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion) | 978-1-6654-1219-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICSE-Companion52605.2021.00049

SetDroid: Detecting User-configurable Setting Issues of Android Apps via Metamorphic Fuzzing

Jingling Sun[§]
Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China
jingling.sun910@gmail.com

***Abstract*—Android, the most popular mobile system, offers a number of app-independent, user-configurable settings (*e.g.*, network, location and permission) for controlling the devices and the apps. However, apps may fail to properly adapt their behaviors when these settings are changed, and thus frustrate users. We name such issues as *setting issues*, which reside in the apps and are induced by the changes of settings. According to our investigation, the majority of setting issues are non-crash (logic) bugs, which however cannot be detected by existing automated app testing techniques due to the lack of test oracles. To this end, we designed and introduced, *setting-wise metamorphic fuzzing*, the first automated testing technique to overcome the oracle problem in detecting setting issues. Our *key insight* is that, in most cases, the app behaviors should keep *consistent* if a given setting is changed and later *properly* restored. We realized this technique as an automated GUI testing tool, SETDROID, and applied it on 26 popular, open-source Android apps. SETDROID successfully found 32 unique, previously-unknown setting issues in these apps. So far, 25 have been confirmed and 17 were already fixed. We further applied SETDROID on 4 commercial apps with billions of monthly active users and successfully detected 15 previously unknown setting issues, all of which have been confirmed and under fixing. The majority of all these bugs (37 out of 47) are non-crash bugs, which cannot be detected by prior testing techniques.**

***Index Terms*—testing, android, setting**

I. MOTIVATION AND CONTRIBUTION

Android provides a number of user-configurable settings. According to the official Android documentation [1], [2], we find that there are 9 major setting categories, *i.e.*, *network and connect*, *location and security*, *sound*, *battery*, *display*, *apps and notifications*, *developer options*, *accessibility*, and *others (including language, theme and time)*. In practice, these settings can be changed via the system app `Settings` on any Android device. The apps on the device are expected to consciously adapt their behaviors to cater for these setting changes, and ensure reliable function at any time.

However, since setting changes are diverse and may happen at any time, achieving the above goal could be challenging. Even well-tested apps may suffer from setting issues. For example, *WordPress* [3] (a popular website and blog content management app with 50 million installations on Google Play and 2,400 stars on GitHub) has such a *critical* issue: when a user turns on the airplane mode in the process of publishing a new blog post, *WordPress* will be stuck at the post uploading status forever, even after the user turns off the airplane mode and connects to the network again [4].

To understand the setting issues in Android apps, we have studied 1,074 setting issues from 180 popular Android apps on GitHub and found that the majority of these issues (759/1,074 $\approx$70.7%) lead to non-crash consequences, *e.g.*, problematic UI display, stuck, and function failure. However, due to the lack of test oracles, existing automated GUI testing tools can hardly uncover these issues [5], [6].

To fill the gap, we leveraged the idea of metamorphic testing and introduced *setting-wise metamorphic fuzzing*, the first automated testing approach to overcome the oracle problem in detecting setting issues for Android apps. We implemented this approach as an automated GUI testing tool, SETDROID, and applied it on 26 popular open-source apps and 4 commercial apps. Finally, it revealed 47 previously-unknown setting issues from these apps. So far, 40 were confirmed and 17 of them were fixed. Most of these bugs (37 out of 47) are non-crash bugs, and cannot be detected by existing testing techniques.

II. RELATED WORK

Android app testing has received much attention [7]–[14]. However, existing generic automated testing tools are limited to crash bugs due to the lack of test oracles and ineffective in detecting setting issues. Prior work [15], [16] however explores limited types of settings and have different research focuses from ours. Sadeghi *et al.* [15] propose PATDROID to detect bugs caused by changing app permissions. Lu *et al.* [16] propose PREFEST to detect bugs caused by the changes of apps' own preferences and some system settings (*i.e.*, WiFi, Bluetooth, mobile data, and location). PATDROID and PREFEST focus on reducing the testing cost due to the combinations of different options but do not consider the impact of setting changes during app usage. Moreover, they can only detect crash bugs, while our work can detect both crash and non-crash bugs.

III. APPROACH AND IMPLEMENTATION

Metamorphic testing [17] is a property-based software testing approach to addressing the test oracle problem. In our scenario, our key observation is that, in most cases, the app behaviors should keep *consistent* if a given setting is changed and later *properly* restored. Otherwise, a likely setting issue happens. For example, an app's function should not be affected if (1) the network is closed but immediately opened; or (2) a specific app permission is revoked but later granted when the app requests that permission again. We leverage this observation as one kind of metamorphic relation to overcome the oracle problem.

[§]This work is conducted under the supervision of Geguang Pu and Ting Su from East China Normal University and supported by NSFC No. 62072178 and the project of STC of Shanghai No. 19511103602.

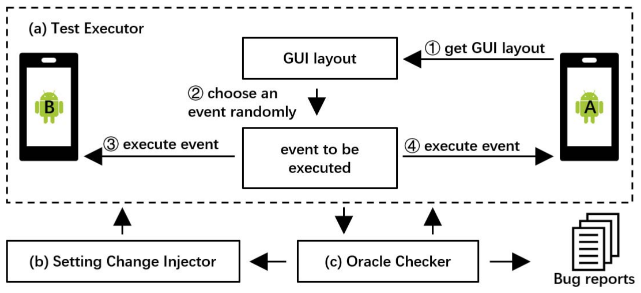

Fig. 1. Workflow of SetDroid to detect setting issues.

Our Approach. We formalize our technique as follows. Let e be a GUI event (*e.g.*, *click*, *edit*, *swipe*, *rotate screen*); let $e.w$ be the GUI widget w that e targets. Let ℓ be a GUI layout (page) of an app which represents a GUI hierarchy tree. Let E be a given seed GUI test which is a sequence of events $E = [e_1, e_2, \ldots, e_n]$. Starting from the initial page ℓ_1 of the app, E can be executed on an app P to obtain a sequence of GUI layouts $L = [\ell_1, \ell_2, \ldots, \ell_{n+1}]$. Specifically, we can view *the execution of* e_i as a function, *i.e.*, $\ell_{i+1} = e_i(\ell_i)$. Then, we inject a pair of events $\langle e_c, e_u \rangle$ into E to obtain a mutant test E', where e_c changes a given setting at random position, while e_u restores the setting. Specifically, we designed two strategies to obtain E' by injecting $\langle e_c, e_u \rangle$ into E:

- *Immediate setting mutation*. We inject e_c followed immediately by e_u. For example, e_c closes the network, and e_u immediately opens the network.
- *Lazy setting mutation*. We inject e_c first, and only inject e_u when it is necessary (*e.g.*, the app prompts an alert dialog). For example, e_c revokes an app permission, and e_u grants the permission only when the app requests that permission.

By comparing the GUI consistency between the seed test E and the mutant test E', we can detect setting issues. Formally, the oracle checking rule is: if there exists one GUI event $e'_i \in E'$ (e'_i corresponds to $e_i \in E$), and its target widget $e'_i.w$ cannot be located on the corresponding layout $\ell'_i \in L'$ (ℓ'_i corresponds to $\ell_i \in L$), then a likely setting issue is found.

$$\exists e'_i.e_i.w \in l_i \wedge e'_i.w \notin l'_i \qquad (1)$$

Tool Implementation. We realized our approach as an automated GUI testing tool SetDroid. Fig. 1 depicts the overview of SetDroid, which contains three main modules: (1) *test executor*, (2) *setting change injector*, and (3) *oracle checker*. We implemented this tool on the *UI Automator* test framework [18], which provides a set of APIs to perform interactions with apps and obtain apps' information such as GUI layouts.

(a) Test Executor. The test executor randomly generates a seed test on device A, and replays the same event sequence (but injected with setting changes) on reference device B. We generate random seed tests because such tests are expected to be much more diverse, practical, and scalable to obtain. SetDroid can also integrate with existing test input generation tools to obtain seed tests.

(b) Setting Change Injector. The two devices A and B were initialized with the default setting environment before testing (*i.e.*, *airplane mode off, Wi-Fi on, mobile data on, location on, battery saving mode off, multi-window off, screen orientation in the landscape, notification on, language is English*). Then, this module randomly injects the setting change event e_c into the test on device B. It now supports these e_c events, *i.e.*, *turn on airplane mode, turn off Wi-Fi, turn off mobile data, turn off location, turn on battery saving, turn on multi-window, rotate the screen, turn off the notification, change to other language*. The corresponding event e_u will be inserted after e_c to restore the corresponding settings as predefined.

(c) Oracle Checker. This module will check whether the oracle checking rule is violated. If the rule is violated, a corresponding bug report will be generated, which records the executed events and GUI screenshots for bug diagnosing.

TABLE I
Setting issues detected by SetDroid

App subjects	26 open-source apps			4 commercial apps	
Bug state	detected	confirmed	fixed	detected	confirmed
#Bugs	32	25	17	15	15

TABLE II
Comparison with existing tools on the 26 open-source apps, C and NC represent crash and non-crash bugs, respectively

Settings	Connect and Location				Permission				Others	
Tool	Prefest		SetDroid		PATDroid		SetDroid		SetDroid	
Consequence	C	NC	C	NC	C	NC	C	NC	C	NC
#Bugs	0	0	0	10	2	0	6	7	3	6

IV. Evaluation

We use the 26 apps from the prior work [16] as our evaluation subjects. We ran SetDroid and the two relevant testing tools PATDroid and Prefest with the same time (12 hours per app). For any found non-crash bug, we manually inspected the bug report and counted the true positives (TP for short) and false positives (FP for short). We replayed each TP bug on real Android devices for validation before reporting on GitHub. In addition, we used SetDroid to test 4 commercial apps from Tecent and ByteDance. They are WeChat [19], QQMail [20], TikTok [21] and CapCut [22], which have billions of monthly active users.

During testing, SetDroid reported 156 errors, 134 of which were TPs (131/156≈83.9%). We analyzed the FPs and found these FPs are caused by specific app features. For example, when the screen orientation setting is changed, *AlwaysOn* will pop up an animation on top of the screen to explain the app function. As shown in Table I, out of the 26 apps, SetDroid found 32 unique and previously unknown setting issues from 24 apps. So far, 25 have been confirmed and 17 were already fixed. We also found 15 setting issues in the 4 commercial apps, all of which have been confirmed and under fixing.

Considering Prefest and PATDroid only cover limited types of settings, we compare the number of bugs they detected in terms of the settings they cover *w.r.t.* SetDroid. As shown in Table II, Prefest did not detect any bug while PATDroid detected two crash bugs related to permission. Most importantly, we can see that SetDroid detected 32 non-crash setting issues from the 26 open-source apps, none of which can be detected by Prefest and PATDroid. Overall, these results clearly show that SetDroid is effective and outperforms existing tools.

References

[1] “Android Developers Documentation,” 2021, retrieved 2021-1 from https://developer.android.com.

[2] “Android Help,” 2021, retrieved 2021-1 from https://support.google.com/android.

[3] “Wordpress,” 2021, retrieved 2021-1 from https://github.com/wordpress-mobile/WordPress-Android.

[4] “Wordpress issue #6026,” 2017, retrieved 2021-1 from https://github.com/wordpress-mobile/WordPress-Android/issues/6026.

[5] L. Fan, T. Su, S. Chen, G. Meng, Y. Liu, L. Xu, G. Pu, and Z. Su, “Large-scale analysis of framework-specific exceptions in android apps,” in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, 2018, pp. 408–419.

[6] T. Su, L. Fan, S. Chen, Y. Liu, L. Xu, G. Pu, and Z. Su, “Why my app crashes understanding and benchmarking framework-specific exceptions of android apps,” *IEEE Transactions on Software Engineering*, 2020.

[7] Y. Li, Z. Yang, Y. Guo, and X. Chen, “Droidbot: a lightweight ui-guided test input generator for android,” in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017, pp. 23–26.

[8] “Android monkey,” 2021, retrieved 2021-1 from https://developer.android.com/studio/test/monkey.

[9] K. Mao, M. Harman, and Y. Jia, “Sapienz: Multi-objective automated testing for android applications,” in *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA)*, 2016, pp. 94–105.

[10] T. Su, G. Meng, Y. Chen, K. Wu, W. Yang, Y. Yao, G. Pu, Y. Liu, and Z. Su, “Guided, stochastic model-based gui testing of android apps,” in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (FSE)*, 2017, pp. 245–256.

[11] T. Gu, C. Sun, X. Ma, C. Cao, C. Xu, Y. Yao, Q. Zhang, J. Lu, and Z. Su, “Practical gui testing of android applications via model abstraction and refinement,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 269–280.

[12] L. Fan, T. Su, S. Chen, G. Meng, Y. Liu, L. Xu, and G. Pu, “Efficiently manifesting asynchronous programming errors in android apps,” in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)*, 2018, pp. 486–497.

[13] Z. Dong, M. Böhme, L. Cojocaru, and A. Roychoudhury, “Time-travel testing of android apps,” in *Proceedings of the 42nd International Conference on Software Engineering (ICSE)*, 2020, pp. 1–12.

[14] W. Guo, L. Shen, T. Su, X. Peng, and W. Xie, “Improving automated GUI exploration of android apps via static dependency analysis,” in *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020, pp. 557–568.

[15] A. Sadeghi, R. Jabbarvand, and S. Malek, “Patdroid: permission-aware gui testing of android,” in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (FSE)*, 2017, pp. 220–232.

[16] Y. Lu, M. Pan, J. Zhai, T. Zhang, and X. Li, “Preference-wise testing for android applications,” in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE)*, 2019, pp. 268–278.

[17] T. Y. Chen, S. C. Cheung, and S. M. Yiu, “Metamorphic testing: a new approach for generating next test cases,” HKUST-CS98-01, Hong Kong University of Science and Technology, Tech. Rep., 1998.

[18] uiautomator2 Team, “uiautomator2,” 2021, retrieved 2021-1 from https://github.com/openatx/uiautomator2.

[19] “Wechat,” 2021, retrieved 2021-1 from https://www.wechat.com.

[20] “Qqmail,” 2021, retrieved 2021-1 from https://en.mail.qq.com.

[21] “Tiktok,” 2021, retrieved 2021-1 from https://www.tiktok.com.

[22] “Capcut,” 2021, retrieved 2021-1 from https://lv.faceueditor.com.