DroidBot: A Lightweight UI-Guided Test Input Generator for Android

Yuanchun Li, Ziyue Yang, Yao Guo, Xiangqun Chen
Key Laboratory of High-Confidence Software Technologies (Ministry of Education)
School of Electronics Engineering and Computer Science, Peking University, Beijing, China
Email: {yuanchun.li, yzydzyx, yaoguo, cherry}@pku.edu.cn

***Abstract*—As many automated test input generation tools for Android need to instrument the system or the app, they cannot be used in some scenarios such as compatibility testing and malware analysis. We introduce DroidBot, a lightweight UI-guided test input generator, which is able to interact with an Android app on almost any device without instrumentation. The key technique behind DroidBot is that it can generate UI-guided test inputs based on a state transition model generated on-the-fly, and allow users to integrate their own strategies or algorithms. DroidBot is lightweight as it does not require app instrumentation, thus no need to worry about the inconsistency between the tested version and the original version. It is compatible to most Android apps, and able to run on almost all Android-based systems, including customized sandboxes and commodity devices. Droidbot is released as an open-source tool on GitHub [1], and the demo video can be found at https://youtu.be/3-aHG_SazMY.**

***Keywords*-Android; dynamic analysis; automated testing; malware detection; compatibility testing;**

根据动态生成的状态转换成ui引导的测试输入

I. Introduction

In recent years, mobile applications (*apps* in short) have seen widespread adoption, with over two million apps available for download in both Google Play and Apple App Store, while billions of downloads have been accumulated.

As there are many apps and many different devices, automating app testing has become an important research direction. In particular, a great deal of research has been focused on automated input generation techniques for Android apps. According to a recent survey [2], most approaches make use of either app instrumentation or system modification in order to get enough information to guide testing.

However, it is unrealistic to instrument an app or the system in some scenarios. For example, in compatibility testing, an app should be tested "as is" on commodity devices in order to find out which device may cause a crash. Another example is malware analysis. As many malicious apps are obfuscated, it might be difficult, even not impossible, to instrument them. Some malicious apps also apply sandbox detection, which might lead to different behaviors on instrumented testing devices and real devices.

This demonstration paper presents DroidBot, a lightweight UI-guided test input generator for Android apps. The design principle of DroidBot is to support model-based test input generation with minimal extra requirements.

DroidBot offers UI-guided input generation based on a state transition model, which is generated on-the-fly at runtime. It then generates UI-guided test inputs based on the transition model. By default the input is generated with a depth-first strategy, which is effective for most cases. Users can also customize the exploration strategy by writing scripts or integrate their own algorithms by extending the event generation modules, making DroidBot a highly extensible tool.

基于状态转换模型的ui引导输入生成

The main reason why DroidBot is more lightweight is that it does not require prior knowledge of unexplored code. Unlike many existing generators which rely on static analysis and instrumentation to get knowledge of unexplored code, DroidBot only models the explored states based on a set of Android built-in testing/debugging utilities. Although this might make DroidBot harder to trigger some specific states, the trade-off enables DroidBot to work with any apps (including the obfuscated/encrypted apps that cannot be instrumented) on almost any customized device (unless the device intentionally removes the built-in testing/debugging modules from the original Android framework, which rarely occurs.).

DroidBot also offers a new way to evaluate the effectiveness of test inputs. Existing approaches mainly use EMMA [3] on open-source apps or instrument apps to calculate test coverage. However, for anti-instrumentation apps (for example verifying the signature at runtime or encrypting the code), it is difficult or even impossible to get their test coverage. DroidBot is able to generate the call stack trace for each test input, which contains the app methods and system methods triggered by the test input. We can use the call stack as an approximate metric to quantify the effectiveness of test inputs.

The source code of DroidBot is available at GitHub [1].

II. Tool Design

The overall architecture of DroidBot is shown in Figure 1. To test an app on a device, DroidBot requires the device being connected via ADB. The device could be an emulator, a commodity device, or a customized sandbox such as TaintDroid [4] and DroidBox [5].

We introduce the *Adapter* module to provide an abstraction of the device and the app under test (AUT). It deals with low-level technical issues such as compatibility with different Android versions and different screen sizes, maintaining connection with the device, sending commands to the device and processing command outputs, etc.

The *Adapter* also acts as a bridge between the test environment and the test algorithm. On one hand, it monitors the state

978-1-5386-1589-8/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE-C.2017.8

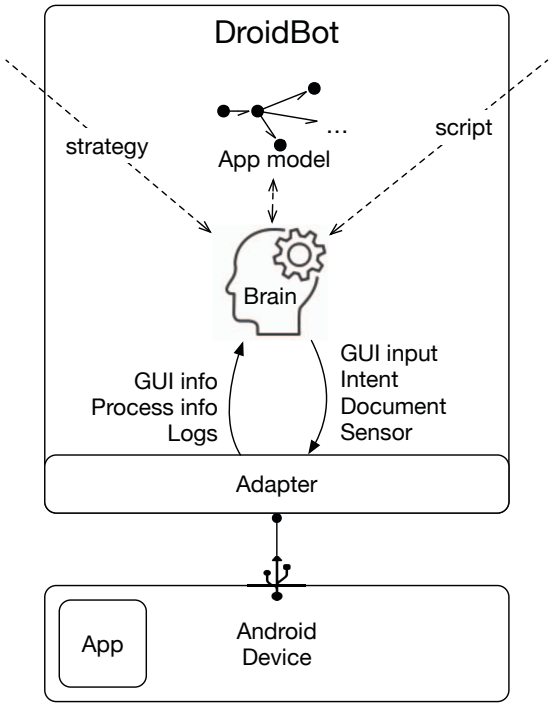


Fig. 1. DroidBot Overview.

of the device and AUT and converts the state information to structured data. On the other hand, it receives the test inputs generated by the algorithm and translates them to commands. With the *Adapter*, DroidBot is able to provide a set of easy-to-use high-level APIs for users to write algorithms while ensuring that the algorithms work in different test environments.

The *Brain* module receives device and app information produced by the *Adapter* at run-time, and sends generated test inputs to the *Adapter*. Test input generation is based on a state transition graph constructed on the fly. Each node of the graph represents a device state, while the edge between each pair of nodes represents the test input that triggered the state transition. DroidBot integrates a simple but effective depth-first exploration algorithm to generate test inputs. It also allows users to integrate their own algorithms or use app-specific scripts to improve the test strategy.

Such design improves the usability of DroidBot. Table I shows the usability comparisons between DroidBot and other public-available test input generation tools. We can see that DroidBot requires as little requirements as Monkey, while providing much more extensible features comparable to other tools requiring instrumentation.

III. Implementation

A. Lightweight Monitor and Input

DroidBot fetches device/app information from the device and sends test inputs to the device through ADB. Both the monitoring and input phases are lightweight because they are mostly based on existing Android debugging/testing utilities, which are available on most Android devices.

TABLE I
USABILITY COMPARISON OF EXISTING PUBLICLY-AVAILABLE BLACK-BOX TEST INPUT GENERATORS. NOTE THAT SOME DATA IS FROM CHOUDHARY *et al.* [2].

Tool	Instrumentation		Strategy	Programmable
	System	App		
Monkey [6]	✗	✗	Random	✗
AndroidRipper [7]	✗	✓	Model	✗
DynoDroid [8]	✓	✓	Random	✗
SwiftHand [9]	✗	✓	Model	✗
PUMA [10]	✗	✓	Model	✓
DroidMate [11]	✗	✓	Model	✓
DroidBot [1]	✗	✗	Model	✓

The information fetched from the device can be categorized into three sets:

1) **GUI information**. For each UI, DroidBot records the screenshot and the UI hierarchy tree dumped using UI Automator (for SDK version higher than 16) or Hierarchy Viewer (for lower versions);
2) **Process information**. DroidBot monitors system-level process status using the `ps` command and app-level process status using the `dumpsys` tool in Android.
3) **Logs**. Logs include the method trace triggered by each test input and the logs produced by the app. They can be retrieved from the Android profiling tool and *logcat*.

The test input types supported by DroidBot include UI inputs (such as touching, scrolling, etc.), intents (BOOT_COMPLETED broadcast, etc.), documents to upload (image, txt, etc.) and sensor data (GPS signal etc.). Note that the sensor simulation is only supported by emulation.

DroidBot provides a list of easy-to-use APIs for fetching information from the device and sending inputs to the device. For example, developers can simply call `device.dump_views()` to get a list of UI views and call `view.touch()` to send a touch input to a view.

B. On-the-fly Model Construction

DroidBot generates a model of AUT based on the information monitored at runtime. The model aims to help input generation algorithms to make better test input choices.

Figure 2 shows an example of a state transition model. Basically, the model is a directed graph, in which each node represents a device state, and each edge between two nodes represents the test input event that triggered the state transition. A state node typically contains the GUI information and the running process information, and an event edge contains the details of the test input and the methods/logs triggered by the input.

The state transition graph is constructed on the fly. DroidBot maintains the information of the current state, and monitors the state changes after sending a test input to the device. Once the device state is changed, it adds the test input and the new state to the graph, as a new edge and a new node.

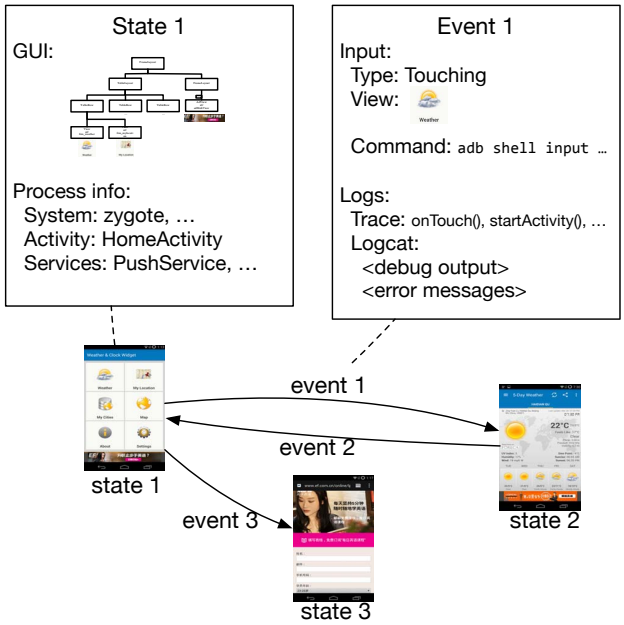


Fig. 2. An example of state transition graph. Note that the data in this graph is simplified for easy understanding.

The graph construction process relies on the underlying state comparison algorithm. Currently, DroidBot uses content-based comparison, where two states with different UI contents are considered as different nodes.

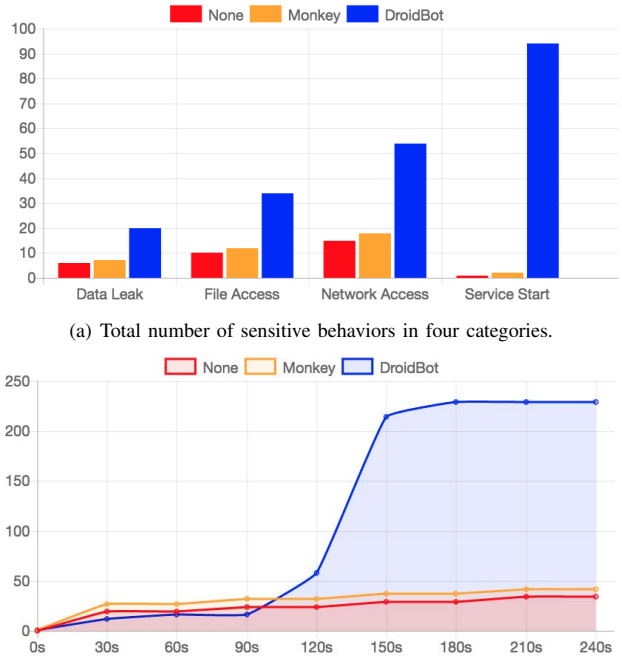
C. Quantifying the Effectiveness of Test Input

One problem faced by researchers and testers when conducting black-box testing is the difficulty to evaluate testing effectiveness, as the existing test coverage methods either require the source code of AUT [3] or need to instrument the AUT [12].

DroidBot integrates two methods to quantify the test effectiveness without source code or instrumentation:

- *Method tracing.* DroidBot is able to print the method trace of each test input using the Android official profiling tool. The method trace contains the app methods and system methods triggered by the test input. With the method trace, we are also able to calculate the method coverage if the total number of methods is available.
- *Sensitive behavior monitoring.* For malware analysis, the number of sensitive behaviors triggered can reflect the test effectiveness. For example, DroidBot can be used with DroidBox [5] to monitor the sensitive behaviors triggered by each input.

The *method tracing* mechanism scales better as it works with almost any device and any app, while the *sensitive behavior monitoring* mechanism requires apps running in a certain sandbox. However, the number of sensitive behaviors might be more intuitive in malware analysis. Both methods are unable to give a normalized value of how effective a test case exactly is, but they can provide meaningful statistics when comparing different test cases on the same app.



(a) Total number of sensitive behaviors in four categories.

(b) Speed of triggering sensitive behaviors.

Fig. 3. Comparison of the effectiveness in triggering sensitive behaviors when testing a malware with Monkey and DroidBot.

IV. Usage Scenarios

A. Compatibility Analysis

One of the useful scenarios of DroidBot is compatibility testing, which is aimed at evaluating the app's correctness and robustness when running on different devices. Compatibility testing should be performed on many different commodity devices thus system instrumentation is unrealistic. Meanwhile, app instrumentation might also be unwanted because instrumented app may behave differently from the original app.

With DroidBot, a developer is able to test his/her app on different devices without instrumentation, reaching more UI states in much shorter time compared to Monkey. Moreover, with the scripting feature provided by DroidBot, the developer can customize the test input to generate.

B. Malware Analysis

Malware analysis is also a useful scenario of DroidBot. As many malware encrypt their code or check their signature before doing malicious things, it might be impossible to instrument them or guarantee the consistency between the instrumented app and the original app.

Monkey [6] is able to test malware without instrumentation, but the random strategy of Monkey might not be efficient in discovering the malicious behaviors. DroidBot is as easy-to-use as Monkey but is better in app exploration as it uses a model-based strategy. For example, if a malware does not perform malicious behavior until the user clicks certain buttons, it might be difficult for randomized test input generator to find the correct buttons, while model-based generator have the

information about the AUT fetched from the device at runtime, thus is easier to trigger the sensitive behaviors.

Figure 3 shows the comparison to Monkey in a proof-of-concept example of using DroidBot in malware analysis. We selected a malware which encrypted its code as the app under test, and used DroidBox [5] as the testing device in order to monitor the sensitive behaviors, such as file accesses, network accesses, data leaks, etc.

We use Monkey and DroidBot to generate test input respectively. The result shows that the amount of sensitive behaviors triggered by DroidBot is much higher than Monkey, while the inputs generated by Monkey almost did not trigger any extra sensitive behaviors. We inspected the test processes of Monkey and DroidBot. The reason for Monkey's ineffectiveness is that the app requires users to touch two buttons in a pop-up dialog successively to enter a malicious state. DroidBot successfully found the buttons and touched them in around 80 seconds, while the randomized test inputs generated by Monkey failed to pass the pop-up dialog.

V. Related Work

Test input generation for Android has been drawing researchers' interests for a long time.

Monkey [6] is the most popular tool to perform black box testing, and it is the most light-weighted. However, the inputs generated by Monkey are completely random, which is not extensible and easy to be intentionally bypassed. DynoDroid [8] also generates randomized input, but it is smarter in selecting test inputs.

AndroidRipper [7], SwiftHand [9], A^3E [13] and GUICC [14] are model-based automated test generators, while using different methods to construct the model and generate input based on the model. PUMA [10] is a model-based test framework, which is programmable with PUMAScript. SmartDroid [15] and Brahmastra [16] are focused on targeted testing which aims to trigger certain pieces of code.

Andlantis [17] is designed for malware analysis. It is focused on large-scale virtual machine management and able to execute malware on multiple emulators at the same time.

DroidMate [11] is a similar approach to DroidBot as it also emphasizes robustness and extensible strategy, however it still needs a slight instrumentation to enable API monitoring.

Compared to these tools, DroidBot is as easy to use as Monkey, while providing much advanced features as most other tools, including model-based input generation and extensible scripting, etc.

VI. Conclusion

This demonstration presents DroidBot, a lightweight test input generator for Android apps. DroidBot is able to test an Android app on almost any device with minor environment requirements. It is easy to use, because on one hand, it is extensible based on a set of high-level APIs and a state transition model constructed on the fly, on the other hand, it provides a set of utilities to evaluate the test effectiveness. Besides regular testing tasks, DroidBot can also be used in scenarios including compatibility testing, malware analysis and other cases where instrumentation is unwanted.

Acknowledgment

This work is partly supported by the National Key Research and Development Program under Grant No.2016YFB1000105 and the National Natural Science Foundation of China under Grant No.61421091.

References

[1] honeynet, "Droidbot: A lightweight test input generator for android," https://github.com/honeynet/droidbot, 2016, accessed: 2016-11-10.

[2] S. R. Choudhary, A. Gorla, and A. Orso, "Automated test input generation for android: Are we there yet? (e)," in *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ser. ASE '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 429–440.

[3] V. Roubtsov, "Emma: a free java code coverage tool," 2006.

[4] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10, 2010, pp. 393–407.

[5] A. Desnos and P. Lantz, "Droidbox: An android application sandbox for dynamic analysis," 2011.

[6] A. Developers, "Ui/application exerciser monkey," 2012.

[7] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon, "Using gui ripping for automated testing of android applications," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2012, 2012, pp. 258–261.

[8] A. Machiry, R. Tahiliani, and M. Naik, "Dynodroid: An input generation system for android apps," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013, 2013, pp. 224–234.

[9] W. Choi, G. Necula, and K. Sen, "Guided gui testing of android apps with minimal restart and approximate learning," in *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, ser. OOPSLA '13, 2013, pp. 623–640.

[10] S. Hao, B. Liu, S. Nath, W. G. Halfond, and R. Govindan, "Puma: Programmable ui-automation for large-scale dynamic analysis of mobile apps," in *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '14, 2014, pp. 204–217.

[11] K. Jamrozik and A. Zeller, "Droidmate: A robust and extensible test generator for android," in *Proceedings of the International Conference on Mobile Software Engineering and Systems*, ser. MOBILESoft '16, 2016, pp. 293–294.

[12] ylimit, "androcov: measure test coverage without source code," https://github.com/ylimit/androcov, 2016, accessed: 2016-11-10.

[13] T. Azim and I. Neamtiu, "Targeted and depth-first exploration for systematic testing of android apps," in *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, ser. OOPSLA '13, 2013, pp. 641–660.

[14] Y.-M. Baek and D.-H. Bae, "Automated model-based android gui testing using multi-level gui comparison criteria," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2016, 2016, pp. 238–249.

[15] C. Zheng, S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, and W. Zou, "Smartdroid: An automatic system for revealing ui-based trigger conditions in android applications," in *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, ser. SPSM '12, 2012, pp. 93–104.

[16] R. Bhoraskar, S. Han, J. Jeon, T. Azim, S. Chen, J. Jung, S. Nath, R. Wang, and D. Wetherall, "Brahmastra: Driving apps to test the security of third-party components," in *Proceedings of the 23rd USENIX Conference on Security Symposium*, ser. SEC'14, 2014, pp. 1021–1036.

[17] M. Bierma, E. Gustafson, J. Erickson, D. Fritz, and Y. R. Choe, "Andlantis: large-scale android dynamic analysis," *arXiv preprint arXiv:1410.7751*, 2014.