ABSTRACT
Over the past few years, researchers have proposed various program analysis tools for automated detection of event-race conditions in Android. However, to this date, it is not clear how these tools compare to one another, as they have been evaluated on arbitrary, disjointed set of Android apps, for which there is no ground truth, i.e., verified set of event races. To fill this gap and support future research in this area, we introduce BenchERoid, a set of 34 Android apps with injected event-race bugs. The current version of benchmark contains 36 types of event-race bugs that were identified by analyzing Android concurrency literature and publicly available issue repositories. We believe that our framework is a valuable resource for both developers and researchers interested in concurrency bug analysis in Android. BenchERoid is publicly available at: https://github.com/seal-hub/bencheroid.

CCS CONCEPTS
• Software and its engineering → Software testing and debugging.

KEYWORDS
Event-Race, Benchmark, Android, Concurrency

Permission to make digital or hard copies of part or all 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. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

MobiSys ’20, June 15–19, 2020, Toronto, ON, Canada
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7954-0/20/06.
https://doi.org/10.1145/3386901.3396602

1 INTRODUCTION
Concurrency mechanisms in Android are an important source of confusion for developers and contribute to many software bugs in Android apps [6]. The Android platform provides a hybrid concurrency mechanism supporting both traditional multi-threaded and event-driven programming paradigms [2]. An app defines many callback methods to handle events originating from the user (e.g., tap on the screen), the system (e.g., notifications of location change), other apps (e.g., requesting data), or the app itself (e.g., downloading a file in a background thread). In addition, Android provides a variety of concurrency libraries for use by developers, e.g., Looper or AsyncTask.

Among the various concurrency bugs, event race is not only the most frequently encountered, but also the most challenging to detect in Android apps [3, 5, 6]. An event race occurs when two event handlers access the same memory location, where at least one of the accesses is a write operation, and there exists no specific order between their executions; therefore, the final value contained in the memory location can change depending upon the stochastic execution order of events [3].

Over the past few years, researchers have proposed various program analysis tools for automated detection of event-race conditions in Android [3, 5, 6]. Despite these efforts, not much is known as to their relative strengths and weaknesses. This is largely due to the fact that the evaluation of these tools have been performed in an ad hoc fashion. The subject apps used for the evaluation of these tools are neither available (the source code and bug locations are not provided) nor representative (the authors randomly chose a handful of apps without specifying any selection criterion). Research in this area is hindered by the lack of a benchmark for which the ground truth is known.

Exiting concurrency benchmarks for Java [4, 8] are not suitable for Android due to the differences between the platforms. For instance, in Android, event races can occur even within a single thread [3], since the events may execute without any specific order, while in Java event races can only occur in scenarios involving multiple threads. Figure 1 shows an event-race example in Android. When the Activity is created, it registers a BroadcastReceiver (line 6). The BroadcastReceiver updates a widget, textView, when it receives a message (line 4). The widget is initialized in onStart and assigned a null value when the activity goes to the background in onStop (lines 7-8). Here, if an intent is sent to the receiver when the app is in the background, a null pointer exception is thrown, forcing a crash. This event race is caused by the non-deterministic execution order between onReceive and onStop even though both are running on the main thread.

While the research community has developed popular benchmarks for various quality concerns, such as security vulnerabilities [1] and energy bugs [7], surprisingly none exists for event-race issues in Android. To fill this gap, we introduce BenchERoid, a first of its kind, publicly available dataset of Android apps containing a variety of event races (https://github.com/seal-hub/bencheroid). We meticulously reviewed real world apps and recent publications to identify entities that contribute to the manifestation of event-race bugs in Android apps. Next, we handcrafted 34 Android apps harboring examples of such concurrency bugs involving a variety of entities. Finally, we built and ran all the apps, and validated the existence of event races. The validated event races provide the ground truth for evaluating the accuracy of event-race detection tools and advancing the research in this area.

Poster: A Benchmark for Event-Race Analysis in Android Apps

Navid Salehnamadi, Abdulaziz Alshayban, Iftekhar Ahmed, and Sam Malek
School of Information and Computer Sciences
University of California, Irvine, USA
{nsalehna,aalshayb,iftekha,malek}@uci.edu

Poster: A Benchmark for Event-Race Analysis in Android Apps

Navid Salehnamadi, Abdulaziz Alshayban, Iftekhar Ahmed, and Sam Malek
School of Information and Computer Sciences
University of California, Irvine, USA
{nsalehna,aalshayb,iftekha,malek}@uci.edu

Poster: A Benchmark for Event-Race Analysis in Android Apps

Navid Salehnamadi, Abdulaziz Alshayban, Iftekhar Ahmed, and Sam Malek
School of Information and Computer Sciences
University of California, Irvine, USA
{nsalehna,aalshayb,iftekha,malek}@uci.edu
1. **Concurrency API**. An event race is caused by concurrent invocations of instructions handled by a concurrency library, e.g., `AsyncTask`.

2. **Synchronization Mechanism**. The concurrency behavior of an app can be managed using synchronization mechanisms, e.g., `wait` in `Thread`.

3. **Involved Components**. Android components, e.g., `Activity`, are created and managed asynchronously by the framework.

4. **Android Callbacks**. An Android app may override some predefined Android callbacks which can be `Life Cycle` callbacks of components, e.g., `onCreate`, or `Event` callbacks, e.g., `onClick`.

5. **Inter-Component Communication**. Different components can communicate with each other concurrently by sending intent messages.

6. **Input**. The execution of events depends on the provided input. For example, an event race may occur only when specific inputs match the condition of an `if`.

7. **Execution Order**. The order of execution of statements may cause (or prevent) event races.

8. **Thread**. Events are executed on threads with various properties, e.g., a Looper handles its messages sequentially. These properties may introduce or prevent an event race.

9. **Time-sensitivity**. The order of execution of events may be dependent on the time of their execution.

For each entity, we identified a set of possible values (Table 1) by analyzing the literature. We created combinations of these values to generate meaningful event-race scenarios. For example, an event race can occur while using `TimerTask` concurrency API that depends on the time of execution. In total, we created 34 apps for 568 pairwise combinations that cover a wide variety of event-race scenarios. We further added some infeasible event races along with true ones for evaluating the precision of event-race detectors. All apps have been developed in Android Studio 3.5 and can be built automatically using Gradle. The detailed list of apps and event races are provided as part of the companion website.

### 3 CONCLUSION

In this paper, we presented BenchERoid, a micro benchmark for supporting the analysis of event-race conditions in Android apps. The benchmark includes 34 apps containing 36 types of concurrency bugs. We release the benchmark under the Apache License (Version 2.0) and make it publicly available on Github. Our goal is to provide the community with the means to support the research on detection of concurrency bugs in Android and BenchERoid was the first step towards fulfilling that goal. Our future work involves comparative analysis of existing tools using BenchERoid.

### REFERENCES


