README for DroidFax Artifacts

❖ Contents

Content of this artifact package

The dropbox artifact folder we shared is named ***droidfaxAE***, it includes two files: **artifact.zip** and **VBox.zip**. The first includes the artifacts we intent to share publicly, and the second file holds a Virtual Box VM that is used for facilitating the use/replication of our artifacts.

Once the **artifact.zip** file is downloaded, unpack it. Then the following structure should be seen:

- code
  - build.xml
  - src
    - covTracker
    - dynCG
    - eventTracker
    - intentTracker
    - reporters
    - utils
  - scripts
    - *.sh
    - processResult
    - apkmng
- data
  - benchmarks
    - used-benign-apps-droidfax.txt
    - app-pair-statistics.html
  - GeneralReport
  - ICCReport
  - interAppICCReport
  - SecurityReport
  - produceall.sh, cleanall.sh
- web

    - page_usage.html
    - page_resultformat.html
    - metricdef.pdf
- README.PDF, README.DOC, README.txt (same content in different formats)

Then, download the **droidfax.zip** file, unzip it, the following can be seen

- VBox
    - droidfaxAE.vbox
    - droidfax.vdi

The **code** and **data** folders include the analysis code and our dataset, respectively, as already described in detail in the one-page artifact paper. In particular, corresponding to what is described at the end of Section 2, scripts under **code/scripts** are those used for running the toolkit's three phases, scripts under **code/scripts/processResult** are those used for processing the raw study results into graphical/tabular formats used in the research paper, and finally the scripts under **code/scripts/apkmng** are those used for accessing (e.g., downloading, uninstalling/installing, and querying) APKs.

The **web** folder contains the project website only in case the web may not be accessible at the time of artifact evaluation. Corresponding to what are mentioned in the artifact paper:

- The *page_usage.html* file therein first gives usage instructions of the study toolkit DroidFax, followed by explanation of each R script used for processing raw characterization results into those used in the research paper.
- The *page_resultformat.html* file explains each raw characterization result data file.
- The *metricdef.pdf* elaborates on the definition of each of the 122 metrics used in our characterization day.

Finally, the VBox folder includes the VM for VirtualBox that we created to ease the artifact evaluation, especially for exploring our study and reproducing *part of* the results presented in our research paper. Specifically, two files are included in this folder: the *droidfaxAE.vbox* file is the VirtualBox configuration/project file, and the *droidfax.vdi* file is the virtual disk image that actually holds the entire VM OS.

To load the VM (called *droidfaxAE*), please first install VirtualBox 5.1.22 from https://www.virtualbox.org/wiki/Downloads. Then either open the *droidfaxAE.vbox* file with VirtualBox or create a new Linux 64bit VM by using the VDI image *droidfax.vdi* instead of creating a new virtual hard disk. (Please make sure that your machine itself supports VM running a 64bit OS; enabling this support sometimes needs to change BIOS options.)

We have verified the workings of this VM on Ubuntu 16.04 LTS (the VM OS itself is also Ubuntu 16.04 LTS). We set 8GB RAM for this VM during our test. Once installed, launch the VM. If logon is asked (we have set auto logon on the VM), use the following credentials:

Username: **osboxes** (or an alias named **hcai**)
Password: **droidfax**

The next two sections assume that you have launched and logged in to the VM.

Explore the artifact

First, open the terminal, then explore as follows (for convenience, you can always use "tree *dirname"* to explore a directory *dirname*'s structure; the tree tool has been installed).

Most of the following steps concern only ***browsing*** original inputs to the study, intermediate files produced, raw study results, and final results reported in research paper. We only provide instructions to partially reproduce the study (see next Section of this document for partial reproduction) due to the total time cost of the study. Our study needs to instrument 122 individual apps in 62 pairs; more critically, it needs to run each pair and each individual app for an hour and then to repeat this process for three times (in total, the profiling phase took at least 552 hours). The trace storage space was over 65GB.

1. Check the study toolkit

   The source code can be found under **/home/hcai/workspace/droidfax**, where the content is the same as **code/src/** above. All dependence libraries are located at **/home/hcai/libs**. To rebuild the code:

   cd /home/hcai/workspace/droidfax
   ant clean build

   The experimentation testbed can be found under **/home/hcai/testbed**/: the scripts in this directory are the same as described above, except for the scripts for accessing APKs in **code/scripts/apkmng** are placed in **/home/hcai/bin** for usage convenience**.**

2. Check the study dataset

   The app-pair benchmark suite is in **testbed/input/pairs**. There are 62 pairs in total, 7 communicating through explicit ICCs and 55 through implicit ICCs. We used these pairs in our study for characterizing inter-app ICCs. All other characterization steps only need and use individual apps (all from the pairs we used).

   A list of individual app benchmarks' package names is located under **/home/hcai/bin** (used-benign-apps-droidfax.txt). We did not repeatedly place the APKs on this VM in order to reduce the total VM size. For the same reason, the instrumented benchmarks and the execution traces are not placed either because they are very large in size. We used **instrAllPairs.sh** (usage demonstrated in **launchdemo.sh**) to instrument these benchmarks.

   The raw study results are located under **testbed/results**, where the folders hold the raw results for the three characterization dimensions (General/Structure, ICC, Security). The folder names are those of the dimensions suffixed with 'Report'; for ICC results, an additional folder called interAppICCReport holds the results related to inter-app ICC metrics. Under each folder, the **.txt** files are raw characterization results and the R and Python scripts are used for visualizing and/or tabulating the results, outcome of which was used for the research paper. These files and scripts are explained on the project website as mentioned above. Usage of the scripts is demonstrated in **testbed/results/produceall.sh**.

To reproduce the final results used in the paper from the raw data, do:
cd /home/hcai/testbed/results
bash produceall.sh

The output of this script indicates what each final result is and where it is located (tabular results are immediately shown on the screen; figures are pointed to their locations). The script cleanall.sh is used to clean all final results (figures in PDF) so that you can reproduce again.

The following table provides a mapping between the generated results and figure/table used in the research paper.

Under **testbed/results**	In the research paper
generalReport/gdist-inds-d-horiz.pdf	Figure 2
generalReport/edgeFreq-scatter.pdf	Figure 3
Cross-layer total call distribution table on screen	Figure 4
securityReport/callback-dins-horiz.pdf	Figure 5
Lifecycle callback categorization ranking table on screen	Table I
Event-handler callback categorization ranking table on screen	Table II
generalReport/compdist-ins-d-horiz.pdf	Figure 6
ICCReport/gicc-iccboth.pdf	Figure 7
ICCReport/gicc-databoth.pdf	Figure 8
securityReport/srcsink-dins-horiz.pdf	Figure 9
Source categorization ranking table on screen	Table III
Sink categorization ranking table on screen	Table IV

There are additional raw data in the folders, and additional figures may be generated out there. Those are results not reported in our paper, but are what our study toolkit can produce to be used in extended/future studies.

Reproduce the artifact (partially)

As explained above, fully reproducing all the results presented in the research paper would not be feasible with respect to the time for this artifact evaluation. However, to validate that the study toolkit actually worked to enable our full study, we prepared a way for partial reproduction.
The reproduction being partial means: reproducing our study on a smaller dataset but covering the complete study process (i.e., from prepared benchmarks to final characterization results).The only step not covered is benchmark download and selection (according to coverage and other criterion).

Specifically, the smaller dataset includes one pair from the 7 explicit-ICC-linked ones, and two pairs from the 55 implicit-ICC-linked ones, as randomly chosen. These three pairs correspond to pairs no. 61, no. 3, and no. 24 in **data/benchmarks/app-pair-statistics.html**. This benchmark subset is located at **testbed/demoinput**.

To reproduce, do

cd /home/hcai/testbed
bash lauchDemo.sh

This will actually run the full study process, albeit at a small scale (on a small benchmark suite). The three phases of our characterization (as depicted in Figure 1 of the research paper) will be carried out one by one, followed by final step on results visualization/tabulation. The **testbed/log.demo** file shows what you should see on screen with a successful run of this all-in-one demo script. To save time, the default profiling time was set to one minute only, and the profiling phase was only performed once (versus one hour and three repetitions in our original study). For this reason, the results may not be representative of what the research paper presented.

Use other tools

Our toolkit includes two additional standalone tools: a coverage tracker and an event tracker, as described in the one-page artifact paper. The former has been used during our benchmark selection, and the latter has not been used in the study in relation to our research paper. However, we shared these tools as part of the artifact for the reasons described in the artifact paper. We prepared two demos to show the usage of these two tools.

To see a demo of the coverage tracker, do
cd /home/hcai/testbed
bash covTrackerDemo.sh

To see a demo of the event tracer/profiler, do
cd /home/hcai/testbed
bash eventTrackerDemo.sh

Without arguments to the demo launch scripts (covTrackerDemo.sh and eventTrackerDemoe.sh), the demo will be running in 'auto' mode, meaning that a preselected app (testbed/adhoc/101.apk) will be used for the demo, and automated generated inputs from Monkey will feed the app for one minute during execution; and at the end of the demo, the screen will show the coverage/event trace.

To run the demo in a manual mode, just append "manual" to the command line (e.g., *bash eventTrackerDemo.sh manual*); then Monkey will not feed inputs, and the screen will prompt you to manually launch the app and manipulate it. As you manipulate the app, the screen will show rolling coverage/event trace.

The second optional argument gives the apk of a different app, and the third gives the time for Monkey to exercise the app in the auto mode.

The format of the coverage trace is self-evident. The event trace consists of lines each in the format of '*[Event] [the category of an event] [the callback method that handles the event]*'

Known issues

It is known that the Android emulator is best run on Intel-x86 architecture that has hypervisor support (kvm). However, the supervisor support is rarely available on a VM such as ours created using and running on Virtual Box.

As a makeshift, we chose to run the study in the VM on an Android AVD based on armeabi-v7a without gpu support. In consequent, the emulator runs much slower than running on a physical machine. In addition, our benchmarks may behave differently on an Intel machine (where we conducted the original study) than on the ARM machine (where we show the artifact through the VM). Thus, the performance issues and results variance can occur. However, this should only affect the partial result reproduction which involves app executions. Producing the final results from raw study results (by running */home/hcai/results/produceall.sh*) should not be affected.

Appendix: use the artifact on a different machine

Our artifacts have been tested with the following environment settings and tools/libraries (i.e., the configurations on a clean machine):

- Ubuntu 16.04 LTS
- Java 1.8.0_131 (java-1.8.0-openjdk-adm64)
- R for Linux version: 3.4.0
- Python 2.7
- Tree 1.7.0 (the tree tool for browsing directories)
- Android SDK (with API 19, including tools liked adb, emulator, monkey, etc.)
- Android AVD (Nexus-One-10 with 1G SD card, 200 RAM, and intel-x86 64 system image)
- expect (year 1994 version)
- aapt (year 2014 version)
- Apktool v2.0.2
- ant v1.9.6
- vim 7.4 (optional, for editing scripts and viewing textual results)
- googleplay-api and its dependee library python-protobuf (optional, for downloading apps)
- keytool (year 2015 version) and zipalign (year 2014 version) (used for signing an apk after instrumentation; without signing, the APK won't be installed successfully.)
    - you also need to create a certificate for signing the APKs, see https://developer.android.com/studio/publish/app-signing.html#sign-manually
    - after creating the certificate using keytool, set the keystore location accordingly (in testbed/signandalign.sh)

After successfully performing all the above steps, download the entire package here http://chapering.github.io/droidfax.tar.gz

Then, unpack it and follow the README inside.