ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

USENIX Security '25 Artifact Appendix: Hercules Droidot and the murder on the JNI Express

Luca Di Bartolomeo[*]
EPFL

Philipp Mao[*]
EPFL

Yu-Jye Tung
UC Irvine

Jessy Ayala
UC Irvine

Samuele Doria
University of Padua

Paolo Celada
EPFL

Marcel Busch
EPFL

Joshua Garcia
UC Irvine

Eleonora Losiouk
University of Padua

Mathias Payer
EPFL

A Artifact Appendix

A.1 Abstract

Poirot's artifact contains the source code necessary to run our Android native library fuzzer and static analysis passes. This document describes how to set up our prototype, and gives a brief overview of the resource requirements to replicate some of the experiments conducted in our evaluation, along with instructions to run them.

A.2 Description & Requirements

The artifact contains Poirot's source code and scripts to reproduce the evaluation.

A.2.1 Security, privacy, and ethical concerns

This artifact does not contain any threat to the system's integrity or privacy. However, we still recommend running it inside a sandboxed environment (either a VM or a container).

A.2.2 How to access

Poirot is accessible at `https://doi.org/10.5281/zenodo.15586319` or `https://github.com/HexHive/droidot.git`.

A.2.3 Hardware dependencies

Our evaluation requires Android emulators to run the mobile applications under test. Modern Android emulators require ARM64 architecture for optimal performance and compatibility with contemporary Android versions.

To accommodate reviewers during the artifact evaluation process, we provide access to two dedicated servers. The first one is an x64 server, `ssh -p 1234 ae@65.108.89.50`, which is used to run the static analysis tools for the first major claim of our paper (C1).

The second one is an ARM64 server, `ssh -p 1235 ae@65.108.89.50`, which is used to run the fuzzing campaigns for the second major claim of our paper (C2) and to reproduce the case study crash for the third major claim of our paper (C3).

This server enables reviewers to reproduce our experimental results without requiring them to have ARM64 hardware locally. However, we note that those servers have more limited computational resources compared to the AWS cluster infrastructure used in our original paper evaluation.

The difference in computational resources between the evaluation servers and our original experimental setup should not affect any of the major claims made in our paper.

A.2.4 Software dependencies

All software dependencies are managed through our Docker container.

The Docker container ensures reproducible builds and eliminates dependency conflicts across different host systems. No additional software installation is required on the host machine beyond Docker itself.

A.2.5 Benchmarks

As outlined in the paper, there are no clear benchmarks defined by literature to evaluate Android native library fuzzers. Therefore, included in the artifact we ship the most 100 popular Android applications. They can be found in the `target_APK` folder.

Note: we kindly ask the reviewers to not redistribute the APKs, as they are sourced from Androzoo and are subject to their terms of use.

A.3 Set-up

A.3.1 Installation

To access the artifact servers, please provide your SSH public key in the HotCRP comments.

ARM64 Server for Fuzzing Campaigns:

1. Login to the server (`ssh -p 1235 ae@65.108.89.50`).

2. Clone the POIROT repository: `git clone https://github.com/HexHive/droidot.git`.
3. Change to the repository directory: `cd droidot`.
4. Run `./setup.sh` to download dependencies and build the Docker container.
5. Spawn a shell in the container: `./run.sh`.

x86 Server for Static Analysis Tools:

1. Login to the server (`ssh -p 1234 ae@65.108.89.50`).
2. Clone the POIROT repository: `git clone https://github.com/HexHive/droidot.git`.
3. Change to the repository directory: `cd droidot`.
4. Change to the static analysis AE directory: `cd static_ae`.
5. Run `./setup.sh` to download dependencies and build the Docker container.
6. Run `./build_docker.sh` to download dependencies and build the Docker container.
7. Spawn a shell in the container: `./start_docker.sh`.

A.3.2 Basic Test

ARM64 Fuzzing Server
(ssh -p 1235 ae@65.108.89.50)

Run the `./start_single_emu.sh` script. After the script has finished running `adb devices` should show at least one emulator, `emulator-5554`, inidicating that the emulator was started succesfully.

x86 Static Analysis Server
(ssh -p 1234 ae@65.108.89.50)

Run the `./basic_test.sh` script. The script will analyze an app with FlowDroid. After a couple of minutes, the script should print `Found 2 leaks`.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Android dataflow state of the art tools do not scale to large apks. POIROT's analysis passes scale to large apks.

(C2): POIROT's analysis passes positively impact coverage.

(C3): POIROT finds bugs in real-world apps.

A.4.2 Experiments

The first experiment will be run on the x86 server. The second and third experiments will be run on the ARM64 server.

(E1): [C1] [10 human-minutes + 10 compute-hour]:
Preparation: Login to the x86 server (`ssh -p 1234 ae@65.108.89.50`) and open the script `run_rq1.sh` in an editor to edit the `NO_OF_APPS` variable. In our paper we used 100 apps, but with the timeout of 30 minutes and testing 5 tools, that amounts to 0.5h × 100 × 5 = 250 hours. For the artifact evaluation we limited the `NO_OF_APPS` variable to 10 apps. We believe this reduced dataset is still representative of the original results.
In the folder `target_APK` we provide the 100 APKs used in the paper (the most popular 100 apps) However, reviewers are also welcome to test with different apps. To do so, they can add new apps to the `target_APK` folder, with both the folder and the apk name as the package name of the app (e.g. `com.example.app/com.example.app.apk`).
Execution: Start a tmux session with `tmux` and run the script `run_rq1.sh`.
Results: The script prints the result of the various tools and the failure symptom. The results should be congruent with Table 2.

(E2): [C2] [10 human-minutes + 12 compute-hour]: Fuzzing ablation experiment on POIROT's analysis passes.
Preparation: Login to the arm64 server (`ssh -p 1235 ae@65.108.89.50`), start a tmux session with `tmux` and then start the container with `./run.sh` .
Execution: Run the `./run-ablation.sh` script. The script runs the ablation study fuzzing campaign, fuzzing a subset of the apks from the dataset with /without the argument value and call sequence analysis pass.
Results: After finishing the script generates the plots from Figure 5 and Figure 6 in the output-dir: `$(datetime)_fuzzing_data`. Download them from the server: `scp -r -P 1235 ae@65.108.89.50:/home/ae/droidot/[output-dir]` and inspect the pdfs. These should show that enabling analysis passes results in coverage increase. (larger green area).

(E3): [C3] [5 human-minutes + 20 compute-minutes]: Reproduce the case study crash.
Preparation: Login to the arm64 server (`ssh -p 1235 ae@65.108.89.50`), start a tmux session with `tmux` and then start the container with `./run.sh`.
Execution: Run the `./reproduce-tplink.sh` script.
Results: The script generates the harness for the vulnerable function, fuzzes it and then reproduces the crash. After finishing the script prints the backtrace of the reproduced and deduplicated crash. It should look similar to this:

```
abort
scudo::die
scudo::ScopedErrorReport::~ScopedErrorReport
scudo::reportInvalidChunkState
scudo::Allocator<scudo::AndroidConfig, &scudo_malloc_postinit>::deallocate
mp4_write_one_jpeg
Java_com_tplink_skylight_common_jni_MP4Encoder_packVideo
fuzz_one_input
main
```

The backtrace contains the allocators detection of double free indicating the detection of the use after free.

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at `https://secartifacts.github.io/usenixsec2025/`.