USENIX Security '25 Artifact Appendix: TapTrap: Animation-Driven Tapjacking on Android

Philipp Beer
TU Wien

Marco Squarcina
TU Wien

Sebastian Roth
University of Bayreuth

Martina Lindorfer
TU Wien

A Artifact Appendix

A.1 Abstract

We provide multiple artifacts to reproduce the results presented in the paper and support future work that builds on our findings. Specifically, we include the following artifacts:

- **Dataset preparation:** Scripts for crawling the Google Play Store, downloading apps, and preparing them for analysis (`/dataset_preparation`). Due to the dataset size, we cannot publicly release the full set of apps used. Reviewers are granted access to it as outlined in Section A.3.1.
- **Malicious app detection:** Code and results for malicious app detection (`/malicious_app_detection`).
- **Vulnerable app detection:** Code and results for vulnerable app detection (`/vulnerable_app_detection`).
- **User study:** Materials from the user study, including the information sheet, consent forms, questionnaires, app, and website used during the study (`/user_study`).
- **TapTrap PoC:** Proof-of-concept of TapTrap (`/poc`).
- **Reproducibility scripts:** Scripts to reproduce the results presented in the paper (`/reproducibility`).
- **Supplementary files:** Additional files not directly relevant for artifact evaluation, such as assets and paper-related licenses (`/assets` and `/paper_licenses`)

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Excessive or rapid scraping and downloads of APKs from the Play Store may violate the Play Store's terms of service and could result in temporary or permanent IP bans. The dataset preparation pipeline uses a conservative approach to avoid excessive downloads. We nevertheless recommend reviewers use a dedicated machine to avoid potential issues.

Running the KillTheBugs app and the TapTrap PoC are intended solely to demonstrate the security vulnerability described in the paper. We do not perform destructive operations and do not collect personal data.

A.2.2 How to access

The artifacts accompanying the paper are hosted at https://github.com/beerphilipp/taptrap and archived at https://doi.org/10.5281/zenodo.15519676.

A.2.3 Hardware dependencies

Access to the APKs. Due to the size of the APK dataset used in the paper, we cannot distribute it directly. Reviewers are granted access to it as outlined in Section A.3.1.

System Requirements. Both x86 and ARM architectures are supported. For running an Android emulator, however, we recommend a Mac with Apple Silicon. A minimum of 16 GB RAM and 150 GB available disk space are suggested.

Physical Android Device. We recommend running the apps on a physical Pixel 6a device running Android 15. While they can be run on other devices, screen element positioning may differ and require manual adjustment for the attack to run. Alternatively, the app can be executed in an emulator.

A.2.4 Software dependencies

Operating System. We have tested and support Ubuntu 24.04 and macOS 15 with a desktop environment. Other systems *may* require adjustments.

Docker. Install Docker (see `/reproducibility/README.md` for a step-by-step guide or https://docker.com/get-started for official instructions).

Rsync. We require `rsync` (preinstalled on Ubuntu 24.04 and macOS 15) to retrieve the APK dataset. We have, however, experienced issues with it on macOS 15 and suggest installing it via Homebrew instead (`brew install rsync`).

Java. To be able to use the necessary Android dependencies, install a recent version of Java (see https://www.java.com/en/download/manual.jsp)

Android Dependencies. Install the Android dependencies (see `/reproducibility/README.md` for a step-by-step guide):

- Download and install the Android command line tools, which include `sdkmanager`.
- Install the platform tools to install `ADB`.
- Set the `ANDROID_HOME` environment variable.
- Add the platform tools to `$PATH`.

A.2.5 Benchmarks

Evaluating the analyses pipelines requires access to the APK dataset we used. See Section A.3.1 for access instructions.

A.3 Set-up

A.3.1 Installation

Clone the Repository. Clone the artifact repository using `git clone https://github.com/beerphilipp/taptrap.git` in a directory of your choice.

Install the Dependencies. See A.2.4 for instructions.

Obtain a Google AAS token. Downloading apps from the Play Store requires a Google account and an AAS token. We provide such credentials for reviewers to use. Other researchers may create a new Google account and refer to `/dataset_preparation/downloader/README.md` for a summary on how to generate an AAS token.

Download APKs. Our experiments use a subset of 500 randomly selected apps and 266 predefined apps from the dataset. To access the dataset, save the private SSH key that we provide for reviewers to a file named `~/.ssh/taptrap_key` and give it the correct permissions with `chmod 600 ~/.ssh/taptrap_key`. Researchers may request access to the APK dataset by contacting the authors.
Run the following commands in the artifact's root directory:

- Select 500 random apps:

```
rsync -e "ssh -i ~/.ssh/taptrap_key" -azn \
--out-format="%n" dl@download.st1.secpriv.wien: . | \
grep -v "/$" | sort -R | head -n 500 > /tmp/apps.txt
```

- Add the set of predefined apps:

```
cat reproducibility/fixed_apps.txt >> /tmp/apps.txt
```

- Download the selected apps, where `<DIR>` refers to where the APKs should be stored:

```
rsync -e "ssh -i ~/.ssh/taptrap_key" -avxz \
--files-from /tmp/apps.txt \
dl@download.st1.secpriv.wien: <DIR>
```

Start the Android Emulator or Connect a Device. Connect the physical device to the host machine via USB. If you are using an emulator, run `reproducibility/start_emulator.sh` in the repository's root directory to automatically download the correct emulator image for your system and start it.

Set up the Android device. On physical devices, enable USB debugging in *Settings* > *About phone*, then tap *Build number* seven times to enable developer options. Go to *System* > *Developer options* and enable *USB debugging*. This step is not required for emulators.

A.3.2 Basic Test

To verify correct installation and setup, run `reproducibility/basic_test.sh <email> <token>` in the repository's root directory. Replace `<email>` and `<token>` with the Google credentials. This command will perform the following steps:

- Builds all required Docker images.
- Checks if the provided Google credentials are valid by attempting to download an app.
- Checks if an Android device is connected.

The script should print *OK* to the console. Depending on the device resources, this may take up to 20 minutes.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): The app downloading and preparation process from the Play Store, as described in Section 5.1, is functional and yields a large dataset for further analysis. This is proven by experiment *E1*.

(C2): TapTrap is effective on Android 15 as described in Section 3 of the paper. This is proven by experiment *E2*.

(C3): A large-scale analysis of 99,705 apps using the animation detection pipeline (cf. Section 5.2) identified 61 unique animations that bypassed the intended duration limit of 3,000ms (Section 5.3.1) and 28 apps containing animations with a maliciousness score of at least 50 (cf. Section 5.3.2). This is proven by experiment *E3*.

(C4): 76.3% of analyzed apps are vulnerable to TapTrap based on the static analysis pipeline provided in Section 6 and Table 3. This is proven by experiment *E4*.

A.4.2 Experiments

(E1): Dataset preparation [15 human-minutes + 1 compute-hour + 20 GB disk]

Preparation: Follow the steps in Section A.3.1 to set up the environment.

Execution: Run `reproducibility/e1.sh <EMAIL> <TOKEN> <OUT>` in the repository and replace `<EMAIL>` with the Google account email, `<TOKEN>` with the AAS token, and `<OUT>` with the desired output directory. The script performs a Play Store crawl and download and prepares the APKs for analysis. Due to time constraints, the crawl stops after 25,000 package names, and only a random subset of 50 apps is attempted for download.

Results: The script should output `OK`.

Alternatively, manually verify that:

- `<OUT>/apps.csv` contains $\geq 15,000$ package names (lower bound after crawling 25,000 package names)
- `<OUT>/apps` contains ≥ 30 apps (lower bound after downloading 50 apps)
- Note that lower bounds are due to regional restrictions and possible device limitations.

(E2): TapTrap functionality [30 human-minutes + 10 compute-minutes + 15 GB disk]

Preparation: Follow the steps in Section A.3.1 to set up the environment, then run `reproducibility/e2.sh` from the repository root to install and launch the app. Note that running an emulator on x86 with nested virtualization is extremely slow and may cause the attack to fail.

Execution: Click the "Start" button in the app to initiate the attack, then click "Click here". The app secretly opens a camera permission prompt and attempts to trick the user into granting access.

Results: The app should display *"Permission granted"* without the user being aware of granting the permission.

Alternatively, manually verify that camera access has been granted to the app: long-press the app icon, select "App info", then "Permissions".

(E3): Detection of potentially malicious animations [15 human-minutes + 45 compute-minutes + 100 GB disk]:

Preparation: Follow the steps in Section A.3.1 to set up the environment, including downloading the APKs.

Execution: Run `reproducibility/e3.sh <APK_DIR> <OUT_DIR>` in the repository root, where `<APK_DIR>` is the input directory containing the APKs and `<OUT_DIR>` is the output directory for results. The script analyzes a random subset of 500 apps, plus a fixed subset of apps that span the 61 animations exceeding 3,000 ms in duration and those that have a score of at least 50.

Results: The script should output `OK`.

Alternatively, manually verify that the generated report at `<OUT_DIR>/report.tex` states:

- `maltapNumberUniqueAnimationsExtendedDuration`: 61 unique animations exceeding the 3,000 ms duration threshold were found (cf. Section 5.3.1).
- `maltapNumberAppsAnimationsScoreMin`: 28 apps containing at least one animation with a maliciousness score of at least 50 were found (cf. Section 5.3.2).

(E4): Detection of vulnerable apps [15 human-minutes + 4 compute-hour + 100 GB disk]

Preparation: Follow the steps in Section A.3.1 to set up the environment, including downloading the APKs.

Execution: Run `reproducibility/e4.sh <APK_DIR> <OUT_DIR>`, where `<APK_DIR>` is the directory containing the APKs and `<OUT_DIR>` is the output directory for results. The script executes the vulnerable app detection pipeline on the app dataset.

Results: The script should output `OK`.

Alternatively, manually verify that 76.3% ±10% of analyzed apps are vulnerable by inspecting the `vulntapAmountAppsMinOneActivity VulnerablePercent` macro in the generated report located at (`<OUT_DIR>/report.tex`). The error margin accounts for possible variability introduced by analyzing only a subset of apps.

A.5 Notes on Reusability

To foster future research and make it easier for others to build on our work, we provide detailed documentation in the `README.md` files included in each subdirectory. These files include troubleshooting information, describe how to adjust the analysis pipeline (e.g., changing the level of parallelism), explain the code organization and module structure, and outline usage outside Docker environments.

A.6 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/.