USENIX Security '25 Artifact Appendix: HyTrack: Resurrectable and Persistent Tracking Across Android Apps and the Web

Malte Wessels, Simon Koch, Jan Drescher, Louis Bettels, David Klein, Martin Johns
Technische Universität Braunschweig
{malte.wessels, simon.koch, jan.drescher, louis.bettels, david.klein, m.johns}@tu-braunschweig.de

A Artifact Appendix

A.1 Abstract

Android apps can freely intermix native and web content using Custom Tabs and Trusted Web Activities. This blurring of the boundary between native and web, however, opens the door to HyTrack, a novel tracking technique. Custom Tabs and Trusted Web Activities have access to the default browser state to enable, e.g., seamless reuse of authentication tokens. HyTrack abuses this shared browser state to track users both in-app and across the web using the same identifier. While we do not find direct evidence that our technique is already employed, our findings indicate that all essential components are currently in place.

This artifact contains a demo with description, the proof-of-concept apps and backend, as well as our tools for our three measurements: Static app analysis, dynamic network analysis of apps, and a web measurement.

A.2 Description & Requirements

Since the dynamic app network analysis requires rooted devices and an intricate network setup, we exclude it from the functionality artifact evaluation. However, our artifact contains the specific version of the pipeline for posterity. It is based on a previous publication at USENIX with minimal changes. Our version is functional but not within the scope of this artifact evaluation. Refer to that previous publications artifact appendix[1], the READMEs in our artifact, as well as the current development version[2] for more information.

A.2.1 Security, privacy, and ethical concerns

Since HyTrack is a Proof-Of-Concept implementation of a new tracking technique, it tracks users by design. Make sure that the general public does not use PoC instances. Since it is a PoC implementation, we do not guarantee security. However, it is neither destructive nor a security risk by design.

The web scraper will request the specified hosts, and the static analysis pipeline will decompile the provided APKs.

A.2.2 How to access

The artifact is available on Zenodo at https://doi.org/10.5281/zenodo.14718794.

A.2.3 Hardware dependencies

Our PoC can be run on real Android phones. However, we rely on the Android emulator to streamline the testing for this artifact evaluation. Therefore, we require a capable x86 host system with hardware-accelerated emulation and a few gigabytes of RAM and storage for the Android guest. Hardware acceleration can be turned on in the host's UEFI, and it is called Intel Virtualization Technology (VT-x, vmx) or AMD-V (SVM)[3].

Additionally, we require about 20 GB of Docker images that will be pulled or built.

A.2.4 Software dependencies

A modern Linux system with docker, docker-compose, and Python 3. An Android toolchain with the commands adb, emulator, avdmanager, and sdkmanager must be available. Refer to your distros resources or Android Studio to install them.

A.2.5 Benchmarks

The app analysis requires an (any) Android app as a single APK file to be run on. An easily accessible source for free APKs is fdroid [4]. The web scraper requires a list of hosts to scrape. However, it is bundled with one exemplary list.

[1] https://www.usenix.org/conference/usenixsecurity23/presentation/koch
[2] https://github.com/App-Analysis/scala-appanalyzer
[3] https://developer.android.com/studio/run/emulator-acceleration
[4] https://f-droid.org/en/packages/

A.3 Set-up

A.3.1 Installation

Emulator Setup Create a new Android virtual device (AVD). Ensure the image is available by running `sdkmanager "system-images;android-30;google_apis_playstore;x86"`. Create the device via `avdmanager create avd -n hytrack0 -k "system-images;android-30;google_apis_playstore;x86" --device "Nexus 5"`. Start the device via `emulator @hytrack0`. Note that this blocks this terminal window.

Mobile Chrome Setup Wait for the device to boot and launch Chrome. Click through its first-use setup. Trusted Web Activities used by HyTrack require a Digital Asset Link. Regularly, Chrome only accepts HTTPS to fetch Digital Asset Links We must disable this for local testing[5]: In the mobile Chrome, navigate to `chrome://flags`. Search for and enable "Enable command line on non-rooted devices". Restart Chrome as prompted. On the host, run `adb shell "echo '_ --disable-digital-asset-link-verification-for-url=\"http://10.0.2.2\"' > /data/local/tmp/chrome-command-line"`.

Android Build Container We can use a docker container to build the apps and avoid conflicts with your host system. This requires a small setup step, as the apps are signed with a release key. In the PoC subfolder, run `docker run -it -v `pwd`:/in runmymind/docker-android-sdk:ubuntu-standalone-20250210 --entrypoint=bash`. This will launch a shell in the container. To generate signing keys for testing run `keytool -genkey -v -keystore /root/testdeploykey -keyalg RSA -alias key0`. Use 'password' as the password. Skip all questions with Enter, and confirm the empty selections with a 'yes'. Don't close this container's terminal until you have built the apps by completing the following steps. The working directory is mounted inside the container at `/in`.

A.3.2 Basic Test

1. To test the PoC web server, it is sufficient to run `docker-compose up` on the host system in the PoC/webapp directory and to navigate to localhost in your favourite browser. A page is loaded.

2. To test the PoC apps (CrossAppLauncher, CrossAppTrackerOne), compile them with the build container: In the container's terminal, cd into one app's folder at `/in` and run `./gradlew assemble`. Repeat with the other app.

3. To test the webscraper, run `docker build . -t wk-crawler:latest` in the webscrape folder.

4. To test the static app analysis, run `docker-compose --env-file .env up -d` in `staticanalysis/analysis/pipeline/`.

5. Ensure that the demo folder contains a playable video and a description file.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *HyTrack can track users across the web and native applications. It can restore itself from browser storage resets, and is seamless.*

(C2): *We provide toolchains to search for indications of HyTrack in the wild.*

A.4.2 Experiments

(E1): *[Demo reenactment] [20 human-minutes]: Reenacting the demo shows HyTrack capabilities.*

How to: Watch the demo video and reenact it in the emulator. Note that in HyTrack threat model, a tracking provider hosts its backend on a domain. In the paper, we've used the generic domain ad.com. For the demo, we hosted the backend on the URL `examplecorp.de`. Replace that with `http://10.0.2.2` in the emulator, which maps to localhost port 80 on your host system. The other tracking web page maps to `http://10.0.2.2:8080` in the emulator.

The 20 minutes includes getting familiar with the video and setup. The demo itself can be conducted in two minutes.

Preparation: With a running emulator, web server, and compiled apps, as described in the Set-Up and Basic test sections, install both apps to the emulator with `adb install app-release.apk` in the `app/build/outputs/apk/release/` directories.

Execution: Watch the demo video in the Demo folder, read the demo description in the file next to it, and reenact it step-by-step in your emulator setup. Adjust the domains as described above. Instead of clicking on shortcuts to URLs, please type them in Chrome's URL bar. The 'clear storage' option might be moved to 3-Dots-Menu -> Settings -> Privacy.

Results: A similar behavior to the video should be achieved. Some minor differences can occur: Due to the debugging mode, Chrome displays a warning about enabled flags, which can be dismissed. Depending on the Chrome version present on the virtual device, the 'Running in Chrome' notice and position of the 'clear storage' options might be different.

[5] https://developer.chrome.com/docs/android/trusted-web-activity/integration-guide#debugging

(E2): *[Webscrape] [10 human minutes]*

How to: Relates to C2: To demonstrate our web scraper, which scrapes the DAL from websites, we run it on three websites. DALs are required to run TWAs, one component used by HyTrack.

Preparation: `docker build . -t wk-crawler:latest` in the webscrape folder.

Execution: To scrape google.com, wikipedia.org, and youtube.com for DALs, run `docker run -v $PWD/resources:/resources wk-crawler:latest -endpointsFilePath=/resources/dal.txt -websitesFilePath=/resources/test_top-5000.csv` in the webscrape folder. Afterward, copy the results out of the docker container: `docker cp containerName:/proj/wellKnownCrawlerMain/crawl_results results`. Replace `containerName` with the container name. It can be found via `docker ps -a | grep wk-crawler`.

Results: A results folder, containing subfolders. The `endpointInformation` subfolder contains a (potentially hidden) `.well-known` subfolder, containing a subfolder for assetlinks, which includes a JSON. This should contain results for the asset links found.

(E3): *[Static App Analysis] [10 human minutes]*

How to: Relates to C2: We will run the static analysis pipeline on one application to demonstrate its functionality. It searches for features that could be used to implement HyTrack or its components.

Preparation: Download one APK[6] in a new folder `staticanalysis/analysis/pipeline/apks`. Run `python analysis/pipeline/folder2json.py apks/` in that folder to generate the file `apps.json`. This file is the input for the static pipeline.

Execution: Run `docker-compose --env-file .env up -d` to prime the pipeline. If you get a port blocked error, quit the emulator or any running ADB process, as ADB defaults to the same port. Run `docker-compose --env-file .env --profile init up -d` to start the analysis.

Results: A JSON file in the `analysis/pipeline/results` folder. It contains many keys for CT features, TWA features, extracted DALs, and schemes. For the example app, all entries are empty/false except for schemes, which contain 'HTTP' and 'HTTPS'.

A.5 Notes on Reusability

This work and our HyTrack-PoCs are intended as an early warning sign, we assume and hope that the issue is fixed systematically in the future. The PoCs could then be used to validate the fixes.

The dynamic app analysis pipeline is actively maintained. If you intend to use it for your work, check out its latest version on GitHub[7] to profit from all future fixes and features. The static app analysis is a fork from previous work by Beer at al. which is available at https://github.com/beerphilipp/tabbed-out.

Refer to the READMEs in the subfolders for more details.

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/.

[6] https://f-droid.org/repo/cz.martykan.webtube_17.apk

[7] https://github.com/App-Analysis/scala-appanalyzer