USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Lost in the Mists of Time: Expirations in DNS Footprints of Mobile Apps

Johnny So, *Stony Brook University;* Iskander Sanchez-Rola, *Norton Research Group;* Nick Nikiforakis, *Stony Brook University*

https://www.usenix.org/conference/usenixsecurity25/presentation/so

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

USENIX Security '25 Artifact Appendix: Lost in the Mists of Time: Expirations in DNS Footprints of Mobile Apps

Johnny So
Stony Brook University

Iskander Sanchez-Rola
Norton Research Group

Nick Nikiforakis
Stony Brook University

A Artifact Appendix

This artifact appendix is meant to be a self-contained document which describes a roadmap for the evaluation of your artifact. It should include a clear description of the hardware, software, and configuration requirements. In case your artifact aims to receive the functional or results reproduced badge, it should also include the major claims made by your paper and instructions on how to reproduce each claim through your artifact. Linking the claims of your paper to the artifact is a necessary step that ultimately allows artifact evaluators to reproduce your results.

Please fill all the mandatory sections, keeping their titles and organization but removing the current illustrative content, and remove the optional sections where those do not apply to your artifact.

A.1 Abstract

In this work, we present the first large-scale analysis of mobile app dependencies through a dual perspective accounting for time and version updates, with a focus on expired domains. First, we detail a methodology to build a representative corpus comprising 77,206 versions of 15,124 unique Android apps. Next, we extract the unique eTLD+1 domain dependencies — the "DNS footprint" — of each APK by monitoring the network traffic produced with a dynamic, UI-guided test input generator and report on the footprint of a typical app. Using these footprints, combined with a methodology that deduces potential periods of vulnerability for individual APKs by leveraging passive DNS, we characterize how apps may have been affected by expired domains throughout time. Our findings indicate that the threat of expired domains in app dependencies is nontrivial at scale, affecting hundreds of apps and thousands of APKs, occasionally affecting apps that rank within the top ten of their categories, apps that have hundreds of millions of downloads, or apps that were the latest version. Furthermore, we uncovered 40 immediately registrable domains that were found in app footprints during our analyses, and provide evidence in the form of case studies as to their potential for abuse. We also find that even the most security-conscious users cannot protect themselves against the risk of their using an app that has an expired dependency, even if they can update their apps instantaneously.

As part of the artifact evaluation, we release datasets and analysis notebooks that enable reviewers to reproduce the figures and tables that are presented in the text. Additionally, we release a version of the main app analysis infrastructure to enable future work.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

There are no security, privacy, and ethical concerns associated with running these artifacts. In terms of running processes, running the Jupyter notebooks requires a running Jupyter server, and running the Cuttlefish infrastructure spawns various disposable components. However, we do recommend using a local firewall (e.g., ufw) if running the Cuttlefish infrastructure, as it will spawn Cuttlefish virtual devices that can be manipulated over the network via adb. The only host-level settings that are modified come from the Cuttlefish infrastructure, which provides scripts to manipulate iptables and ufw rules (`iptables.sh` and `ufw_modify_cvd.sh` in the `/cuttlefish/scripts` directory, respectively). These scripts enable all traffic from the Cuttlefish virtual devices to correctly pass through ufw (if it is enabled), and route DNS traffic from the Cuttlefish devices to a specified DNS server.

A.2.2 How to access

We provide access to our artifact on Zenodo at the following link: https://doi.org/10.5281/zenodo.14737144. The artifact will be updated with new versions at this URL according to the discussion of the artifact evaluation period.

A.2.3 Hardware dependencies

The only hardware requirements are imposed by the Android Cuttlefish infrastructure components, which require CPU support for `kvm`.

For convenience, we provide reviewers with access to a VM which we have pre-configured with all necessary dependencies, and installed the artifacts at `/home/ubuntu/artifacts`. In addition, this VM comes with the sample of APKs used for the Cuttlefish experiments

(which unfortunately cannot be publicly shared in the artifact itself). We recommend using the provided VM as a reference as there are various dependencies required by the Cuttlefish infrastructure components.

A.2.4 Software dependencies

These artifacts were exclusively developed and tested on an Ubuntu 20.04 machine. The Jupyter notebooks should work on any platforms that support Python, and the Cuttlefish-based infrastructure should work on most Linux distributions. If you would like to run the analysis notebooks, the dependencies are fairly simple. However, if you would like to set up your own testing environment with the Cuttlefish virtual devices, there are additional dependencies that need to be installed, and certain components may not be cross-platform compatible (e.g., the script that uses `iptables` to route DNS traffic from Cuttlefish devices).

To run the Jupyter Notebooks, the only requirement is a Python environment with all the dependencies in `/requirements.txt`.

To run the Cuttlefish app analysis infrastructure, there are additional requirements:

1. Docker Engine for the DNS servers and HTTPS proxies
2. Android Cuttlefish - which can be installed using `cuttlefish_setup.sh` and `cuttlefish_download_images.sh` in the `/cuttlefish/scripts` directory
    - **NOTE:** `kvm` needs to be supported by your CPU
3. The fork of DroidBot in `droidbot.tar.gz` installed in the Python environment
    - After inflating the `droidbot` folder, it can be installed via `pip install -e /path/to/droidbot`
4. Frida server - which can be installed by `download_frida_server.sh` in the `/cuttlefish/scripts` directory
5. `iptables` to route DNS traffic from the Cuttlefish devices
6. **[recommended]** `ufw` as Cuttlefish devices expose additional ports
7. **[recommended]** a desktop environment / VNC server (e.g., `turbovnc`) to interact with the Cuttlefish devices

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

As Zenodo records only support flat files, the artifacts have been compressed.

1. Download the data artifacts from Zenodo at https://doi.org/10.5281/zenodo.14737144 and ensure they are all in the same directory.
2. `chmod +x inflate.sh && chmod +x deflate.sh`
3. `/inflate.sh` to inflate the artifacts

Depending on whether you would like to only run the analyses with Python or run the Cuttlefish infrastructure, you will need to install different dependencies. The steps below outline the installation steps for both cases.

Python Environment [Jupyter and Cuttlefish]. The official downloads can be found on the Python site, but Python is likely to have been pre-installed on your machine. We recommend creating a separate virtual environment for the required Python dependencies for this artifact (see this for a primer on virtual environments).

1. After setting up your Python virtual environment, please install the packages in `/requirements.txt` with the corresponding commands for your environment manager (e.g., `pip install requirements.txt`).

The below dependencies are only required to test the Cuttlefish infrastructure.

Python Environment [Cuttlefish, required]. After installing your Python environment as described above, you will also need to install the provided fork of DroidBot to use in the Cuttlefish infrastructure. To do so, please inflate the artifacts, and run:

```
pip install -e /path/to/droidbot-fork
```

inside your Python virtual environment.

Docker [Cuttlefish, required]. The official setup instructions can be found at the Docker docs site.

1. Docker Engine can be installed on Linux with `docker_setup.sh` in the `/cuttlefish/scripts` directory.
2. Log out and log back in so that your group membership is re-evaluated to run `docker` commands without `sudo`.

Android Cuttlefish [Cuttlefish, required]. The official setup instructions can be found at the Android Open Source site.

1. Cuttlefish packages can be set up on Linux with `cuttlefish_setup.sh` in the scripts directory.

2. Download the following Cuttlefish Android 11 artifacts from aosp-android11-gsi@11718355 to some directory (e.g., `$HOME/cf-images/11718355` which is created by the prior script):
   (a) `aosp_cf_x86_64_phone-img-11718355.zip` for the Android image
   (b) `cvd-host_package.tar.gz` for the host cuttlefish utilities
3. Extract the downloaded artifacts by running:

```
tar -xvf cvd-host_package.tar.gz
unzip aosp_cf_x86_64_phone-img-11718355.zip
```

4. Then, add the absolute path of the `cf-images/android11/bin` to your PATH (it is recommended to add to do so in a persistent manner)

Desktop Environment/VNC [Cuttlefish, recommended]. If you are working in a remote/headless environment, it is recommended to install a desktop environment and a VNC server so that you can visually monitor and control the Cuttlefish virtual devices with a browser and WebRTC.

It should be possible to connect to a remote WebRTC process from your local computer without installing a desktop environment on the machine running the Cuttlefish infrastructure (e.g., with SSH port tunneling), but we encountered problems with this and found that installing the desktop environment was simpler.

If you need help, here is an example guide on how to install TightVNC with Xfce4 on Ubuntu.

Configuration. After installing the required dependencies, make sure to update the configuration for the desired components. For the Jupyter notebooks, the main configuration file is `/analysis/.parameters.py`, but this should not require any changes as long as the directory structure was not modified after inflating the artifacts.

For the Cuttlefish infrastructure, the main configuration file is `/cuttlefish/.env`. In particular, make sure to change the following variables:

1. `DIR_BASE` to the absolute file path to the inflated `/data/cuttlefish` directory
2. `DIR_ADB_FILES` to the absolute file path to the inflated `/cuttlefish/adb_files` directory
3. `MITMPROXY_CACERT_FILENAME` to be the name of the created mitmproxy CA certificate produced by the `generate_mitmproxy_cert.sh` script. See Section A.3.2 for more details on what to put for this setting.
4. `NUM_APPS_PER_SAMPLE` to the number of apps per sample group desired
5. `NUM_CUTTLEFISH_DEVICES` to the number of Cuttlefish devices

Additionally, configure `sudo` to allow executing the script `cuttlefish/scripts/iptables.sh` without requiring a password. This script is executed for each Cuttlefish virtual device in the current user, so it normally requires elevated privileges. After verifying the contents of the script, you can do this by running `sudo visudo` and adding the following line:

```
your_username ALL=(ALL) NOPASSWD: /path/to/cuttlefish/scripts/iptables.sh
```

A.3.2 Basic Test

To test functionality of the Jupyter notebooks, activate the Python environment (with `poetry shell` in the artifact directory in the provided VM), and launch a Jupyter server by running `jupyter lab` in the artifact directory. Then, configure SSH port forwarding to your local machine over the default Jupyter port 8888, and navigate to the URL from the output of the Jupyter command in your browser of choice.

To test functionality of the Cuttlefish devices after installing all dependencies, perform the following:

1. Navigate to your `cf-images` directory where you downloaded the Android Cuttlefish images (e.g., `$HOME/cf-images/11718355`), and run the following:
   `HOME=$PWD ./bin/launch_cvd -num_instances=2 -resume=false -start_webrtc=true -start_vnc_server=false`
2. Wait until you see a message in the output (colored green) asking you to navigate to `https://localhost:8443`
3. Connect to the desktop environment, launch a browser, and navigate to `https://localhost:8443`.

Next, bootstrap the mitmproxy CA certificate creation that will be injected into the Android devices by doing the following:

1. In `/cuttlefish/compose.yaml`, change the file path of the `hardumps` volume to the absolute path of the inflated `data/cuttlefish/hardumps` folder.
2. Navigate to the `/cuttlefish` directory and run `docker compose up -build -d`.
3. After the containers have been created, run the `generate_mitmproxy_cert.sh` script from the same directory.
4. The CA certificate should then be in the `/cuttlefish/adb_files` directory (i.e., `c8750f0d.0`)

5. Make sure to change the environment variable `MITMPROXY_CACERT_FILENAME` in `/cuttlefish/.env` to the name of the certificate.

Then, test that the Cuttlefish virtual device is appropriately configured by the infrastructure by running `python analyze.py -test`. You should see that the Cuttlefish devices are set up before the script exits.

Finally, modify the input APKs to run with the Cuttlefish infrastructure by:

1. Modifying the file `samples_cuttlefish.csv` in `/data/cuttlefish/samples`, adding the SHA256 `hash`, `package` name, `version` string, and `group` name (the provided samples file uses `successful` and `unsuccessful` to refer to the originally-successful and originally-unsuccessful APKs).

2. Place the added APKs into the directory `DIR_BASE/samples/<group>` with the name `<package>_<version>_<hash>.apk`.

3. To run the actual analysis, run `analyze.py` without the test flag.

A.4 Evaluation workflow

A.4.1 Major Claims

The major claims made in the paper are as follows:

(C1): *Expired domain names are found in the DNS traffic of multiple versions of Android apps, regardless if they are out of date or the latest available versions.*

(C2): *The purpose and use of many such domains can be identified from the context of the request and inspecting the decompiled APKs, and they can be abused by malicious re-registrants to change app behavior, even if the app itself does not change.*

(C3): *All eight figures and four tables presented in the text can be produced from our extracted data. However, please note that because the commercial telemetry data we used cannot be released, this artifact produces slightly different versions of Figures 3 and 4, and Table 4. Additionally, Figure 7 is also slightly different, but because of minor differences in how the original code processed the raw data.*

(C4): *The app analysis infrastructure extracts DNS network traffic of APKs.*

A.4.2 Experiments

The following experiments can be performed to verify that the artifacts are functional and can be used to reproduce the results from the text. The Jupyter notebooks used for analysis are bundled with pre-processed data so that the raw data — which is large — does not have to be re-processed.

(E1): *[5 human-minutes + 5 compute-minutes + 0GB disk]: Run all cells in the footprints.ipynb notebook and verify that domains were checked for expirations.*
How to: *Ensure the basic test for the Jupyter notebooks has been completed, with a Jupyter server now running in a Python environment with the dependencies installed.*
Preparation: *Connect to the Jupyter server with your browser and open the `analysis/footprints.ipynb` notebook.*
Execution: *Press Run → Run All Cells.*
Results: *The `df_footprints` variable is a DataFrame that has a column named `expired_at_exec`, denoting domains that were expired during app execution. The rows that have this column as True span multiple versions of different apps.*

(E2): *[30 human-minutes + 5 compute-minutes + 0GB disk]: Run all cells in the decompilations.ipynb notebook and verify that most of the domains can be found directly inside the APK, and their purpose can be identified from inspecting the decompiled APKs.*
How to: *Ensure the basic test for the Jupyter notebooks has been completed, with a Jupyter server now running in a Python environment with the dependencies installed.*
Preparation: *Connect to the Jupyter server with your browser and open the `analysis/decompilations.ipynb` notebook.*
Execution: *Press Run → Run All Cells.*
Results: *Verify that the `Searching for Domains in Decompiled Code` launches grep commands to search the decompiled APKs for their corresponding domains. Manually confirm that files in the directory `DIR_BASE → samples-jadx-processing/expired_at_reg` contain the output of the grep commands. Randomly sample some of the identified domains to look at the decompiled APKs.*

(E3): *[10 human-minutes + 10 compute-minutes + 0GB disk]: Run all cells in each Jupyter notebook and verify that all figures and tables are produced.*
How to: *Ensure the basic test for the Jupyter notebooks has been completed, with a Jupyter server now running in a Python environment with the dependencies installed.*
Preparation: *Connect to the Jupyter server with your browser and open the `footprints.ipynb`, `decompilations.ipynb`, and `cuttlefish.ipynb` notebooks in the `analysis/` directory.*
Execution: *Press Run → Run All Cells in each notebook.*
Results: *Verify that `footprints.ipynb` produces Tables 1, 2, and 4 and Figures 2, 3, 4, 7, and 8, `decompilations.ipynb` produces Tables 3 and 4, and Figure 5, and `cuttlefish.ipynb` produces Figure 6.*

(E4): *[10 human-minutes + 3*2*N/D compute-minutes + 0GB disk]: Launch the Cuttlefish infrastructure on a sample of APKs.*
How to: *Ensure the basic test for the Cuttlefish infrastructure setup has been completed and the configuration has been updated.*
Preparation: *Modify the* `cuttlefish/.env` *file and change* `NUM_APPS_PER_SAMPLE`*, the number of apps that will be analyzed from each of the two sample groups (N), and* `NUM_CUTTLEFISH_DEVICES`*, the number of live Cuttlefish virtual devices (D).*
Execution: *Using the configured Python environment, run* `python analyze.py` *in the* `/cuttlefish` *directory.*
Results: *When analysis of an app completes, there is a pcap file in* `DIR_BASE` → `DIR_DATADUMPS` → `tcpdumps` → `<group>` → `<package>_<version>_<hash>_<timestamp>.pcap`.

A.5 Notes on Reusability

The number of Cuttlefish devices that can be launched concurrently can be adjusted by modifying the value of the `num_instances` flag provided to the `launch_cvd` command, changing the `NUM_CUTTLEFISH_DEVICES` variable in `cuttlefish/.env`, and adding additional Docker containers to `cuttlefish/compose.yaml` if necessary. Furthermore, the Cuttlefish infrastructure can be reused for different APK inputs, and can be done by modifying the input `samples_cuttlefish.csv` file that describe the APKs and placing them into the expected locations. The analysis notebooks will continue to function even with the addition of new data, although it may also be desirable to insert your own API keys for Farsight and Dynadot to analyze new domains.

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/.