USENIX Security '25 Artifact Appendix: "Endangered Privacy: Large-Scale Monitoring of Video Streaming Services"

Martin Björklund, Romaric Duvignau
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden
martibjo@chalmers.se, duvignau@chalmers.se

A Artifact Appendix

This artifact appendix guides the reader into reproducing the results (i.e., plots) of our paper. Since we provide the raw network capture and the entire tool chain to reproduce our evaluation, all claims contained in our work can be reproduced with our artifact.

A.1 Abstract

Our work presents a fingerprinting-based attack on streaming platforms where the attacker is shown to be capable of rapidly and accurately identifying any video from the streaming service, based on capturing network traffic (either as a "strong attacker" by having access to the network or by eavesdropping a wifi connection aka the "weak attacker" model).

Our entire fingerprint dataset (covering the streaming platforms *Amazon Prime Video*, *Max* and *SVT Play*) and all the needed tools to run the attack are made publicly available in our Zenodo permanent artifact[1], which includes:

- TSV datasets (in `datasets/tsv/`) containing video metadata and fingerprints from Amazon, Max, and SVT;
- karl (Go source files `software/karl/`, binary `karl-x86_64-linux`), our data collection tool for retrieving fingerprints;
- BurstShark (Rust source files `software/burstshark/`, binary `burstshark-x86_64-linux`), our TShark wrapper tool for network traffic capture;
- Models (Python modules `software/models.py` and `software/datasets.py`) implementing the video identification logic.

In addition, our artifact also contains the data that we have used during our evaluation:

- PCAP capture files from the main evaluation (in `datasets/pcaps/`), the unknown traffic experiment (in `datasets/unknown-pcaps/`), and DAITA testing (in `datasets/daita-pcaps/`);
- An evaluation script (Python script `software/evaluate.py`) to reproduce the main results;
- Miscellaneous scripts (in `software/misc/`), e.g., the automated WebDriver streaming script (`software/misc/webdriver-stream/main.py`).

At last, note that tables appearing in our evaluation are simply summarization of results that our evaluation script also produces. We also provide the pcaps we used with Mullvad's DAITA service activated (in `datasets/daita-pcaps`) that we mention in § 5 "Mitigation" as a preliminary evaluation.

In addition, to speed-up the reproducibility of all our results, we provide preprocessed pcap data in `software/precomputed/`. This data allows calculating the results of the identification of the 900 videos played during our evaluation for one set of parameters within a few minutes. The preprocessed data can also be reproduced as well but requires longer time and larger memory requirement as the 3 k-d trees are stored in memory during the evaluation (following the weak attacker model).

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

We hereby certify that all our tools are harmless. The source code of the three developed programs is shared and is short to verify (under a few hundred lines of codes for each tool):

- for our network capture program BurstShark, in the `software/burstshark` folder (written in Rust) for the precompiled binary `burstshark-x86_64-linux`;
- for our fingerprinting data collection tool karl, in the `software/karl` folder (written in Go) for the precompiled binary `karl-x86_64-linux`;

[1] https://doi.org/10.5281/zenodo.14676526.

- in the `software/` folder for our Python identification tool: modules `datasets.py` and `models.py` and our evaluation script `evaluate.py`.

Disclaimer: One of our artifact's tool (karl) collects data directly from live services, which might contravene terms of service (ToS) of the targeted platforms. It's designed for research purposes and should be used with caution. Additionally, since the program interacts with live services, the functionality may break at any moment if the targeted services update their implementation.

A.2.2 How to access

Our artifact is permanently hosted on the Zenodo platform: https://doi.org/10.5281/zenodo.14676526.

A.2.3 Hardware dependencies

No special hardware is required to run our artifact, however the memory requirements are as follows:

- For running the evaluation based on **preprocessed pcaps**: a minimum of about **32 GB** is recommended. In our tests, memory usage of the evaluation script typically peaked at about 20 GB.
- For running the evaluation from the **raw pcap files** as well as running the identification tool (with a database of 240k videos for the 3 services, as in our evaluation) in live tests: a minimum of **192 GB** of memory is recommended.

A.2.4 Software dependencies

Our tools are dependent on the following libraries/programs:

- **Python packages** (main identification tool): joblib, numpy, pandas, pyarrow, python-dateutil, pytz, scikit-learn, scipy, six, threadpoolctl, tzdata; (web driver used in our automatized evaluation) attrs, certifi, charset-normalize, h11, idna, outcome, packaging, PySocks, python-dotenv, requests, selenium, sniffio, sortedcontainers, trio, trio-websocket, typing_extensions, urllib3, webdriver-manager, websocket-client, wsproto;
- **Python 3.12.4**;
- **tshark** for BurstShark.

Required python packages are listed in the corresponding requirement files.

In our artifact, we provide precompiled binaries for Linux X86_64 architecture. If one needs other architecture or wishes to compile the binaries from the source files, the standard **Go** and **Rust** compilation toolkit are necessary.

In this guide, we give command-lines for a Unix user, however, all our tools are also available on other OS. For running the time measurement tests (E1 in § A.4.2), the `bc` utility is used, it can be installed e.g. on Debian-based distributions with:

```
sudo apt-get install bc
```

Streaming Platform Subscriptions In this guide, all examples do not require any login information on the streaming platforms used. In particular, this concerns retrieving the fingerprint directly from a video's URL using our karl tool. To run our data collection tool for the Amazon Prime and Max platforms, one needs to set the required cookies by being authenticated on the streaming platforms, i.e., the authentication cookies ("ubid-main", "x-main", and "at-main" for Amazon; "st" for Max) are required so to access certain data.

A.2.5 Benchmarks

The only required datasets and models are the ones provided in our artifact and the VNAT public dataset.

A.3 Set-up

A.3.1 Installation

Download The README provided as part of our Zenodo artifact record describes how to setup our artifact:

1. Download[2] the two necessary archives from Zenodo's permanent record datasets.tar.gz (**23.8 GiB**, uncompressed 97.2 GiB) and software.tar.gz (**5.9 GiB**, uncompressed 7.9 GiB) and decompress both archives in the same directory, i.e., run the following commands:

```
mkdir endangered-privacy
cd endangered-privacy
wget https://zenodo.org/records/15051891/files/
     datasets.tar.gz
tar -xvzf datasets.tar.gz
wget https://zenodo.org/records/15051891/files/
     software.tar.gz
tar -xvzf software.tar.gz
```

2. To rerun the full unknown traffic evaluation, the VNAT dataset is required; the dataset can be downloaded from MIT's webserver[3]. Download (**36 GB**) and extract the dataset from VNAT Dataset directly into the `datasets/unknown-pcaps/` directory (e.g., `datasets/unknown-pcaps/VNAT_release_1/`).

Installation (via conda) Here follows guided instructions for installing the required Python packages via conda[4] with

[2] For convenience, we provide here direct links to elements of version v5 of our artifact. If a newer version becomes available, we recommend to download the latest version of the artifact.

[3] https://www.ll.mit.edu/r-d/datasets/vpnnonvpn-network-application-traffic-dataset-vnat.

[4] https://www.anaconda.com/download/success.

a dedicated virtual environment for running the code. First conda must be installed, thus follow the corresponding guide for the system at hand[5]. For Linux-64, this is done as follows:

```
mkdir -p ~/miniconda3
wget https://repo.anaconda.com/miniconda/Miniconda3-
     latest-Linux-x86_64.sh -O ~/miniconda3/miniconda.sh
bash ~/miniconda3/miniconda.sh -b -u -p ~/miniconda3
rm ~/miniconda3/miniconda.sh
```

Then, run the following commands:

```
conda create -n py3124 python=3.12.4
conda activate py3124
python -m ensurepip --upgrade
pip install --upgrade pip
pip install -r requirements.txt
pip install matplotlib
```

Now, you can check that you can run the evaluation script:

```
python evaluate.py --help
```

Installing TShark Install TShark following https://tshark.dev/setup/install/, e.g. on Ubuntu run:

```
sudo apt-get install tshark
```

Compile binaries from sources (optional) We use here the standalone binary `burstshark-x86_64-linux` for our tool BurstShark. If one needs to recompile it, use the source files from folder `burstshark` and the appropriate Rust tools.

The karl binary, `karl-x86_64-linux` included in the package, is our custom data collection tool. The source code is available in the `karl/` directory and can be compiled with Go tools.

A.3.2 Basic Test

Guided test of the data collection tool Let us first illustrate how we gather information and metadata about the videos we would like to fingerprint. Our data collection tool has two options `extract-urls` and `extract`, cf. running the tool with `-help` flag. Here is an example of such usage:

```
./karl-x86_64-linux extract https://www.svtplay.se/video
     /e9qaywe/leif-och-billy/1-en-ny-tjej-i-byn
```

Once karl has acquired the necessary metadata for a given video, the tool builds a fingerprint for each representation of the queried video using the information from its manifest file.

For instance, when the above command-line is executed, karl retrieves 10 fingerprints of the 10 representations (aka "qualities") associated with the queried video, with the downloaded data organized in a json (called `extract_<timestamp>.json`.) with the following format:

[5] https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html.


```
{
  "service": "svt",
  "url": "https://www.svt[...]billy/1-en-ny-tjej-i-byn",
  "videos": [
    {
      "id": "eQJRo6j",
      "title": "Leif \u0026 Billy - 1. En ny tjej i byn",
      "playback_url":
           ↪ "https://www.svtplay.se/video/eQJRo6j",
      "duration": 835,
      "expires_at": "2025-04-01T23:59:00+02:00",
      "variants": [
        {
          "mime_type": "video/mp4",
          "codecs": "hvc1.2.4.L123.90",
          "width": 1280,
          "height": 720,
          "bandwidth": 2875167,
          "fingerprint": {
            "segment_sizes": [
              812697,
              1337592,
              1274793,
              [...]
            ],
            "segment_durations": [
              49152,
              49152,
              49152,
              [...]
            ],
            "timescale": 12800
          }
        },
        [...]
```


karl can also build a fingerprint directly from a manifest file's URL, useful for fingerprinting arbitrary manifests/mp4 files; an example of such functionality can be tested by running the following command-line:

```
./karl-x86_64-linux fingerprint https://media.axprod.net
     /TestVectors/Cmaf/clear_1080p_h264/manifest.mpd
```

This generates a file `fingerprint_<timestamp>.json`, with the following json schema (`variant` schema is identical to `variants`'s format from extract's json):


```
{
  "url": "https://media[...]_1080p_h264/manifest.mpd",
  "variant": [
    {
      "mime_type": "video/mp4",
     [...]
[...]
```


As of March 2025, you can test the fingerprinting functionality on the 6 provided example manifest files listed in our `README.md`, covering DASH manifests, HLS manifests and Fragmented MP4.

A single entire video representation is typically gathered in the matter of 10-100ms, hence already demonstrating that our data collection **does not** require to watch the fingerprinted videos. Note that our karl tool is designed to make the process of mass collecting of such fingerprints very efficient: connections will be kept open, reused and parallelized. Also, if

no country code is explicitly specified, a small geolocation lookup subroutine is run once before downloading the fingerprint(s). SVT fingerprints generally take slightly longer because of explicit addressing requiring many requests, but even for those videos, our tool still runs in subsecond time per representation.

The "segment sizes" information is the one our identification tool uses. After running our tool over the 3 studied streaming platforms, these information have then been gathered into 3 datasets. You can check how a representation is encoded into our tsv database:

```
head -n 2 datasets/tsv/amazon_representations.tsv
```

Guided test of the identification tool For quick evaluations or experiments (the default setting), precomputed results can be used, significantly reducing the processing time (minutes) and system requirements.

The standard parameters for the evaluation script (that we have used in most of our evaluation) are:

- `-model strong,weak,unknown` – all models;
- `-method network,wifi` – all methods;
- `-theta THETA` – θ =2.2;
- `-time [1-600]` – 600;
- `-recompute` – this extra flag is not added at this stage so not to recompute tsv files from raw pcaps (cf. complete evaluation in § A.4.2);
- `-burstshark-path` – only if the binary is not `./burstshark-x86_64-linux`.

For example, to run a single model/network evaluation with a given θ, use:

```
python evaluate.py --model strong --method network --
    theta 1.5 --time 600
```

The expected output is as follows:

```
Loading dataset... DONE
Eval. model 'strong' with method 'network'... DONE
Saved ./results/strong-network-1.5-600.csv
Saved ./results/strong-network-1.5-600-aggregated.csv
```

Hence, this first test produces 2 "result" csv files saved in the `results` folder, with the non-aggregated file showing our identification results (successful identification and identification time) per played video (from a pre-preprocessed pcap capture):

```
device service video id[...] unknown mis[...] id_time
linux amazon amzn1.[...]2c56 1 0 0 49.15[...]
linux svt jwAypqm 1 0 0 59.9738812446594
linux max 899[...]448 1 0 0 60.25[...]
windows svt jaRm5gA 1 0 0 87.2628059387207
windows amazon amzn1.[...]70a2 1 0 0 72.60[...]
windows max e7[...]ccf6 1 0 0 70.41[...]
mac amazon amzn1.[...]f92d 1 0 0 61.4[...]
```

As an indication, on our recent but not high-end laptop (CPU: Intel Ultra 7 155U, 12 cores; 32GB RAM), it took about 3min to run that test with the program's memory usual usage at around 10GB and peaking at about 18-20 GB. Since this test is running over a fixed network capture and our identification system is deterministic, re-running the evaluation will produce identical results (note that `identified_time` refers to the packet capture time not the time of the identification system, which is negligible in comparison).

The aggregated results merge the results per platform, eg for $\theta = 1.5$, accuracy, precision and recall are about 98.3% for the Max service.

To run our complete evaluation script with the standard parameters, simply run:

```
python evaluate.py
```

This test took for us about 6min and test the strong attacker with "network" method as well as the weak attacker with both "network" and "wifi" methods. The unknown traffic test creates a result `unknown-scores.csv` file with the following simple format:

```
dataset application score
self hbomax 0.167802384471442
self hbomax 0.584277198585956
self hbomax 0.52584947872736
```

Since this shows the maximum score obtained by the most likely representation-timestamp from the fingerprint database, it suffices to check if any is above a given threshold θ to count "false positives"; the highest value for the maximum score is 1.998 in our experiments, hence any $\theta > 2$ will produce 0 false positive with this test data. Recall, our recommended and default value for θ is 2.2.

A.4 Evaluation workflow

A.4.1 Major Claims

The main claims of our work can be summarized as:

(C1): *It is fast for the attacker to build the necessary fingerprint database*. We provide our capture tool and show here how to run it with some examples. The required information for an entire video is typically acquired in less than 100ms per fingerprint, demonstrating the rapidity at which one can build the database. We publish our entire fingerprint database acquired by running the tool over the libraries of the three services, something easily reproducible with a laptop and an ordinary internet connection (some of the services require being authenticated).

(C2): *The proposed attack quickly shows high accuracy, regardless of the type of attacker*. This claim is supported by Figure 6 which can be reproduced by running our evaluation script over the PCAP captures that we have recorded during our experiments. In addition, the scripts

that were used to generate our network captures are also provided, hence one could also generate new PCAPs under the same conditions. Figure 8 can also be reproduced, simply by running the provided evaluation with different value for our threshold parameter θ.

(C3): *The false positive rate is very low*. This claim is supported by the distribution of maximum scores produced (Figure 7) by our system on the "unknown traffic". We provide the pcaps corresponding to HBO Max and our youtube captures in folder `datasets/unknown-pcaps/self` whereas the VNAT dataset is publicly available and should be added in `datasets/unknown-pcaps/` to run the script without modification. Our provided evaluation script directly calculates the scores of Figure 7 from the network captures.

A.4.2 Experiments

To check our claim C1:

(E1): *[Data-Collection] [15 human-minutes]: Test the provided tool for building the fingerprint dataset, and check time measurements of acquiring video fingerprints.*
Preparation: Follow the previous instructions on running the example tests to setup the system.
Execution: Run the commands with example URLs and manifest files following our artifact's README. Check gathered information and jsons and our fingerprint tsv databases. To test the speed of our data collection, download and execute the following speed tests from our artifact (this assumes the artifact was uncompressed in the current directory as per § A.3.1's instructions) – time per fingerprint is the displayed time divided by 10 (number of representations for the tested video):

```
wget https://zenodo.org/records/15051891/files/time-
    experiment.tar.gz
tar -xvzf time-experiment.tar.gz
cd time-experiment/
./time.sh 20 ../software/karl-x86_64-linux --out-
    dir time-test-svt extract https://www.svtplay.
    se/video/edPkzzZ/leif-och-billy/1-stromlost
```

Optionally, test also with more SVT videos and Amazon's time collection following the time experiments' README's instructions.
Results: Check the (mean, median, standard deviation) times it took for collecting the json fingerprints.

Claims C2-C3 can be assessed using the preprocessed pcaps following the test guide. One can also run our complete evaluation directly from raw pcaps (hence, running our burst generator over the recorded network traffic) as follows:

(E2): *[Complete-Evaluation] [30 human-minutes + about 5-6 compute-hour + 192GB main memory]: To run the entire evaluation directly from raw pcaps, it suffices to run the evaluation program with the parameters used in our evaluation.*
Preparation: Follow the previous instructions on running the example tests to setup the system.
Execution: To recompute all data, run the command:

```
python evaluate.py -recompute
```

In our experiments, it took about 90min to build the trees and about 5h to run BurstShark over all the raw pcap files. The resulting files are saved in `results/` folder and should be identical to the files[6] used in our evaluation.
Results: The claims are supported by reproducing the results as shown in the paper, see `results/` folder.
Use the provided plotting code to regenerate the paper's result figures (Figure 6, 7, and 8).
For Figure 6 of the paper, we record our information statistics for every seconds between 1 and 600 with $\theta = 2.2$; to reproduce the figure, run:

```
python plot.py 6
```

For Figure 7, the file `unknown-scores.csv` is used to plot the distribution of maximum scores over the unknown dataset; to reproduce the figure, run:

```
python plot.py 7
```

To reproduce Figure 8, one simply has to generate the result files for every θ between 1 and 5 by increment of 0.05, and the figure is generated by running:

```
python evaluate.py --all-theta
python plot.py 8
```

A.5 Notes on Reusability

We welcome any re-use of our artifact in further research.

Licensing Information Our Zenodo deposit contains both source code and datasets, each with separate licensing:

- **Source Code**: BSD 3-Clause License.
- **Datasets**: Creative Commons Attribution 4.0 International.

By using our artifact, you agree to comply with these respective licenses.

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] All result files are provided for convenience in the folder `precomputed-results/` in https://zenodo.org/records/15020614/files/plot-and-results.tar.gz.