Drawn Apart

A Deep-Learning Enhanced GPU Fingerprinting Technique

Tomer Laor\textsuperscript{1}, Naif Mehanna\textsuperscript{2}, Antonin Durey\textsuperscript{2}, Vitaly Dyadyuk\textsuperscript{1}, Pierre Laperdrix\textsuperscript{2}, Clémentine Maurice\textsuperscript{2}, Romain Rouvoy\textsuperscript{2}, Walter Rudametkin\textsuperscript{2}, Yossi Oren\textsuperscript{1}, Yuval Yarom\textsuperscript{3}

(1) Ben-Gurion University of the Negev
(2) Univ. Lille / Inria / CNRS
(3) University of Adelaide
A possible use case

Unethical advertiser

Jack likes sports
Jack uses facebook & reddit

Let’s show sport ads in there!
A possible use case

Unethical advertiser

Jack likes sports
Jack uses facebook & reddit
Let’s show sport ads in there!

Jack disabled his cookies
He made sure to randomize his browser fingerprint
He did not log in sensitive website
A possible use case

How did the advertiser manage to track Jack?
A possible use case

Unethical advertiser

We used innovative hardware fingerprinting!
What can we extract from it?
Each GPU, even from the same model, shows **differences** on some scale.
We need to be able to:

● Experiment with different scripts interacting with the GPUs.
● Run the same code on multiple machines with the same software and hardware.
● Have the multiple machines in the same environment (temperature, pressure, etc…)
The Setup
The Setup

Our identical PCs with our daemon

Commands

Data

Our server
Web Environment

We need to use WebGL to run code on the GPU from a web page.

WebGL doesn’t have mutexes.
def drawn_apart_offscreen():
    times = []
    for vertices_to_stall in power_set(vertices_num):
        start_time = time.now()
        apply_in_parallel(vrtx_to_render=>render_vertex(vertices_to_stall, vertex_to_render))
        end_time = time.now()
        times.append(end_time - start_time)
    return times

def render_vertex(vertices_to_stall, vertex_to_render):
    if vertex_to_render not in vertices_to_stall:
        render(color='green')
    else:
        render(color=intensive_compute())
Web Environment - Offscreen

These traces were classified using Random Forest.
## Web Environment - Offscreen Results

<table>
<thead>
<tr>
<th>Device Type</th>
<th>GPU</th>
<th>Device Count</th>
<th>Base Rate (%)</th>
<th>Accuracy (%)</th>
<th>Gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel i5-3470 (GEN 3 Ivy Bridge)</td>
<td>Intel HD Graphics 2500</td>
<td>10</td>
<td>10</td>
<td>36.3±1.6</td>
<td>3.6</td>
</tr>
<tr>
<td>Intel i5-4590 (GEN 4 Haswell)</td>
<td>Intel HD Graphics 4600</td>
<td>23</td>
<td>4.3</td>
<td>63.7±0.6</td>
<td>14.7</td>
</tr>
<tr>
<td>Intel i5-8500 (GEN 8 Coffee Lake)</td>
<td>Intel UHD Graphics 630</td>
<td>15</td>
<td>6.7</td>
<td>55.5±0.8</td>
<td>8.3</td>
</tr>
<tr>
<td>Intel i5-10500 (GEN 10 Comet Lake)</td>
<td>Nvidia GTX1650</td>
<td>10</td>
<td>10.0</td>
<td>70.0±0.5</td>
<td>7.0</td>
</tr>
<tr>
<td>Apple Mac mini M1</td>
<td>Apple M1</td>
<td>4</td>
<td>25.0</td>
<td>46.9±0.4</td>
<td>1.9</td>
</tr>
</tbody>
</table>
Can DrawnApart Work On Mobile Phones?

<table>
<thead>
<tr>
<th>Device</th>
<th>Resolution</th>
<th>Available Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Galaxy A53</td>
<td>2400 x 1080 (FHD)</td>
<td>1</td>
</tr>
<tr>
<td>Galaxy S21 FE</td>
<td>2340 x 1080 (FHD+)</td>
<td>2</td>
</tr>
<tr>
<td>Galaxy Tab S8+</td>
<td>2000 x 1752 (WQXGA+)</td>
<td>5</td>
</tr>
<tr>
<td>Galaxy Tab S8 Ultra</td>
<td>2960 x 1840 (WQXGA+)</td>
<td>3</td>
</tr>
<tr>
<td>Galaxy S22 Ultra</td>
<td>3088 x 1440 (Quad HD+)</td>
<td>2</td>
</tr>
<tr>
<td>Galaxy S22+</td>
<td>2340 x 1080 (FHD+)</td>
<td>7</td>
</tr>
<tr>
<td>Galaxy S22</td>
<td>2340 x 1080 (FHD+)</td>
<td>1</td>
</tr>
<tr>
<td>Foldable Display</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Galaxy Z Flip3</td>
<td>2560 x 1600 (WQXGA)</td>
<td>34</td>
</tr>
<tr>
<td>Galaxy Z Fold3</td>
<td>2560 x 1600 (WQXGA)</td>
<td>9</td>
</tr>
<tr>
<td>Galaxy Tab S6</td>
<td>2400 x 1080 (FHD+)</td>
<td></td>
</tr>
<tr>
<td>Galaxy Tab S5e</td>
<td>2560 x 1600 (WQXGA)</td>
<td></td>
</tr>
<tr>
<td>Galaxy S20 FE</td>
<td>2400 x 1080 (FHD+)</td>
<td></td>
</tr>
</tbody>
</table>

Remote Test Lab
Web Environment - Results

We swapped the hard drives of 2 devices in the Intel i5-3470 set.

**Spoiler:** We were still able to identify the correct device using DrawnApart!

What happened?

Scan to watch the cyb3r video!
Web Environment - Hypothesis

WebGL deterministically assigns execution units to vertices.
AmIUnique Integration

AmIUnique has a Chrome extension that follows changes in your browser fingerprint over time.

We integrated the DrawnApart offscreen method with AmIUnique in order to gather data in a real world setting.
AmIUnique Integration

- Make sure that users won’t feel slowdowns
- Select the best stall function for the in-the-wild settings
- Ensure that it will support all the different configurations that can occur in the wild.
The dataset contains ~370,000 fingerprints collected from ~2,500 unique devices through the AmIUnique platform. Each collection includes 7 traces.

https://amiunique.org
Large Scale Experiment - ML

We first tried to use Random Forest.

But… training on ~2,500 labels required an extensive amount of RAM → Not ideal in a real world setting

We tried to make clusters of devices using the canvas hash and renderer string, but the story wasn’t compelling enough…
Large Scale Experiment - ML

Neural networks have the expressive power that we need, with a reasonable runtime and RAM usage.
We trained a CNN with semi-hard triplet loss to map the original feature space into a lower dimension Euclidean space.
In The Wild Dataset Split

- **1MP**
  - 1MP_65 used for training the neural network
  - 1MP_rest used for standalone evaluation
  - 3-Jan-2021

- **2MP**
  - 2MP used for standalone evaluation
  - 7-Feb-2021

- **3MP**
  - 3MP used for improving the SOTA evaluation
  - 3-May-2021
  - 8-Jul-2021
Grid5000 is a **big cluster** that let us access machines with powerful GPUs.

We trained the DrawnApart deep learning solution on Grid5000.

https://grid5000.fr
Prior to picking the Convolutional Neural Network, we tried various methods:

**Vision Transformers** → lower accuracy overall - harder to optimize

**Training using a Siamese Networks setting** → harder to optimize

**LSTM networks** → Significantly lower accuracy

**DeepAR** → We considered a trace to be a time-serie - bad accuracy
What Would an Attacker Do?

1) Gather a lot of data from a lot of users

2) Train an embedding CNN

3) In production: transform each incoming trace using the CNN and compute the distance to the existing embeddings.
Evaluation: In-the-Wild Conditions

- We have a lot of data at hands
- Results are good thanks to the amount of data

<table>
<thead>
<tr>
<th>Top-10 Accuracy</th>
<th>Top-5 Accuracy</th>
<th>Top-1 Accuracy</th>
<th>Rate Top-1 Base</th>
<th>Nu. of Traces For Memorizing</th>
</tr>
</thead>
<tbody>
<tr>
<td>67.15%</td>
<td>55.09%</td>
<td>28.28%</td>
<td>1.0%</td>
<td>420K~</td>
</tr>
</tbody>
</table>
What Would an Attacker Do?

1) Same attacker - different website with different users.

2) The attacker already have a trained neural network

3) In production: **1-shot learning**
   
   This is a harder task!
Evaluation: K-Shot Conditions

- A small amount of data
- Results are impressive even though the task is harder.

<table>
<thead>
<tr>
<th>Top-10 Accuracy</th>
<th>Top-5 Accuracy</th>
<th>Top-1 Accuracy</th>
<th>Top-1 Base Rate</th>
<th>Nu. of Traces For Memorizing</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>19.95%</td>
<td>14.10%</td>
<td>5.44%</td>
<td>0.00%~</td>
<td>14K~</td>
<td>Shot-1</td>
</tr>
<tr>
<td>26.75%</td>
<td>19.34%</td>
<td>7.11%</td>
<td>0.00%~</td>
<td>59K~</td>
<td>Shot-5</td>
</tr>
<tr>
<td>31.09%</td>
<td>22.77%</td>
<td>9.22%</td>
<td>0.00%~</td>
<td>100K~</td>
<td>Shot-10</td>
</tr>
</tbody>
</table>
Improving the State-of-the-Art

- **FP-Stalker**: Published at **S&P 2018**

- At the time, it was shown that browser fingerprinting was efficient for identification but **not for long-term tracking**

- **FP-Stalker** showed that browser fingerprinting could be used for long term tracking

FP-Stalker has two main steps:

- Rule-based filtering
- Machine learning inference

(b) Hybrid variant of FP-STALKER. The training phase is used to learn the probability that two fingerprints belong to the same browser instance, and the testing phase uses the random forest-based algorithm to link fingerprints.
FP-Stalker & DrawnApart

<table>
<thead>
<tr>
<th>Collection timestamp</th>
<th>User Agent (HTTP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hashed Canvas (JS)</td>
<td>Do Not Track (JS)</td>
</tr>
<tr>
<td>Language (HTTP)</td>
<td>Cookies (JS)</td>
</tr>
<tr>
<td>Plugins (JS)</td>
<td>Local (JS)</td>
</tr>
<tr>
<td>Renderer (JS)</td>
<td>Flash-based attributes</td>
</tr>
<tr>
<td>Screen Resolution (JS)</td>
<td></td>
</tr>
<tr>
<td>Timezone (JS)</td>
<td></td>
</tr>
</tbody>
</table>

Processed DrawnApart trace
Adapting FP-Stalker

- Between 2017 and today, the web ecosystem changed!
- Flash became unsupported by all major browsers since 2021
- FP-Stalker relied on flash-based attributes for its rule-based step → it couldn’t be applied in the current web

We adapted FP-Stalker to the current web, while ensuring that the results remained on par with the paper.
Adapting FP-Stalker

1) Understanding the logic behind FP-Stalker & inspecting the existing code

2) Identifying what can be optimized → We identified several bugs that could impact the accuracy and optimized the code logic and readability

3) Comparing our adapted version of FP-Stalker to its original algorithm on the provided dataset
Adapting FP-Stalker

In order to introduce DrawnApart into FP-Stalker, we had to identify the ideal position in the algorithm.

We noticed that the Machine learning step classified only a few percentage of the traces due to the generated threshold being too high.
Adapting FP-Stalker

FP-Stalker uses the **average and maximum tracking time** to quantify the performances of its algorithm.

**Average tracking time**: For a given device, how long can we track it on average?

We used both metrics to quantify the improvement of DrawnApart on FP-Stalker.
Adapting FP-Stalker

We integrated our deep-learning pipeline by **short-circuiting** the machine learning step.

If the Cosine distance between two traces is below the threshold, we conclude the search.
Analyzing DrawnApart

![Graph showing probability density against Euclidean distance with thresholds for different device comparisons.](image)
Adapting FP-Stalker

FP-Stalker was criticized for its algorithm being too slow.

We noticed that our updated version with DrawnApart mitigated this limitation by significantly reducing the execution time.

Improving FP-Stalker

Differences in Average Tracking Time between FP-Stalker (\textit{Nude FPS}) and FP-Stalker with DrawnApart (\textit{FPS+DA})
Thank You
Let’s discuss!

GitHub repository

Naif Mehanna
naif.mehanna@univ-lille.fr

Tomer Laor
tomerlao@post.bgu.ac.il