CSE406: Computer Security Sessional Side-Channel Attack

(Including Bonus: Collaborative Dataset Collection)

Masnoon Muztahid
Student ID: 2005067
Section: B1
Dept: CSE
Bangladesh University of Engineering and Technology (BUET)

June 20, 2025

1 Introduction

This project demonstrates a website fingerprinting attack using a cache-based side-channel leak. The technique allows identifying the website a victim is visiting based solely on low-level cache behavior, which is captured through JavaScript. Machine learning models, particularly convolutional neural networks (CNNs), are used to classify the collected cache access traces and determine which website is being visited.

The project also includes a bonus component: collaborative data collection across multiple contributors. This allowed us to scale up the dataset to 51,000 samples, providing a more diverse and realistic evaluation of our models.

2 System Design

The overall system is composed of both frontend and backend components, working together to collect side-channel traces, store them, and perform machine learning-based classification.

- **Frontend UI (Alpine.js + Pico CSS):** A minimalistic web interface built with Alpine.js for reactivity and Pico CSS for styling. Users can trigger actions like collecting latency/trace data, viewing heatmaps, and downloading traces.
- **Timing Measurement (JavaScript):** Uses `performance.now()` and dedicated Web Workers to measure cache access latency at varying scales. Also includes custom workers (e.g., `warmup.js`, `worker.js`) to run data collection in parallel.
- **Backend Server (Flask):** A lightweight Python web server that receives trace data from the browser, generates heatmaps (via matplotlib), and returns classification predictions using a trained model.

- **Database (SQLite):** A simple and efficient local database for storing trace records and metadata.
- **Automation (Selenium):** A headless browser automation framework used to systematically visit websites and collect a large dataset of cache traces for model training.
- **Machine Learning Pipeline (PyTorch):** The backend includes two CNN-based models (Basic and Complex) implemented in PyTorch. These models are trained on normalized trace data and saved as `.pth` files. A trained model is also deployed for real-time website prediction in the Flask app.

3 Task 1: Warming Up with Timing

The first step of this project involved understanding the timing characteristics of cache accesses in modern browsers. We implemented a `readNlines(n)` function in JavaScript to read `n` cache lines and measure the elapsed time using `performance.now()`.

Latency Results

Table 1: Cache Access Latency Measurements

N (Cache Lines)	Median Access Latency (ms)
1	0.00
10	0.00
100	0.00
1,000	0.00
10,000	0.00
100,000	0.30
1,000,000	0.80
10,000,000	8.20

Observations

- `performance.now()` showed limited resolution for small memory reads.
- Cache access latencies became distinguishable for large `n` values (>100,000).
- These results helped us understand how many accesses are needed to generate measurable delays — essential for the success of cache side-channel attacks.

4 Data Collection and Preprocessing

Traces were collected for three websites:

- `https://cse.buet.ac.bd/moodle/`
- `https://google.com`

- `https://prothomalo.com`

Two datasets were prepared:

- **Personal Dataset:** 3,000 traces collected solely on a Linux machine (1,000 per website).
- **Merged Dataset (Bonus Task):** 51,000 traces (17,000 per website) contributed by multiple participants using different platforms (Linux and Windows).

Before training, we applied **min-max normalization** to all traces using the following formula:

$$x' = \frac{x - \min(X)}{\max(X) - \min(X)} \tag{1}$$

Normalization was performed on each dataset **before merging**, ensuring that platform-specific biases were minimized as much as possible.

5 Experimental Results

5.1 Personal Dataset (3,000 traces)

Table 2: Basic CNN Performance (Personal Dataset)

Website	Precision	Recall	F1-score
Moodle	0.85	0.79	0.82
Google	0.80	0.85	0.83
Prothomalo	0.98	1.00	0.99
Accuracy		0.8783	

Table 3: Complex CNN Performance (Personal Dataset)

Website	Precision	Recall	F1-score
Moodle	0.94	0.97	0.96
Google	0.96	0.94	0.95
Prothomalo	1.00	0.99	1.00
Accuracy		0.9683	

5.2 Merged Dataset (51,000 traces)

Table 4: Basic CNN Performance (Merged Dataset)

Website	Precision	Recall	F1-score
Moodle	0.78	0.75	0.76
Google	0.75	0.76	0.75
Prothomalo	0.88	0.90	0.89
Accuracy		0.8037	

Table 5: Complex CNN Performance (Merged Dataset)

Website	Precision	Recall	F1-score
Moodle	0.80	0.79	0.79
Google	0.79	0.78	0.78
Prothomalo	0.89	0.91	0.90
Accuracy		0.8259	

6 Analysis and Findings

Which Websites Were Easiest and Hardest to Classify?

We evaluated each website's classification performance using four metrics: precision, recall, F1-score, and overall accuracy. From both personal and merged datasets, the results consistently show:

- **Easiest to classify:** `prothomalo.com`. It achieved the highest metrics across all settings:
  - In the personal dataset with the Complex CNN, it reached **1.00 precision**, **0.99 recall**, and **1.00 F1-score**.
  - In the merged dataset with the Complex CNN, it still performed best with **0.89 precision**, **0.91 recall**, and **0.90 F1-score**.

  This consistent performance suggests that the cache behavior of `prothomalo.com` is likely more deterministic or distinguishable.

- **Hardest to classify:** `moodle.buet.ac.bd`. It scored lower across all models:
  - In the personal dataset with Basic CNN, it had only **0.79 recall** and **0.82 F1-score**.
  - In the merged dataset with Complex CNN, its F1-score dropped to **0.79**, the lowest among the three websites.

  This indicates that `moodle.buet.ac.bd` likely generates more dynamic or inconsistent cache access patterns, making it less distinguishable for the models.

Effect of Merging Datasets on Accuracy

While the larger merged dataset provided more data points for training, the accuracy dropped slightly. This can be attributed to:

- **Cross-Platform Variability:** Traces were collected on both Windows and Linux, leading to different timing behaviors and cache access patterns.
- **Hardware Differences:** Contributors used machines with varying CPUs, cache sizes, and background loads, introducing noise and inconsistencies.
- **Environmental Noise:** Some traces may have been collected with background browser tabs, applications, or differing network speeds, further affecting uniformity.

Despite normalization, these inconsistencies naturally reduce the overall accuracy when compared to a highly controlled, single-device dataset.

Model Architecture Comparison

The Complex CNN clearly outperforms the Basic CNN across all experiments. It handles noise and variability better, likely due to deeper convolutional layers and batch normalization, which stabilize learning and improve generalization.

Effect of Data Size

The 3,000-trace personal dataset achieved very high accuracy (up to 96.8%), but this was under tightly controlled conditions. The larger merged dataset (51,000 traces), although noisier, allowed us to simulate real-world deployment conditions. Even under such conditions, the best model still achieved over 82.5% accuracy, showcasing the feasibility of this attack in practical scenarios.

7 Bonus Task 3: Real-Time Website Detection

To demonstrate the practicality of this attack, we implemented real-time website prediction into our Flask web app. When a trace is collected, the backend normalizes the data and feeds it into our trained CNN model, which returns the most likely website.

Demo Features

- Fully integrated with the web interface.
- Displays predicted website immediately after trace collection.
- Uses the trained Complex CNN model on the dataset.

Findings

- The model was able to correctly identify the visited website in most real-time scenarios.
- However, occasional misclassifications occurred. This is likely due to the fact that training data was collected in a clean, idle browser environment with minimal background activity, while real-time detection operates in noisier, more dynamic conditions.
- Side-channel signals are extremely subtle, and even small deviations—such as background tabs, browser rendering tasks, or system processes—can introduce noise that confuses the model.
- Despite these challenges, the system effectively demonstrates the feasibility of live website fingerprinting from an adjacent browser tab in real-time.

8 Conclusion

This project demonstrates that website fingerprinting using cache side-channel data is a viable and effective attack method. Key conclusions are:

- CNN-based classifiers can detect websites with high accuracy using only timing traces.
- Normalization is a crucial preprocessing step and was consistently applied before training and merging.
- The collaborative (merged) dataset introduced realistic noise and platform differences, slightly lowering accuracy but significantly improving generalizability.
- `prothomalo.com` was the most distinguishable website, while `moodle.buet.ac.bd` was more ambiguous.
- Complex CNN architectures are better suited for large-scale, noisy data.
- Real-time prediction integration proved that side-channel website fingerprinting can work practically in-browser.
- This project highlights the importance of both timing resolution and collaborative dataset collection for robust experimentation.

9 Resources

- **Google Drive Repository:**
  `https://drive.google.com/drive/folders/1wl9Qvo61TyD-YCbO7urBBfDHU-uknhoV?usp=sharing`

This repository includes:

- Trained PyTorch models (`.pth` files) for both Basic and Complex CNNs
- Merged and personal trace datasets in `.json` format