CSE190 / CSE291I: How the Web Tracks You **Fall 2026**

PA2 — Stateless Tracking

Project Release: 11/4/2026
Deadline: 11/19/2026

Introduction

Eve's company, **PrivacyDefenders**, has recently learned about a novel new technique to target end-users: *stateless tracking*. Stateless tracking is a way to uniquely identify users based solely on their browser patterns and behaviors—no cookies involved at all!

Unfortunately, Eve doesn't really know anything about stateless tracking, and she needs your help to understand the risks of stateless tracking. Your task is to help Eve by implementing several types of stateless tracking so she can try to build defenses against them. Happy hacking!

Goal

This project will introduce you to stateless, fingerprint based tracking. You will implement static and dynamic tracking, and by the end of the project, you should have familiarity with several browser APIs, including the Canvas API and the audio API.

Read this first!

This project asks you to develop fingerprints that can legitimately infringe on people's privacy. Attempting these same kinds of attacks against real people may result in fines, expulsion, and jail time. You must not use these techniques to attack or deanonymize anyone without authorization! You are required to respect the privacy and property rights of others at all times or you will fail the course.

Forming a Group

This is a group project; you can work in a team of size at most two and submit one project per team. You are not required to work with the same team on every project. You and your team should collaborate closely on each part.

The code and other answers you submit must be entirely your team's own work. You may discuss the conceptualization of the project and the meaning of the questions with other teams, but you may not look at any part of someone else's solution or collaborate with anyone other than your team. You may consult published references, provided that you appropriately cite them (e.g., with program comments).

Solutions must be submitted to Gradescope.

Index

- Setup
- Resources and Guidelines
- Static Tracking
- Canvas Fingerprinting
- Audio Fingerprinting
- Submission Details

Setup

We provide you with starter code for each section. You can download the starter code here: https://webtracking.sysnet.ucsd.edu/pa2/starter.zip.

After unzipping the archive, you should see the following directory structure:

```
/
├── static/
│   ├── collect_static.js
│   └── helpers_static.js
├── canvas/
│   ├── collect_canvas.js
│   └── helpers_canvas.js
└── audio/
    └── collect_audio.js
```

In order to complete the assignment, you will be primarily writing client-side JavaScript and Python. You will need the `playwright` package to run certain scripts. You may want to install these in a Python virtual environment to ensure you do not damage your system's global configuration.

Static

The `static` folder contains two files: `collect_static.js`, where your code will live, and `helpers_static.js`, which is a helper file that you do not need to edit.

Canvas

The `canvas` folder contains two files: `collect_canvas.js`, where your code will live, and `helpers_canvas.js`, which is a helper file you do not need to edit. Note that this `helpers_canvas.js` file is *different* from the file in the static directory.

Audio

The `audio` folder contains just one file, `collect_audio.js`, which your code will live.

Part 1: Static Tracking

In part 1, your goal is to build a fingerprinter based on *static features*—these are features that are *properties* of the browser, rather than *behaviors* you need to dynamically infer.

As an example, if you open the file `collect_static.js`. You should see the following code:

```
export async function collect() {
    const out = {};
    try { out["userAgent"] = navigator.userAgent; } catch { out["userAgent"] = null; }
    return fingerprint(out);
}
```

This code is relying on the `navigator` property of the web browser to identify the browser's `UserAgent`, which is a common way to identify a browser.

Your task is to write a fingerprinter that can uniquely distinguish uniquely between **fifteen, statically different browsers**. To test your fingerprint, submit your JavaScript code to: https://webtracking.sysnet.ucsd.edu/pa2/features. We will run your fingerprint across each browser configuration, and send you a report back as to **how many browsers you are able to uniquely identify.** The server will not leak any information to you about *what* it is testing, rather, just how many unique browsers your fingerprinter was able to identify.

As an example, visit https://webtracking.sysnet.ucsd.edu/pa2/features and upload the starter code provided—you will see simply encoding the `userAgent` only uniquely identifies *three* browsers out of fifteen.

Do not modify the final line of `collect`—otherwise the grader will not be able to properly parse your solution and you will receive no points.

> You may be at a total loss to start... that is part of the fun! You may wish to look into some top level static browser APIs, like the `navigator` and `screen` APIs. Happy hunting!

Part 2: Canvas Fingerprinting

In this part, you will implement *canvas fingerprinting*—a robust method that leverages the HTML5 Canvas to distinguish between browsers. For more details on Canvas fingerprinting, you should read the very accessible paper by Mowery and Shacham.

Your task is to use the HTML5 Canvas to build unique fingerprints of the browser your code is running on. You will implement three functions, which are detailed in `collect_canvas.js`: `textCanvasFingerprint`, `detectInstalledFonts`, and `graphicalCanvasFingerprint`. We detail each below:

`textCanvasFingerprint`

In this first section, you need to build a canvas fingerprint that relies on drawing *text* to the Canvas and nothing else. All other parts of the Canvas are disabled on our end. There are lots of examples of text based Canvas fingerprinting, but the simplest version of it exists inside of Mowery and Shacham.

`detectInstalledFonts`

In this next section, you need to use the HTML5 Canvas to determine a list of fonts that are installed on the system. To scope your exercise, you only need to identify which of the fonts in the `FONTS` variable in `helpers_canvas.js` are loaded onto the system. A hint for the algorithm to do this is described in the paper.

`graphicalCanvasFingerprint`

In this final section, we have disabled *text-based fingerprinting*, and so you will need to find another way to create a Canvas fingerprint (by drawing non-text objects).

Similarly to Part 1, we have created a web server for you to test your code: https://webtracking.sysnet.ucsd.edu/pa2/canvas. Your code will be run across **nine**, dynamically unique browsers, all of which test different components of your fingerprints. You succeed if you are able to sufficiently distinguish between *all nine*.

Part 3: Audio Fingerprinting

In this part, you will implement *audio fingerprinting*; a parallel fingerprinting method to canvas fingerprinting that instead fingerprints how the underlying machine processes *audio* (via the Web Audio API) rather than text/image rendering. You can read more about audio fingerprinting here.

Similarly to Part 2, you will need to implement two functions in `collect_audio.js`: `audioSum`, `audioSumWithSimpleFarbling`. One last part, `audioSumWithComplexFarbling`, is extra credit. We detail each below:

`audioSum`

This function should produce a hash that is reliant on the underlying Audio API. There are dozens of ways to produce audio, ranging from simple sounds to extremely complex ones. Your generated audio does not

need to be very complicated to produce a unique fingerprint!

At a high level, you can begin by implementing the following simple algorithm: create an `OfflineAudioContext` (for fast, silent processing), create a sound wave (e.g., via `createOscillator`), compress the sound wave (e.g., via a `DynamicsCompressorNode`), and store the output of the rendered audio (via an `AudioBuffer` and `getChannelData`). If you implement this correctly, it should give you a stable enough fingerprint to pass this first section.

`audioSumWithSimpleFarbling`

Some privacy oriented browsers, like **Brave** and **Firefox**, implement a defense against Audio Fingerprinting called *farbling*. The basic idea is to add some *noise* to the output of common JavaScript APIs, so that standard fingerprints do not work anymore.

In this part, we have implemented a simple farbling defense against the basic audio fingerprint you have built in step 1. Our farble works as follows:

```
out[i] = data[i] * (1 + A_SIMPLE * noise(i));
```

In essence, for every sample, we apply a `noise` function which is uniformly drawn from $[-1, 1)$, fresh for every call to `getChannelData`. Notably, the `noise` distribution has an expectation of **zero**, which might give you some ideas on how to defeat the farble.

> As a hint, consider what might happen if you were to run your fingerprinting code *multiple times*. Can you use that somehow to remove or reduce the noise?

Extra Credit: `audioSumWithComplexFarbling`

In this part, we have implemented a much more complicated farbling mechanism very similar to the one used by Brave. The farbling mechanism employs a random seed value into its noise function, as well as a *second* multiplier on each sample that randomly updates each sample $+/-5\%$. Breaking this farble is *possible*, but will require a deeper understanding of how the underlying audio sampling is morphed by the farble.

Similarly to the previous parts, you can test your JavaScript files at the following URL: https://webtracking.sysnet.ucsd.edu/pa2/audio.

Submission Details

You may use the following "autograders" to test your solutions:

1. **Part 1:** https://webtracking.sysnet.ucsd.edu/pa2/features
2. **Part 2:** https://webtracking.sysnet.ucsd.edu/pa2/canvas
3. **Part 3:** https://webtracking.sysnet.ucsd.edu/pa2/audio

Every autograder requires you to submit your *student ID* along with your JavaScript file. **Each student ID** must be registered in the course and only receives *five submissions a day*. Note that each part of the assignment is handled independently, so you get up to *fifteen* submissions across all three parts. **Course staff reserve the right to increase this limit at our discretion.**

When you are satisfied with your work, you can submit a ZIP of your files to Gradescope. Ensure you do not change the names of the required files, or you will receive a zero on that part of the assignment from the autograder.

Resources and Guidelines

Basic Resources

Review the slides on web fingerprinting, and consider reading How Unique Is Your Web Browser? by Peter Eckersley for more context. You may also consider looking at your fingerprint (and relevant features) from AmIUnique for inspiration.

Mozilla Developer Network Docs

The web documentation at MDN is world-class and gives you a lot to consider as you embark on fingerprinting . Consider reading the Fingerprinting page for hints on how to get started.

Canvas Fingerprinting

Mowery and Shacham's Pixel Perfect paper is a great resource: the link is here.

Audio Fingerprinting

There are *many* audio fingerprinting blogs and resources on the Internet—

If you are stuck as to where to begin, try to read some articles on browser fingerprinting and see if you can discern some common patterns. For all else, come to office hours!