FP-tracer: Fine-grained Browser Fingerprinting Detection via Taint-tracking and Entropy-based Thresholds

Soumaya Boussaha
SAP Security Research / EURECOM

Lukas Hock
SAP Security Research

Miguel Bermejo
UC3M

Rubén Cuevas Rumin
UC3M

Angel Cuevas Rumin
UC3M

David Klein
Technische Universität Braunschweig

Martin Johns
Technische Universität Braunschweig

Luca Compagna
SAP Security Research

Daniele Antonioli
EURECOM

Thomas Barber
SAP Security Research

ABSTRACT

Browser fingerprinting is an effective technique to track web users by building a fingerprint from their browser attributes. It is also stealthy because the tracker uses legitimate JavaScript API calls offered by the browser engine, which can be obfuscated before they are sent to a (third-party) server. Current browser fingerprinting detection methodologies employ limited collection and classification techniques, such as binary classification of fingerprinters based on the number of non-obfuscated exfiltrated attributes. As a result, they produce inconsistent findings. Meanwhile, the privacy of millions of web users is at risk daily.

We address this gap by presenting FP-tracer, a novel methodology to detect and classify browser fingerprinters based on dynamic taint tracking and joint entropy classification. Our methodology enables detecting first- and third-party fingerprinters even when they use obfuscation by tainting attributes, propagating them, and logging when they are leaked (via 62 sources and 25 sinks). Moreover, it discriminates the invasiveness of fingerprinting activities, even from the same service, by measuring the joint entropy of the collected attributes and clustering them.

We implement FP-tracer [3] by extending Foxhound, a privacy-oriented Firefox fork with numeric type tainting, more taint tracking sources and sinks, support for multiple sources, and better logging capabilities. We embed our implementation in our automated crawling infrastructure, which is capable of testing websites in parallel using programmable and reproducible logic. We will open-source our implementation [3].

We evaluate FP-tracer by performing a large-scale crawl over the Tranco Top 100K, and detect, amongst others, audio, canvas, and storage fingerprinting on the web. Among others, we find high fingerprinting activities in 8% of domains, with more moderate activity reaching 75%. Notably, fingerprinting is almost five times more likely to be performed by third-party scripts for high activity levels. In addition, we measure that the most severe category of fingerprinting obfuscates 46% of transmitted attributes, and 38% of fingerprinters involve two or more domains. Finally, we find that existing consent banners do not provide an effective defense against browser fingerprinting.

Proceedings on Privacy Enhancing Technologies 2024(3), 540–560

https://doi.org/10.56553/popets-2024-0092

KEYWORDS

Browser fingerprinting, Tainting, JavaScript, Privacy, Entropy, GDPR

1 INTRODUCTION

Browser fingerprinting is a stealthy technique used to uniquely identify users across the web by crafting fingerprints from browser attributes accessible through its JavaScript APIs. These APIs give access to fingerprintable browsers' configurations, plug-ins, screen dimensions, and installed fonts. Browser fingerprinters are hard to detect because they use legitimate JavaScript API calls, and they generate web traffic that looks benign. On the other hand, they are violating the privacy of millions of Internet users [10, 20, 54].

Several browser fingerprinting detection methodologies have been developed. But their experimental results are *inconsistent*, their collection strategy is *coarse-grained*, and their classification methods are *binary*. For instance, the state of the art has varying opinions on the amount of browser fingerprinting on popular websites, with reported rates ranging from 10% [12, 28] to 70% [10, 39]. Course-grained techniques based on browser API monitoring or code analysis [6, 7, 19, 44] cannot provide details on the destination of collected attributes. In addition, binary classification based on attribute counting [39] or machine learning [28] are either too restrictive or too liberal and only provide part of a broader picture.

We address these three relevant gaps by presenting *FP-tracer*, an innovative browser fingerprinting analysis and detection methodology based on *fine-grained taint-tracking* and *joint entropy thresholds*. As shown in Figure 1, FP-tracer analyzes client-side JavaScript code from first and third-party websites and instruments it to enable *dynamic taint tracking*. This allows for the detection of *data flows* from sensitive browser attributes (sources), such as `Navigator.userAgent`, into JavaScript APIs (sinks), such as `img.src`, which are sent to first- and third-party domains. Then, FP-tracer computes the *joint entropy* of all attributes sent to a particular domain and classifies its fingerprinting activity into six categories ranging from no activity to very high activity to provide a privacy

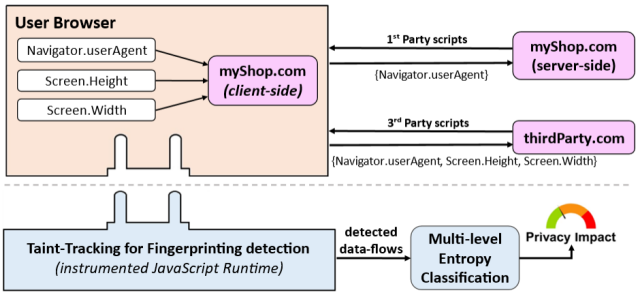

Figure 1: FP-tracer's high-level overview.

impact score. FP-tracer supports 62 sources, including canvas, audio, and storage APIs, and 25 sinks, including XHR requests, element attributes such as `src` and the `sendBeacon` API. See Tables 8 to 10 for the complete list.

We implemented our methodology by *extending Foxhound* [50], an open-source Firefox fork developed for client-side web vulnerability scanning. We enhanced its instrumented JavaScript engine to support numeric data types. We added support for multiple sources using set operations. We improved Foxhound's list of supported sinks and sources and automated their addition using Firefox's WebIDL C++ code generator. Finally, we extended Foxhound's logging capabilities. We also implemented an automated *crawling* infrastructure based on Playwright[42] and integrated our Foxhound extension into the crawler.Our crawler can automatically and reproducibly visit multiple websites using parallel browser instances and a programmable page visit logic. Each instance logs the fingerprinting activities and can be stopped, restarted, or restored in case of a crash.

We built our multi-threshold joint entropy classification using a real-world dataset of approximately 86k fingerprints. The dataset was collected in [14] by running a large-scale experiment on consenting users. Each user browser was fingerprinted using ads containing JavaScript code which accessed sensitive browser attributes. We found that the choice of attributes is critical to the entropy and, therefore, the extent to which users can be identified. For example, while some sites collect up to 25 attributes, similar entropies can be obtained with just 7 carefully chosen ones.

We evaluated FP-tracer with a large-scale fingerprinting experiment visiting 80 618 *domains* from the Tranco Top 100K. Using our fine-grained collection approach, we observed 269 784 fingerprinting flows exfiltrating 15 239 unique browser attribute combinations. With our joint entropy classification technique, we found five types of fingerprinting activities that we label as Negligible, Low, Medium, High, and Very High. We also extract the attribute vectors associated with each activity (e.g., `userAgent` and `storageEstimate` have very high joint entropy).

Our study revealed *novel insights* offering a consistent reading of the inconsistent results published so far and demonstrating the importance of our collection and classification approaches. For example, we found 8% of domains perform fingerprinting in the very high category. This is comparable to the lower rates presented in prior studies [12, 28] at around 10%. Additionally, we observed more moderate fingerprinting activity in 75% of the successfully crawled domains, aligning with the approximate 70% prevalence rates reported previously [10, 39].

Or, for a high level of activity, we find that fingerprinting is almost five times more likely to be performed by a third-party script than a first-party one. By examining over 6 million string values transmitted in our sample, we find that while up to 90% of fingerprinting attributes are transmitted in plain text, very high severity scripts perform some form of obfuscation in 47% of the time. We also find that in this very high category, 38% of fingerprinting is performed by a collusion of two or more domains. In the extreme case, we found two websites sending attributes to a single destination with scripts from 7 different domains.

We validate FP-tracer against the popular Disconnect [1] and EasyPrivacy [2] lists. If our high and very high grouping broadly aligns with these lists, our results also identify intensive fingerprinting activities for domains that were only tagged as general (not invasive) fingerprinters in Disconnect. We also show a good agreement when comparing our findings with results from FP-inspector [28]. Of the 911 domains crawled by both studies, we find at least a moderate fingerprinting activity in 95% of cases. Moreover, we measure whether fingerprinters respect *user consent banners* using the Consent-O-Matic plugin [4, 5, 45]. While we do find that fingerprinting activity tends to increase after a user has agreed to data collection, there is still a worrying baseline of activity, especially in the higher entropy groups.

We summarize our contribution as follows:

- We present FP-tracer [3], a novel methodology to analyze and detect browser fingerprinting using fine-grained taint-tracking and multi-threshold joint entropy. Our techniques, among others, allow us to track different fingerprinting activities from the same fingerprinter, discriminate first-party and third-party trackers, track aggregated attributes, and classify the severity of a fingerprinting activity in 5 groups based on their privacy impact.
- We implement FP-tracer by extending Foxhound, an open-source Firefox fork instrumented for taint analysis, and embedding it in our automated crawling infrastructure. We also built our joint entropy classification from a real-world dataset of heterogeneous browsers fingerprinted on different platforms using advertising JavaScript code. We will *open-source* our implementation and submit it for artifact evaluation [3].
- We conduct a large-scale crawl experiment on the Tranco Top 100K and evaluate our results against real-world fingerprinters (e.g., Disconnect [1], FP-Inspector [28]). Our findings provide a consistent reading of the inconsistent fingerprinting measurement results published so far. In addition, we provide novel insights into third-party activity, collective fingerprinting, and obfuscation techniques.
- We present the first analysis of the user consent implications on browser fingerprinting by running a dedicated secondary crawl, showing that fingerprinting scripts commonly ignore user consent.

```
// Collection
let height = screen.height;
let width = screen.width;
let userAgent = navigator.userAgent;

// Aggregation
let resolution = width * height;
let fingerprint = userAgent + resolution.toString();
let userId = MD5hash(fingerprint);

// Exfiltration
let requestUrl = 'https://example.com?userId=' + userId;
fetch(requestUrl);
```

Listing 1: A simple browser fingerprinting script that collects, aggregates, and exfiltrates three browser attributes.

2 BACKGROUND

Here, we discuss the paper's preliminaries.

2.1 Browser Fingerprinting

A browser fingerprinter tracks users via their browser attributes, such as user agent and keyboard layout [20, 41]. It begins when a fingerprinting script is delivered to the client's browser from a remote server (upper part of fig. 1). This script collects browser attributes, like screen size and width, via the browser JavaScript APIs, constructs the fingerprint, and sends it to a remote server. The collected fingerprinting attributes form the fingerprint. A fingerprint can contain attributes or their aggregate (e.g., a sum, a hash, or an encoding). For instance, listing 1 shows a fingerprinting script collecting three attributes, aggregating them using math operations and hashing, and then exfiltrating the resulting fingerprint.

2.2 Normalized Shannon Entropy

For each fingerprinting attribute X with n distinct values $\mathcal{A}_X = \{x_1, x_2, \dots x_n\}$ and a probability $P(x_i)$ of each value occurring, $H(X)$ (Shannon entropy) is defined as:

$$H(X) = -\sum_{x \in \mathcal{A}_x} P(x) \cdot \log_b P(x) \tag{1}$$

The Shannon entropy is measured in bits and employs a base-2 logarithm ($b = 2$).

To facilitate comparisons between attributes with different cardinalities (where cardinality $|\mathcal{A}_X| = n$), we use H_n (normalized Shannon entropy), which is computed by dividing Equation (1) by the maximum entropy $\log_b(n)$:

$$H_n(X) = \frac{H(X)}{\log_b(n)} \tag{2}$$

H_n measures the uncertainty or information content within fingerprinting attributes in a dataset. It has been used extensively in the past by studies that aimed at determining critical fingerprinting attributes [11, 20, 26, 35].

2.3 Anonymity Sets

An Anonymity set quantifies the number of users with identical or similar fingerprinting attributes within a dataset. A larger anonymity set implies a higher level of anonymity, as it indicates that multiple users exhibit the same fingerprinting characteristics. The anonymity set size A of a particular fingerprinting attribute value x is defined as the number of times that value occurs in a dataset of N samples:

$$A(x) = P(x) \cdot N \tag{3}$$

$A(x)$ describes how many users share the same fingerprint and hence gives a measure of anonymity. High values imply high anonymity, whereas a value of $A(x) = 1$ means that the value x is unique in the dataset.

We also define the mean anonymity set size overall values in the dataset as:

$$\bar{A} = \sum_{x \in \mathcal{A}_x} \frac{P(x) \cdot N}{n} = \frac{N}{n} \tag{4}$$

$\bar{A}$ has the drawback that it depends on the total size of the dataset, and hence difficult to compare between studies. However, it is a more intuitive metric than entropy.

2.4 Dynamic Taint Tracking

Dynamic taint tracking (or *tainting*) is a dynamic analysis technique that employs labels to monitor data flows during program execution [15, 17, 18, 21]. The term "runtime tainting" originates from Perl's taint mode [56]. Various programming environments have integrated built-in tainting capabilities, such as Perl [56], PHP [55], and Ruby [48]. Notable research has utilized taint tracking capabilities: For example, TaintBochs [17] was one of the earliest tools to track data at the byte level.

In theory, taint *sources* are any functions or variables that emit relevant or significant data. The definition of relevance depends on the context of the analysis. For instance, in a security-focused analysis, taint sources might include data that an attacker can manipulate, like input fields on a webpage. In contrast, for a privacy-centered analysis, taint sources typically involve sensitive information about a user. Data originating from a taint source is labeled using a labeling strategy. Some systems use boolean flags, while others employ named tags or dedicated objects. Taint propagation keeps track of labeled data. For example, when tainted values are added to non-tainted ones, the taint should be propagated to avoid a taint loss event. The event is logged when tainted data reaches a taint *sink*. In a privacy context, a sink is an API sending sensitive data to a (third-party) server.

3 THREAT MODEL

In this section, we present our system and attacker models.

3.1 System model

Our system model includes a user employing a (web) browser, such as Firefox or Chrome, to browse the Internet. The browser runs on any compatible device, including desktop computers, laptops, smartphones, or tablets. The browser could run on any relevant operating system like Linux, Windows, iOS, or Android. The user visits web pages that can load first-party or third-party Javascript code into the browser. The browser executes the code by its JavaScript engine, such as V8 (Chrome) or SpiderMonkey (Firefox). The executed code can call any supported JavaScript API to interact with the browser and the device. For example, it can access the screen's width and

height, the local storage, the audio and video setup, the user agent, and so on.

3.2 Attacker model

We consider a *remote* attacker who wants to track a web user via *browser fingerprinting*. The adversary either controls a first-party website that sends Javascript code to the target browser or employs a third-party service from which a legitimate website loads Javascript code. The attacker fingerprints the browser (and hence, the user,) by using its Javascript APIs to collect one or more distinguishing *browser attributes*. For example, in a basic attack scenario, the attacker can tell two browsers apart by looking at their user agents, screen dimensions, and audio configurations.

The attacker has the capabilities of a commercial browser fingerprinter, such as a service based on the FingerprintJS library. She can obfuscate the Javascript API calls to get the attributes (e.g., via minify), aggregate attributes on the browser side before collection (e.g., hash or encrypt them), and fingerprint the browser using scripts loaded from different and unrelated domains. Moreover, the adversary can securely exfiltrate the collected attributes via a TLS connection and conspire with other browser fingerprinters to create more accurate fingerprints. The attacker is interested neither in sending malicious JavaScript code to violate the browser's security (e.g., remote code execution or privilege escalation) nor in interacting with compiled code (e.g., WebAssembly).

4 DESIGN

In this section, we motivate (see Section 4.1) and describe the design of FP-tracer [3], which is based on fine-grained taint-tracking (see Section 4.2) combined with multi-threshold joint entropy classification (see Section 4.3).

4.1 Motivation

While the occurrence of browser fingerprinting on the web at large has been studied widely, the findings are *inconsistent*. Results from the last five years have measured the prevalence of browser fingerprinting from as low as ≈ 10% [12, 28] to as high ≈ 70% [10, 39]. To understand the cause of these inconsistencies, we need to examine how fingerprinting is typically detected. Most studies perform a *collection* stage, where website properties related to fingerprinting are recorded. A *classification* stage follows, where those measurements are used to decide whether fingerprinting activity is present.

Many studies detect fingerprinting activity via dynamic [6, 19, 22, 23, 36, 47, 54] or hybrid [7, 12, 28] analysis to collect the usage of APIs related to fingerprinting. These techniques are limited as there is no certainty that the attributes were sent to a remote server to identify users. Other techniques monitor HTTP traffic (e.g., [10]) and match request content to known attribute values. But, this method cannot identify encoded, encrypted, or hashed attributes. Other techniques collect script domains [44] or function names [46] and compare them to known fingerprinting scripts. While this method helps to keep the false positive rate low, it struggles to detect new domains or script variations (e.g., minimized or obfuscated).

To overcome these limitations, we propose a detection technique based on *fine-grained taint tracking* (detailed in Section 4.2), which allows accurate identification of fingerprinting attributes sent to a remote server, regardless of the malicious Javascript code and destination domains.

Another fundamental flaw with existing taint tracking detection methods is that they lack robust classification techniques. That is, they miss an effective technique to measure the impact of fingerprinting scripts on users' privacy. A popular method (e.g., Li et al. [39]) considers every website that collects at least one sensitive browser attribute as a fingerprint. But this technique is too coarse-grained. The same applied to other classification techniques relying on heuristics [6, 7, 12, 22, 23, 36], such as the detection of canvas fingerprinting by measuring the dimensions or content of HTML canvas elements. To achieve more objective classification rules, two studies [19, 28] use machine-learning-based approaches for classification, but even these are trained using known fingerprinting scripts, which can lead to false negatives as the model would fail to classify novel fingerprinting methods.

The lack of an adequate classification technique motivates the need for a better way to measure how severe a browser fingerprinter is. For instance, we must distinguish when a fingerprint has high or low entropy (i.e., high or low distinguishing power). Previous studies collected fingerprints of web users employing either dedicated websites [20, 35, 41] or by embedding fingerprinting code into real websites [26]. These studies computed the entropy (see Section 2.2) of *individual* fingerprinting attributes and provided a quantitative measure of their privacy impact. However, they fail to address two issues. First, they do not estimate how common those attributes are in the wild. Second, individual entropy values are insufficient to measure a fingerprinter's privacy impact, which will tend to collect *multiple* attributes. To measure the information content of fingerprinters transmitting multiple attributes, we propose computing the *joint entropy* of all attributes (as described in Section 4.3) and perform multi-threshold classification to group fingerprinting behavior based on *real-world* patterns that we empirically observed in our large-scale experiments.

4.2 Collection via Fine-Grained Taint Tracking

When designing FP-tracer, we need to consider which granularity to apply over the tainted labels. Existing work [39] proposed a single label per JavaScript object. While this is appropriate for primitive objects (such as the `Number` type), it may lead to *over-tainting* and, therefore, false positives when considering `String` objects. For example, consider an array of strings, of which only one is tainted. The array can be serialized to a single string (using `join`) and then converted back to an array (using `split`). With object-level tainting, all elements in the array will now be tainted. Instead, we adopt a *fine-grained* approach that allows us to store distinct taint labels for each character in a given string instance. This also allows us to log which parts of a string entering a sink are tainted and provide deeper insights into how the attributes were manipulated or obfuscated before transmission (see section 6.6).

For instance, consider the code snippet presented in listing 1. The script computes an MD5 hash derived from aggregating three attributes: `screen.height`, `screen.width`, and `Navigator.useragent`. Utilizing our methodology will successfully identify the last 32 characters of the URL as tainted. Consequently, our

system can successfully log this transmission, explicitly recognizing it as an aggregation of these three attributes.

Another novelty of FP-tracer is the structure of the tainted data labels. We choose labels $M = \{s_1, s_2, \ldots s_n\}$ as a set of sources, where s_i is information about a source instance containing the name of the API together with the location of the source (function name, script URL, line and position). Unlike other studies based on taint-tracking [39], which only track the attribute name, this allows us to accurately pinpoint the code location where the attributes were accessed.

The tainting labels are then propagated during JavaScript operations, such as string concatenation or arithmetic operations. For unary operations (e.g., incrementation), we simply copy the label, while for arithmetic operations (e.g., addition), we propagate a label containing the union of all sources (e.g., $M = M_1 \cup M_2$). During string manipulation, we also ensure that the character level granularity of labels is preserved during operations such as string concatenation or splitting.

FP-tracer [3] utilizes *62 sources* and *25 sinks* that we carefully selected. They are listed in Tables 8 and 9 and Table 10 respectively. We define a sink as any JavaScript API that sends tainted data to an external party. This includes, for example, the `src` attribute of `img`, the `script` tag, the data sent via `fetch`, and `XHR` request URLs, headers and bodies. When a sink is called with tainted data (i.e., data label sets are not empty) we log the value of the string flowing into the sink, the locations of tainted characters in the string, and the corresponding set of fingerprinting attributes.

In addition, we log information about the various scripts involved in the fingerprinting process. First, we record the URLs of scripts responsible for accessing the fingerprinting attribute APIs (known as *source scripts*). Secondly, we register the script URL responsible for making the network API call (the *sink script*). Finally, we extract the URL of the network request made by the sink to save the fingerprinting attributes' final destination. This granular logging capability allows us to *distinguish* between *first-party* and *third-party* fingerprinting activities.

4.3 Classification via Joint Entropy Thresholds

FP-tracer employs the *joint Shannon entropy* to accurately measure fingerprint information content (see Section 2.2 for the relevant background). Given a fingerprint composed by a set of attributes $X_1, \ldots, X_n$ sent to a particular destination domain. We compute the joint Shannon entropy (i.e., total information transmitted to that domain) as:

$$H(X_1, \ldots, X_n) = \\ - \sum_{x_1 \in \mathcal{A}_{x_1}} \cdots \sum_{x_n \in \mathcal{A}_{x_n}} P(x_1, \ldots, x_n) \cdot \log_b P(x_1, \ldots, x_n), \quad (5)$$

where $P(x_1, \ldots x_n)$ is the joint probability of a particular combination of attribute values.

To calculate the joint entropy, we require the set of probabilities $P(x_1, \ldots x_n)$ for each set of attributes. This information must be obtained by collecting per-attribute fingerprinting data from a representative sample of real web users. Such a dataset is, therefore, an input for our methodology. While in section 5.3 we describe an implementation using a dataset collected by means of an Ad-fingerprinting technique, other datasets collected with other techniques could also be used (e.g., [20, 26, 35, 41]).

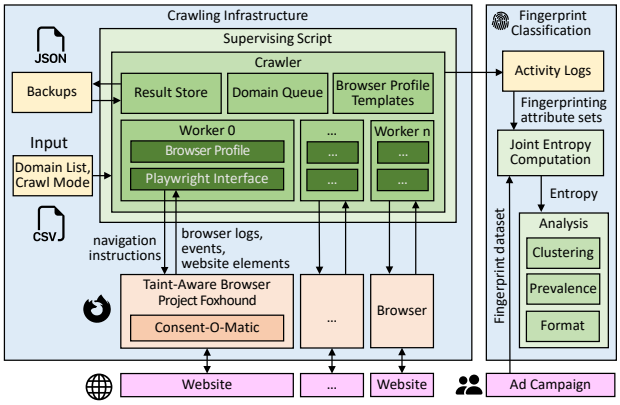

Figure 2: Crawler high-level architecture.

Our joint entropy approach considers the *correlation* between fingerprinting attributes, unlike widespread classification techniques, which treat each attribute in isolation or count them [10, 39]. As shown in Section 5.3, a small set of attributes can have a larger joint entropy (i.e., be more effective for fingerprinting) than a big attribute set. For instance, an attribute vector, which comprises six aggregate attributes `navigator.maxTouchPoints`, `navigator.appName`, `navigator.doNotTrack`, `navigator.product`, `navigator.platform`, and `navigator.vendor`, does not exhibit strong discriminatory power when compared to another attribute vector composed solely of `userAgent`. In addition, certain attribute combinations may have strong correlations. For instance, the attributes `screen.width` and `screen.height` are correlated due to physical computer monitor constraints. The joint entropy ensures that all of these correlations are taken into account in a statistically robust manner.

To take full advantage of joint entropy, FP-tracer classifies browser fingerprinting using several thresholds, extracted from the data using a clustering algorithm, such as Jenks Natural Breaks [30] and K-means clustering [40]. Each cluster groups attribute vectors by the invasiveness of the fingerprinter. In Section 6, we show how to use our classification technique to build a detailed picture of fingerprinting activities of real-world fingerprints in the wild. For instance, we measure that a first-party or third-party fingerprinter can be included in multiple clusters.

5 IMPLEMENTATION

We implemented FP-tracer (presented in Section 4) following the architecture shown in Figure 2. Specifically, we: (i) extended Foxhound, a taint-aware fork of Firefox, to collect data flows related to fingerprinting attributes; (ii) integrated our instrumented browser into a crawling infrastructure that we created to automate and scale our browser fingerprinting experiments; (iii) developed the tools to classify a fingerprinting activity based on the tainted attributes' joint entropy and verify their uniqueness using a real-world dataset from an advertising campaign. We now describe these three implementation aspects in more detail.

5.1 Foxhound Extensions

Our instrumented browser FP-tracer extends on a modified version of *Foxhound* [50] (v96.0.3), a fork of Firefox that can perform taint tracking of strings. Foxhound had been used to detect client-side security vulnerabilities [32, 33] by tracking data flows from user-controlled data sources to sinks, resulting in HTML parsing or code execution. Foxhound implements taint tracking by instrumenting string types in Firefox's JavaScript engine (SpiderMonkey) and Browser Engine (Gecko) to include taint information. To enable fine-grained fingerprinting activity detection for Foxhound, we extended its instrumentation engine with *four* features:

Numeric Type Support. One key contribution was the extension of Foxhound to support taint tracking for number types. This is essential to accurately track the usage of fingerprinting attributes, many of which are numbers. For example, fingerprinting scripts often collect and aggregate the user's screen width and height as seen in listing 1. In JavaScript, numbers are commonly represented as primitive types, where the numeric value is stored directly on the stack in a double-precision floating-point format. Alternatively, JavaScript also allows the creation of `Number` objects[1], which are ordinary objects consisting of a single internal slot holding the `Number` value.

We introduced the `TaintableNumber` built-in type, which extends the `Number` object with a second internal slot to store the taint tracking information. We then ensured that any sources emitting numbers would return TaintableNumber types with the appropriate label. We enhanced the JavaScript interpreter to ensure the taint information is propagated during arithmetic and binary operations (e.g., `+`, `-`, `*`, `/`, `+=`). In addition, any built-in functions converting between string and number types (e.g., `toString` and `parseInt`) are also instrumented to pass the taint information during the conversion.

Multiple Source Support. Foxhound stored information about the source of tainted data as a pointer to an object. In other words, a particular string character can only be associated with a single source. This is insufficient after introducing support for number tainting, as numbers can have multiple sources related to them. This required us to extend Foxhound's taint representation to enable storage of a *set of associated sources*, rather than a single one. In addition, we also instrumented implementations of numerical operations in the interpreter to compute the union of source sets as required by section 4.2.

Source and Sink Extensions. As Foxhound was previously used to detect client-side security vulnerabilities, we also adapted the source and sink APIs for the fingerprinting use case. In total, we label *62 sources* associated with accessing fingerprinting attributes, selected from relevant studies [39] and fingerprinting libraries [24]. We also identified *25 sinks*, supporting any JavaScript API call enabling data transfer from the client to a remote server. A detailed list and comparison with related work can be found in Table 8 and Table 10, respectively. To support such a large number of new sources and sinks in a scalable manner, we adapted Firefox's WebIDL[2] code generator to automatically insert C++ code to set and log taint instrumentation appropriately. This reduces the effort to add new sources or sinks, which can now be done by simply adding a custom `Taint` attribute to the appropriate property in the WebIDL file.

Activity Logging. When a tainted value is detected entering a sink API, Foxhound triggers a custom event with the following information: the value of the data entering the sink and a list of tainted character ranges. We extended the implementation to record the associated sources (and hence fingerprinting attributes) for each range, including the relevant source name, location (script URL, line, and position), and function name. Finally, we record the sink name, location, and function name, together with any additional information required to reconstruct the destination URL of the external request.

5.2 Crawling Infrastructure

To scale our fingerprinting collection technique to a large number of websites, we integrated our modified Foxhound browser from section 5.1 into the Playwright browser automation framework [42] (v1.21.1). We then created a node.js crawler application for automated website browsing. As presented in Figure 2, our crawler takes a list of target domains as input, which are added to a queue. A scalable number of workers manage the queue, each using a Playwright-controlled browser to visit websites and log fingerprinting activity.

Our crawler implements an *automatic* and *re-usable* website visiting logic. We visit each target website's top-level (home) page, followed by visits to three subpages chosen randomly from hyperlinks on the home page. The duration of the visits to a page is carefully structured: the home page loads first, and the crawler waits for 20 seconds with a mid-point scroll to the bottom. As shown by previous work [12], fingerprinting scripts may first be active once the user navigates to secondary pages. As such, we subsequently revisit the home page, followed by the three randomly selected secondary pages. The crawler waits 10 seconds on each page with a mid-point scroll. A transition to `about:blank` and a 5-second pause occurs between each page to ensure a clear separation of measurement data.

The crawler aggregates activity logs from multiple workers into a single output and allows backup and restoration of the crawling state in the event of expected errors. We log all fingerprinting-related dataflows for each web page visited during the crawl. We then aggregate this information to build the attributes sent to a particular destination domain from all data flows detected on the crawled domain (referred to as a *domain-destination pair*). The attributes are then used as input for the subsequent analysis stage.

5.3 Fingerprint Classification

To compute the joint entropy of each fingerprinting attribute set, we calculate $P(x_1, \ldots, x_n)$ from eq. (5) using values obtained from real web users. In particular, we employ a dataset of user fingerprints obtained from a related research study [14] utilizing a technique known as *Ad Fingerprinting*, whereby dedicated JavaScript code is embedded into online advertisements. The advertising campaigns were executed within the Sonata Platform, a Demand Side Platform (DSP) operated by TAPTAP Digital [53]. Sonata is a mid-sized DSP

[1] https://tc39.es/ecma262/multipage/numbers-and-dates.html
[2] https://webidl.spec.whatwg.org

Table 1: H_n values of selected fingerprinting attributes.

Attribute	[35]	[26]	FP-tracer
User Agent	0.580	0.341	0.565
Do Not Track	0.056	0.091	0.066
Content language	0.351	0.129	0.313
Screen Resolution	0.290	0.231	0.572
Canvas Drawing	0.491	0.407	0.654

that delivers millions of daily advertisements in 15 countries across Europe, North America, South America, and Africa. On loading the advert, the embedded JavaScript code will collect fingerprinting attribute values from the browser and report them back to a dedicated server. Furthermore, the device advertising ID provided by the DSP was also collected as a feature. As a result, a fingerprint is considered truly unique in the dataset if all the samples with a specific *device fingerprint* value share the same Advertising ID. This process enables the establishment of a *ground-truth uniqueness* within the dataset and, thus, defines the probabilistic variance of fingerprints based on different attribute values that make up each fingerprint of the devices in the dataset.

The resulting dataset from these measurements comprises approximately 161k fingerprint samples collected at various time intervals (February-June 2022, May 2023, and September 2023) and from diverse browser instances across different device configurations. Each fingerprint in this dataset is characterized by a combination of device type (mobile or desktop), operating system (Android, iOS, Windows, macOS, or Linux), and browser (Chrome, Safari, Firefox, Edge, or MiuiBrowser). As FP-tracer is implemented as a desktop browser, we filter the samples to remove fingerprints from mobile browsers, which may bias our results. This leaves a total of 85 576 fingerprints from desktop browsers. In Table 1, we show the normalized entropy values as measured by previous work [26, 35] and us for five attributes typically collected by browser fingerprinting scripts.

The collected fingerprint dataset is then used to calculate a (normalized) joint entropy for each unique set of attributes collected during the crawling stage described in section 5.2. In addition, we also calculate the mean anonymity set size for each set to assess how unique a particular set is within our dataset (cf. section 2.3). The entropy values allowed us to perform a detailed analysis of the state of fingerprinting on the web, as presented in section 6. Our fingerprinting classification and analysis framework was implemented in Python using a series of dedicated Jupyter notebooks.

5.4 Ethics Considerations

The paper's experiments have obtained the approval of our institution's Institutional Review Board (IRB) via the relevant Data Protection Officer (DPO). Moreover, we made sure to have specific considerations to protect users' privacy. For example, we limit our fingerprint collection to users with an existing advertising ID and do not process data from devices with the `do-not-track` flag active in their browsers.

We also made sure our experimental setup complies with best practices for crawling. For instance, we added custom HTTP headers to identify our crawl as an academic study and hosted an explanatory website with contact and opt-out information on our crawler's IP address. In addition, all fingerprinting takes place inside the browser, with minimal disruption for visited sites.

6 EVALUATION

In this section, we present FP-tracer's evaluation setup. Then we discuss results related to its fine-grained collection capabilities in Section 4.2 and multi-threshold joint-entropy classification ones in Section 4.3.

6.1 Setup and Perfomance

We performed our evaluation using the 100k most popular websites according to the Tranco List [37] (version N7QVW [38]). We tried to visit each domain using HTTPS, and if it failed, we used HTTP or prefixed the domain with `www`. Our crawling server is based in Europe, and therefore, the websites we visit must comply with the General Data Protection Regulation (GDPR). We started our crawl on 31/03/23 and ran it for 18 days using up to 12 parallel workers on a CentOS Linux server with 16 CPUs and 32GB of RAM. To evaluate that additional overhead caused by FP-tracer, we ran dedicated experiments as described in appendix A.1. We measured an overhead of $11 \pm 1\%$ compared to an unmodified Firefox browser, which is comparable to related work (e.g. 9% [39]). However, our implementation is more advanced, as it can track taint labels at character level, compared to object-level tracking of Li et al. [39].

Of the 100 000 domains, we were able to visit 80 618 of them. Of the unsuccessful domains, we found around 15 000 were unreachable, with the remaining 4319 provoking crawler errors such as timeouts.

6.2 Collection and Classification

We detected around *6.8 million tainted strings* from 33 sources (see table 11) and 15 sinks (see table 12). This corresponds to a collection of 7.4 million transmitted attributes as a single string can be labeled with multiple attributes. The collected flows comprise 15 239 combinations of fingerprinting attributes. For each set of attributes, we computed the (normalized) joint entropy as described in section 5. We find fingerprinting activity collecting up to 27 attributes, with a maximum entropy of 15.9 bits and a maximum normalized joint entropy of 0.989.

Figure 3 shows a 2-dimensional histogram presenting the relationship between the joint entropy and the number of attributes collected in each case. The plot also shows the mean entropy per attribute count and data points from known fingerprinting libraries. Interestingly, the entropy starts to flatten off (i.e., reaches maximum curvature) after collecting just seven attributes. Figure 3 also shows that counting attributes is a poor metric for classifying fingerprinting activity. For a given attribute count, there is an extensive range of joint entropies depending on which attributes are collected.

Figure 4 shows the correlation between the joint entropy and the mean anonymity set size (see eq. (4)) for each of the 15 239 attribute combinations. As expected, increasing joint entropy corresponds to a logarithmic decrease in the anonymity set size.

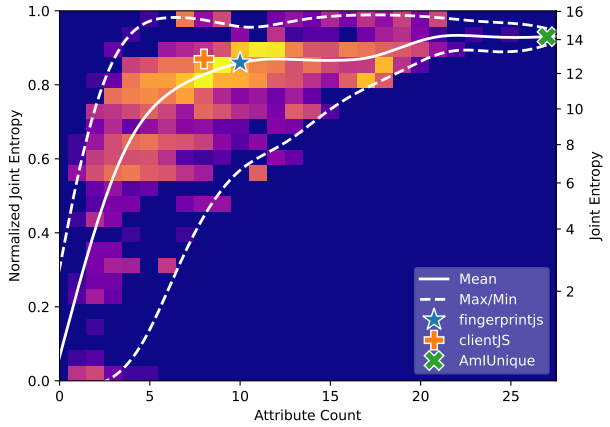

Figure 3: Normalized joint entropy vs. number of attributes. The color scale indicates the number of attribute combinations in each bin from low (blue) to high (yellow).

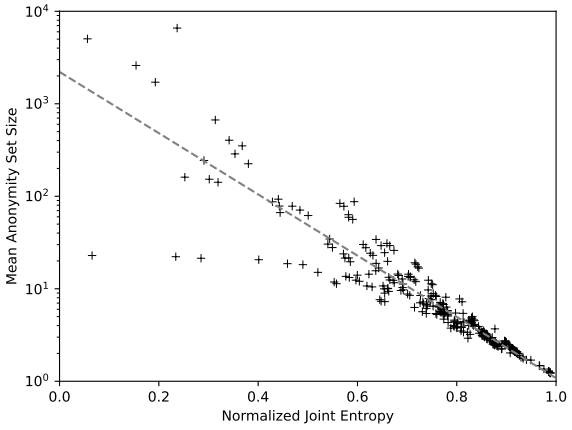

Figure 4: Normalized joint entropy vs. anonymity set size.

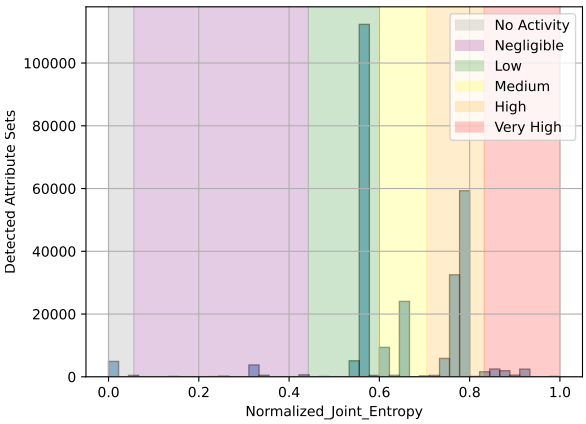

Figure 5: Detected tainted attribute sets per normalized joint entropy cluster.

Table 2: Entropy ranges clustering.

Set	Entropy Range	$\bar{A}$	Prevalence
Negligible (1)	(0.000, 0.154)	1600	0.68%
Negligible (2)	(0.154, 0.442)	2000	5.25%
Low	(0.442, 0.596)	32	58.99%
Medium	(0.597, 0.705)	15	30.94%
High	(0.705, 0.832)	5.1	61.49%
Very High	(0.832, 0.989)	2.6	8.08%

Table 3: Joint Entropy of selected attribute sets.

Attribute sets	H_n	Label	$\bar{A}$
userAgent, storageEstimate	0.907	Very High	2.0
userAgent, canvas	0.743	High	12
canvas	0.654	Medium	34
userAgent	0.565	Low	350
maxTouchPoints	0.154	Negligible	2600

Figure 5 shows the normalized joint entropy of the tainted attributes we collected during our experiments. We calculated the joint entropy for each domain-destination pair as described in Section 5.2. After this aggregation step, we find a total of 269 784 transmissions between crawled and destination domains. We bin the values using six intervals: No Activity, Negligible, Low, Medium, High, and Very High. We set the thresholds using the Jenks natural breaks algorithm. We also performed clustering using the K-means algorithm and found only minimal differences between the intervals obtained.

We find some activity below the normalized entropy (see eq. (5)) of $H_n < 0.5$, with very distinct structure and peaks between $0.5 < H_n < 0.8$, followed by a tail up to 0.989. Note that there is some activity with $H_n = 0$, which, interestingly, is caused by scripts transmitting deprecated attributes such as `Navigator.appName`, which will always return the string `Netscape`.

We find six classes of fingerprinting activity, as summarized in table 2. Each class is assigned a descriptive label, from "Negligible" to "Very High". The groups are also shown as colored bands in fig. 5 and can be seen to select different distribution features successfully. We also provide two additional metrics in table 2 to provide a more intuitive measure of how well each class can distinguish users. The mean anonymity set size $\hat{A}$ shows that fingerprinting in the "Very High" cluster will be shared between 2.6 users on average in our dataset. On the other hand, "Negligible" fingerprints will be shared with over one thousand different users.

Table 3 lists some examples of attribute sets in each cluster. For example, "Low" fingerprinting can be achieved by collecting just

the `userAgent` attribute, whereas "Medium" requires the canvas element. Collecting the two attributes together results in "High" activity. Note that "Very High" classes can be achieved with as little as three attributes. Notably, we also found that the fingerprinting libraries Client-JS (entropy: 0.87) and FingerprintJs (entropy: 0.86) both fall into the "Very High" cluster, together with the set of attributes collected by the Am-I-Unique website (entropy: 0.93).

6.3 Prevalence on the Web

We also measured how often each class of fingerprinting activity occurs on the web. To do this, we take each cluster and measure the fraction of crawled domains where at least one transmission occurs with entropy in that cluster. The results are shown in the final column of table 2. Interestingly, we detect significant fingerprinting activity across all fingerprinting classes. The exception is the "Negligible" cluster, which we will no longer consider in the remainder of the analysis.

Our results show that almost 59% of websites perform fingerprinting activity within the "Low" cluster, with fewer domains (31%) performing "Medium" fingerprinting. More concerning is the fact that 61% of domains are found in the "High" cluster, indicating they can uniquely identify users in populations of $\approx$ 17 000. In the worst case, we find 8% of domains where "Very High" fingerprinting occurs.

In fact, the results in table 2 may be able to explain the discrepancy in previous results. Studies measuring very high rates of $\approx$ 70% [10, 39], do so by detecting the presence of at least one fingerprinting attribute. Considering the fraction of domains that perform at least a "Low" level of activity yields a result of 75%, which is broadly in agreement with previous work. On the other hand, lower rates of $\approx$ 10% [12, 28] are measured when using stricter classification techniques, which is in agreement with the 8% measurement in the "Very High" cluster.

This observation leads to a more general insight that browser fingerprinting cannot be considered as a binary property of websites; rather, it encompasses a broad range of activities with different privacy impacts. What is traditionally referred to as fingerprinting would typically fall into the "Very High". Lower entropy clusters (e.g., "High") may not be able to target a single user but could distinguish groups of users for analytics and marketing purposes.

6.4 Destination Categories

In order to investigate this in more detail, we examine the categories of domains in each of our fingerprinting clusters. To do this, we first divide our results into two categories: first-party fingerprinting (where the destination domain matches the crawled domain) and third-party (otherwise). We then group each domain based on the website category as reported by the Webshrinker service [57].

Table 4 presents the fractions of crawled domains performing first-party and third-party fingerprinting, broken down into clusters. For each cluster, we also list the top three categories, together with the top three destination domains in the case of third-party activity. The results show some interesting insights. For example, at the "Low" activity level, first-party domains display a prevalence rate of 34%, while third-party domains show a notably higher rate of 50%. The latter predominantly pertains to Business-related activities (40%). In the "Medium" activity level, first-party domains exhibit a low prevalence rate of 2.2%, mainly concentrated on Technology & Computing destinations. Conversely, third-party domains show a much higher prevalence rate of 30%, with a focus on Web Search engines and marketing domains.

Moving to the "High" activity level, first-party activity manifests a prevalence rate of 12%, emphasizing Technology & Computing. On the other hand, third-party domains have a substantially higher prevalence rate of 58%, majorly constituted by Marketing-labeled domains (45%). Similarly, at the "Very High" activity level, first-party domains demonstrate a prevalence rate of 2.8%, whereas third-party domains exhibit a rate of 5.8%.

In summary, we find that fingerprinting is carried out by both first and third-party domains across all of the clusters in our analysis, with third-party activity more common overall. Fingerprinting is overall pervasive, occurring across a wide range of website categories.

6.5 Common Sources and Sinks

Our analysis shows that attributes like `audiocontext`, `audioNode`, and HTML canvas elements are strong contributors to high joint entropy. Indeed, in the high entropy category, we found audio context fingerprinting on 54 crawled domains and canvas fingerprinting on 6237. Our analysis found that the attribute `storageestimate.quota` collected as part of a high entropy vector has a strong distinguishing power (i.e., high entropy). We detected `storage.quota` in 148 domains. The detailed list of active sources that we detected, together with the number of transmissions per entropy level, is provided in table 11 within the Appendix.

On the other hand, the analysis of active sinks revealed that medium entropy vectors were mainly transmitted via `img.src`, `navigator.sendBeacon` APIs and XHR requests. Meanwhile, High entropy collections mostly used XHR headers and URLs, as well as `iframe`, `img`, and `script src` attributes. Details are in table 12.

6.6 Aggregation and Obfuscation

Our fine-grained tainting approach allows us to survey the types of aggregation and obfuscation techniques performed by fingerprinting scripts in the wild. To do this, we examined the content of the 6.8 million strings that are transmitted through sinks in our dataset. We then extracted the tainted substrings and their associated fingerprinting attribute labels.

We find that substrings are associated with a single fingerprinting attribute in 95% of cases. In these cases, we were able to compare the string content to the expected value given by our browser implementation. For example, `navigator.platform` will have an expected value of "Linux x86_64". As shown in table 5, we found that in 85% of these cases, the strings are transmitted in plain text, either in their entirety or as a substring of the attribute. This fraction drops to 47% when considering strings that are collected as part of a "Very High" transmission. This indicates that more harmful fingerprinting activity tends to employ obfuscation techniques such as encoding or encryption. Notably, this implies that techniques using network API or traffic monitoring will fail to detect over half of fingerprinters. In cases where strings were not apparent as

Table 4: Category of first-party domains and third-party destinations performing fingerprinting.

	First party		Third party		
	Prevalence	Top 3 Categories	Prevalence	Top 3 Destination Categories	Top 3 Destinations
Low	34%	Technology & computing (24.8%) News/weather/Information (15%) Business (9.5%)	50%	Business (40%) Technology & computing (30%) Web Search (11%)	doubleclick.net (19346) google.de (9266) google.com (8311)
Medium	2.2%	Technology & computing (18%) Education (15%) News/weather/Information (12%)	30%	Web Search (25%) Marketing (20%) Technology & Computing (18%)	google.com (7525) google-analytics.com (6230) baidu.com (1908)
High	12%	Technology & Computing (22%) News/weather/Information (20%) Business (8.8%)	58%	Marketing (45%) Business (29%) Non-standard content (9.3%)	google-analytics.com (38294) doubleclick.net (13614) youtube.com (6746)
Very High	2.8%	Technology & computing (16%) News/weather/Information (13%) Business (9%)	5.8%	Technology & computing (34%) Uncategorized (30%) Business (9%)	webgains.io (767) baidu.com (233) datadome.co (233)

Table 5: Summary of fingerprinting encoding.

Format	Low	Medium	High	Very High	All
Plain	77.5%	91.0%	87.6%	47.3%	84.5%
Encoded	1.0%	0.9%	0.9%	6.4%	1.2%
Hashed	0.0%	0.1%	0.5%	0.3%	0.4%
Other	21.4%	8.0%	11.1%	46.0%	14.0%

plaintext, we found that 1.2% strings were transformed using well-known encoding techniques (e.g., Base64, Hex, or URL encoding). In addition, we could identify 0.4% of strings that were hashed using common algorithms (e.g., MD5, SHA1, or SH256). Of the remaining cases, we performed a manual inspection of frequently occurring scripts and found a wide range of custom hashing, encoding, and combination techniques. Some examples of such algorithms can be found in appendix A.2.

We also examined the 5% of strings that are tainted with two or more attributes. A simple example is line 7 of listing 1, where two numbers are multiplied so that the result will be tainted with both the `height` and `width` attributes. We found that multiplication and addition of the screen dimensions were responsible for almost 8% of these aggregated transmissions. For the remaining cases, we assessed whether the string value obtained was constant for a given combination of attributes and script domain. We found that 30% of values were unique, implying that these scripts could be used to identify a user across multiple websites. The remaining cases produced different values across crawled domains due to the combination of additional variables such as random numbers or the website URL.

6.7 Collaborative Fingerprinting

We discovered that multiple scripts loaded from various domains can participate in the same browser fingerprinting activity. For instance, if two scripts are loaded from two different domains, but ultimately, they collect fingerprinting attributes and share them with the same destination, then we refer to this as collaborative fingerprinting. Our granular tracking methodology allows us to detect such behavior.

This phenomenon is represented in fig. 6, which provides a heatmap of the 269 784 attribute vectors per number of domains serving scripts that transmit attributes to a single destination. The heat map also shows activity split by entropy level, thus representing the privacy impact of the final constructed vector. While the majority of activity is carried out by scripts loaded from a single domain, we find significant activity from two or more domains, especially in the High entropy category where 38.34% of transmitted vectors involved more than just 2 domains. Furthermore, in two extreme instances, we found scripts loaded from 7 domains, all of which transmitted attributes to the same destination.

For example, in 10 of the crawled domains, `baidu.com` received a vector of seven attributes, namely: the screen `colorDepth`, `width` and `height`, and the navigator `language`, `hardwareConcurrency`, `platform`, and `userAgent`. This vector has a very high joint entropy of 0.85 and is assembled from two vectors collected by different domains. Specifically, a script from `baidu.com` gathers four attributes with a joint entropy of 0.65, while the remaining three attributes with a joint entropy of 0.66 are collected by a script loaded from `bdstatic.com`.

Additionally, we observed that `globo.com` gathered a high joint entropy (0.9) vector consisting of 8 attributes, namely: `useragent`, `platform`, `width`, `height`, `colordepth`, `canvas`, `language`, and `hardwareconcurrency`. Among these attributes, `chartbeat.com` collected 5 attributes of 0.83 and `insurads.com` collected 4 attributes of 0.76 joint entropy, while `useragent` was collected by both. This observation suggests that `globo.com` may be utilizing both domains as services to gather fingerprinting attributes.

We cannot comment on the intent behind every collection utilizing multiple domains presented in fig. 6, as we lack visibility on the remote server side. Nevertheless, we highlight the potential for abuse of such capabilities by more sophisticated fingerprinting actors. In the provided example, Baidu uses `bdstatic.com` as a service to receive extra attributes. Other fingerprinting actors could

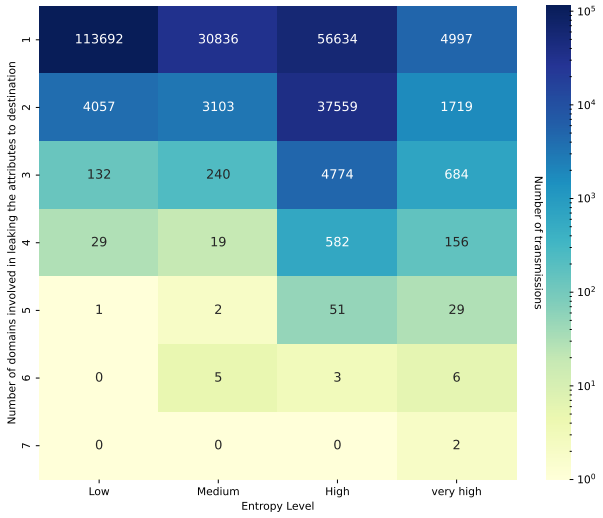

Figure 6: Number of distinct detected involved domains in fingerprint collection sent to the same common destination.

leverage multiple fingerprinting vendors to increase their fingerprinting capabilities such as presented in the example of `globo.com`.

6.8 Third-party Scripts

We observed that 1.26 million scripts were sending attributes through a sink-supported function, with approximately 1 million being loaded from third-party domains. Our analysis focuses on identifying the most commonly used ones that exhibit varying behaviors across different domains, indicating a configurable collection of attributes. In table 13, we present the most notable scripts in this regard, as they were found to leak more than 20 different attribute combinations across multiple domains. One notable example is Google Analytics' `analytics.js`, which was detected in a total of 38.6K domains and was found to leak 28 different attribute sets in total.

7 DISCUSSION

In this section, we discuss further insights into the cross-validation conducted against the state of the art, and the results of our secondary crawl evaluating user banner impact.

7.1 Disconnect and EasyPrivacy Lists Validation

We validate our findings using the Disconnect [1] and the EasyPrivacy [2] lists. These provide lists of domains that are suspected of performing tracking activity and are commonly utilized by ad blocker services and browser plugins. While EasyPrivacy only provides a list of tracking servers, Disconnect deserves more explanation as it provides several interesting categories. Namely, it categorizes services conducting fingerprinting into two severity categories: *FingerprintingInvasive* (FI) and *FingerprintingGeneral* (FG).

Table 6: Validation against Disconnect and EasyPrivacy lists.

	Disconnect [1]		EasyPrivacy [2]
Level	**Invasive (FI)**	**General (FG)**	**Tracking servers**
Low	10	35	187
Medium	6	42	134
High	20	473	100
Very high	26	17	131

Under FI, they individually assess each service's fingerprinting activity using various techniques. For instance, they use results from API monitoring studies to identify access to sensitive fingerprinting attributes, like canvas elements or audio context. In addition, they identify the presence of specific fingerprinting libraries, such as fingerprints. On the contrary, services labeled as FG are categorized based on investigations into the privacy policies of these services.

In table 6, we list the number of domains found on the respective lists, split by joint entropy level. If we find a domain performing fingerprinting in multiple categories, we always choose the highest level as input for the Table. Overall we find a good agreement with our results and the two lists. For example, we can detect trackers from the EasyPrivacy list across all entropy levels. Similarly, we found 20 trackers from the Disconnect list in the High cluster and 26 in Very High. However, we also measured 10 trackers in the Low entropy level, which were labeled as FI by Disconnect. We manually inspected these domains and found that they were categorized as invasive by Disconnect due to the presence of a FingerprintingJS script or the detection of canvas fingerprinting. Cross-checking with our results, we only collected lower entropy attributes such as screen dimensions or `userAgent`.

This discrepancy may be due to the difference in methodology used by our technique compared to the Disconnect list. For example, our results will only provide a lower bound on the fingerprinting activity of a given domain. This is due to the dynamic nature of our detection technique: we only detect fingerprinting if the responsible code is executed. Increasing the number of crawled domains as well as the number of visited subpages could be an easy way to mitigate this discrepancy.

More interestingly, our results detected the majority (473) of domains in the High category that were only labeled as FG by Disconnect. Furthermore, 17 domains labeled as FG by Disconnect were found by our approach to conduct a "Very High" entropy level of collection. For instance, we found doubleclick.net to conduct both canvas or audio fingerprinting and googlesyndication.com to conduct canvas fingerprinting. However, both of these domains are listed by Disconnect (under Google) as General fingerprinting domains based on the privacy policy of the services. Despite this, we found them to both collect very high entropy attribute combinations. We plan to share our results with Disconnect with the ultimate goal of contributing to their classification.

7.2 Validation with FP Inspector

We also compared our results with the most recent study by Iqbal et al. [28], which makes their crawling results publicly available.

They proposed FP-Inspector, a classification technique using machine learning to infer fingerprinting from JavaScript analysis. This technique detected fingerprinting activity on 10.18% of the top 100,000 Alexa websites, with their data publicly available [29]. Out of their results, we found a total of 911 domains, which we also crawled during our study. Despite the volatile nature of the web and a 2-years gap between the studies, we found a good agreement with their findings. We found that 94.52% of the common domains have fingerprinting activity in the Medium category or higher.

7.3 Consent Banner Impact

We investigated the impact of consent banners on fingerprinting activity. A consent banner enables a website to let the user provide consent regarding its data collection policies. To this end we performed two additional crawls: (i) we performed a filter crawl to detect the presence of consent banners by crawling the Tranco top 100K with the Consent-O-Matic plugin [4, 5, 45]. (ii) we crawled the filtered domains using three profiles using the methodology described in section 6.1. The first profile provides the baseline, as no interaction with the consent banner is attempted. The second one uses Consent-O-Matic to interact with the banner and consents to all data collection. The third profile tries to reject as much data collection as possible. For each profile, we collected a separate dataset of dataflows, computing entropy computation, and clustering as in section 6.

Table 7 summarizes our consent banner results. We visited 4860 domains, with the fraction of domains with various levels of fingerprinting activity shown in table 7. The baseline results agree with those from the main crawl and show that websites with consent banners in our sample are representative of the web at large. In addition, we find that accepting all data collection purposes produces a surge in activity across the “Low”, “Medium”, and “High” clusters, registering increases in domain activity of 18.50%, 13.04%, and 20.14%, respectively. Intriguingly, the “Very High” cluster displayed only a slight rise of 1.05%, indicating a complete disregard for user consent for this collection level. We also noticed slight increases in activity even if data collection is rejected, specifically notable for the “Low”, “Medium”, and “High” groups, which show increases in activity of 4.61%, 2.3%, and 2.91%, respectively. This suggests that despite the user rejecting data collection, some entities still conduct fingerprinting disregarding the user’s consent. Our results demonstrate that the baseline fingerprinting rate is high even before a user conducts any interaction with a consent banner. As expected, fingerprinting activity increases to almost 90% of domains after accepting all data collection, interestingly the very high group is the least likely to respect the user choice as its baseline prevalence is almost identical before and after the user’s consent it given. Finally, we also measure a slight increase in activity events if data collection is explicitly rejected.

7.4 Limitations

Although our methodology is robust, it does have certain limitations. For example, while FP-tracer can track explicit data flows, implicit data flow tainting is not supported. Implicit data flows occur when variable assignment is conditional on specific values of tainted data. Tracking implicit flows introduce significant runtime overhead, as demonstrated by Staicu et al. [52]. Additionally, the same study argued that for detecting security vulnerabilities, supporting implicit flows was not particularly useful. While we cannot make the same claim for browser fingerprinting as the context is different, we decided to make the trade off to only support explicit flows.

Our implementation does not support tracking of dataflows which involve execution of WebAssembly code. Future work could enhance our implementation with the specific scope of measuring the false negative impact of implicit-related flows or WebAssembly-related flows.

Despite the broad coverage of instrumented sinks and sources by FP-tracer, as detailed in the Appendix Tables 8 and 10, there are APIs that either lack support in FP-tracer or are simply absent from Firefox. Based on the limitations stated, we suggest that our results should be viewed as a lower-bound estimation of the real browser fingerprinting occurring on the web.

It’s also important to note that the choice of fingerprinting dataset influences the entropy calculation in our methodology. We attempt to reduce this bias by collecting a data from a representative sample of real web users, as opposed to dedicated fingerprinting collection sites [35] or a single site [26].

8 RELATED WORK

Entropy-based classification. Several studies attempt to measure the entropy of fingerprinting attributes based on populations of web users [20, 26, 35, 41]. In general, they tend to compute the entropy of single attributes in their dataset as a measure of how effective they are at identifying users. These studies lack a meaningful evaluation of the privacy impact of attribute combinations that are used by real websites. We take this into account by accurately measuring the combinations of attributes sent to a particular site and computing the joint entropy for that particular combination. This also considers any correlations between attributes in a statistically robust manner.

Dynamic API monitoring. Several studies rely on JavaScript API monitoring using various techniques, including JavaScript injection, browser debugging interfaces, or browser source code modifications. Many studies use this technique alone [6, 19, 22, 23, 36, 47, 54] or as part of a hybrid approach [7, 12, 28] combining it with other classification methods such as machine-learning-based ones. However, this technique has limitations. It can detect if an attribute is accessed but not whether it is sent to a (third-party) server, potentially leading to false positives.

Fingerprinting static analysis. Other works utilize static analysis techniques to recognize fingerprinters from known characteristics, like script names. For instance, Papadogiannakis et al. [46] use a technique that queries the browser’s profile for running methods, assuming fingerprinting activity if known fingerprinting script method names are detected. However, static analysis can only detect known elements and is vulnerable to obfuscation.

Fingerprinting traffic analysis. Other studies rely on traffic analysis to identify fingerprinting attributes from network packets, as proposed by Al-Fannah et al. [10]. This technique analyzes website traffic but faces limitations when attributes are encoded, encrypted, or sent as a single visitor ID. Our results show that for the very high

Table 7: Prevalence Rates for Different Consent Modes and Fingerprinting Levels.

Consent Mode	Negligible		Low		Medium		High		Very High	
Baseline	7.8%		71.38%		27.3%		63.32%		9.55%	
Accept All	9.68%	+1.88%	89.88%	+18.50%	40.34%	+13.04%	83.46%	+20.14%	10.60%	+1.05%
Reject All	8.80%	+1.00%	75.99%	+4.61%	29.61%	+2.31%	66.23%	+2.91%	9.64%	+0.09%

entropy collection, only 47.5% of the attributes were transmitted in plain text, meaning that such techniques relying on traffic analysis would have a false negative rate exceeding 50% when it comes to detecting severe fingerprinting.

Dynamic flow analysis. Recent studies have embraced dynamic flow analysis for browser fingerprinting. The work by Sjösten et al. [51] introduced dynamic flow detection but faced challenges, crawling only 30 websites due to high overhead. They needed extensive time on each site (up to 12 hours) to ensure script execution. Recently, Li et al. [39] introduced FPFlow, a dynamic taint-tracking solution based on Chromium browser instrumentation. FPFlow addressed prior performance issues, enabling the crawling of 10,000 websites, but it falls short on several points. For instance, it supports only seven sinks, while our results show that at least fifteen sinks are used in the wild. Furthermore, FPFlow is not fine-grained as it implements taint tracking at the object level, which leads to over-tainting (i.e., false positives) and cannot track attribute aggregates. Unfortunately, an experimental comparison is not possible as FPFlow is, to the best of our knowledge, not available as an open-source tool. Finally, our approach enhances flow classification by leveraging joint-entropy analysis of detected vectors, distinguishing our study from [51] and [39].

Consent Banner. Recent works have analyzed the effect of consent banners on user privacy, focusing on areas such as analytics [31], tracking pixels [13], and security vulnerabilities [33]. Papadogiannakis et al. [46] attempts to understand the influence of user interaction with consent banners on web trackers, but only considers fingerprinting in passing. To our knowledge, we provide the first systematic study into the effect of consent banner interaction on fingerprinting activity.

Surveys. Recent applications of taint tracking to fingerprint detection [39, 51] either lack the accuracy to gain deep insights into fingerprinting [39] or suffer performance issues, which make them unsuitable for large-scale studies [51]. Based on a survey from 2020 [34], browser fingerprinting serves many purposes beyond targeted advertising or tracking. It encompasses diverse applications such as identifying device vulnerabilities, preventing bot and fraud activities, and facilitating augmented authentication. For instance, it assists in identifying vulnerable devices and can be utilized to inform and protect users with vulnerable systems. Moreover, fingerprinting's ability to detect unusual compositions or sudden changes aids in bot and fraud prevention. Lastly, it enhances authentication by identifying logins from new devices, making it more challenging for attackers to hijack active sessions. Other recent studies also have shown that adversaries can use stolen browser fingerprints to compromise security too, as they may allow to bypass security checks for 2-factor authentication [9, 25, 43, 49].

9 CONCLUSION

We introduce FP-tracer, a new methodology utilizing fine-grained taint tracking and multi-threshold joint-entropy classification to detect browser fingerprinting. Implemented through an extension of Foxhound, a privacy-centric Firefox fork, our approach is embeddable within our automated crawling infrastructure. We successfully crawled 80K of the top 100K domains listed in Tranco. Utilizing a representative dataset, we calculated joint entropy to classify fingerprinting trends via K-means and Jenks natural breaks clustering algorithms.

Our large-scale experiments unveiled four real-world fingerprinting trends, i.e., Very high, High, Medium, and Low. Very high fingerprinting was detected in 8.08% of domains, while Low, Medium, and High in 58.99%, 30.94%, and 61.49% of domains. Very high is the most severe category, which enables unique browser identification. High, Medium, and Low, while still invasive, may serve other purposes, such as targeted advertising or analytics. Moreover, we measure that third-party fingerprinting is more frequent than first-party, regardless of the trend.

FP-tracer uncovered valuable insights into actual browser fingerprinting practices. It discovered that 5% of transmissions involved attribute aggregation on the client side and exposed that as much as 53% attributes in the very high fingerprinting category are obfuscated. It revealed that 38.34% of attribute vectors in the Very high group loaded scripts from multiple domains and leaked vectors to the same destination, i.e., collaborative fingerprinting. Moreover, despite user consent, severe fingerprinting activity remained constant, emphasizing the need for more robust privacy measures and regulations. The prevalence of obfuscation techniques and the dominance of third-party fingerprinting underscored the necessity of refining detection methods.

ACKNOWLEDGMENTS

We thank our anonymous reviewers and shepherd for their constructive feedback. This work has been supported by the Apricot/ENCOPIA ANR MESRI-BMBF project (ANR-20-CYAL-0001). This work also received funding from the European Union's Horizon 2020 research and innovation program under project TESTABLE, grant agreement No 101019206 as well as by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2092 CASA – 390 781 972. Angel Cuevas acknowledges the funding for his contribution to the project ENTRUDIT (Grant TED2021-130118B-I00) funded by the MCIN/AEI/10.13039/501100011033 and the European Union NextGenerationEU/PRTR. Miguel Angel Bermejo's contribution received funding from Grant PRITIA-CLOUD from the UNICO-CLOUD program funded by the Ministry for Digital Transformation and Civil Service and the European Union NextGenerationEU/PRTR.

REFERENCES

[1] 2023. *Diconnect Entities.* https://github.com/disconnectme/disconnect-tracking-protection/blob/master/entities.json

[2] 2023. *EasyPrivacy.* https://github.com/easylist/easylist/tree/master/easyprivacy

[3] 2023. FP-tracer Artifact Release. https://github.com/soumboussaha/FP-tracer Accessed: March 11, 2024.

[4] Aarhus University Centre for Advanced Visualisation and Interaction. 2022. *Consent-O-Matic.* https://consentomatic.au.dk/

[5] Aarhus University Centre for Advanced Visualisation and Interaction. 2022. *Consent-O-Matic GitHub Repository.* https://github.com/cavi-au/Consent-O-Matic

[6] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. 2014. The Web Never Forgets. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA), Gail-Joon Ahn, Moti Yung, and Ninghui Li (Eds.). ACM, 674–689. https://doi.org/10.1145/2660267.2660347

[7] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. 2013. FPDetective. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security - CCS '13* (New York, New York, USA), Ahmad-Reza Sadeghi, Virgil Gligor, and Moti Yung (Eds.). ACM Press, 1129–1140. https://doi.org/10.1145/2508859.2516674

[8] Adam Horvath. 2012. *MurMurHash3, an ultra fast hash algorithm.* https://blog.teamleadnet.com/2012/08/murmurhash3-ultra-fast-hash-algorithm.html

[9] Ariel Ainhoren. 2019. *Digital Browser Identities: The Hottest New Black Market Good.* Technical Report. IntSights.

[10] Nasser Mohammed Al-Fannah, Wanpeng Li, and Chris J. Mitchell. 2018. Beyond Cookie Monster Amnesia. In *Developments in Language Theory*, Mizuho Hoshi and Shinnosuke Seki (Eds.). Lecture Notes in Computer Science, Vol. 11088. Springer International Publishing, 481–501. https://doi.org/10.1007/978-3-319-99136-8_26

[11] Nampoina Andriamilanto, Tristan Allard, Gaëtan Le Guelvouit, and Alexandre Garel. 2021. A large-scale empirical analysis of browser fingerprints properties for web authentication. *ACM Transactions on the Web (TWEB)* 16, 1 (2021), 1–62.

[12] Mohammadreza Ashouri. 2019. A Large-Scale Analysis of Browser Fingerprinting via Chrome Instrumentation. In *ICIMP 2019, The Fourteenth International Conference on Internet Monitoring and Protection* (Nice, France). IARIA. https://hal.archives-ouvertes.fr/hal-01811691

[13] Paschalis Bekos, Panagiotis Papadopoulos, Evangelos P. Markatos, and Nicolas Kourtellis. 2022. The Hitchhiker's Guide to Facebook Web Tracking with Invisible Pixels and Click IDs. https://doi.org/10.48550/arXiv.2208.00710

[14] Miguel A. Bermejo-Agueda, Patricia Callejo, Rubén Cuevas, and Ángel Cuevas. 2023. adF: A Novel System for Measuring Web Fingerprinting through Ads. arXiv:2311.08769 [cs.CR]

[15] Kejun Chen, Xiaolong Guo, Qingxu Deng, and Yier Jin. 2021. Dynamic Information Flow Tracking. 12, 8 (2021). https://doi.org/10.3390/mi12080898 arXiv:34442520 Journal Article.

[16] CHEQ AI Technologies Ltd. 2022. *CHEQ.* https://cheq.ai/

[17] Jim Chow, Ben Pfaff, Tal Garfinkel, Kevin Christopher, and Mendel Rosenblum. n.d.. Understanding Data Lifetime via Whole System Simulation. Unpublished. (n.d.).

[18] Dorothy E. Denning. 1976. A lattice model of secure information flow. 19, 5 (1976), 236–243. https://doi.org/10.1145/360051.360056

[19] Antonin Durey, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2021. FP-Redemption. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Leyla Bilge, Lorenzo Cavallaro, Giancarlo Pellegrino, and Nuno Neves (Eds.). Lecture Notes in Computer Science, Vol. 12756. Springer International Publishing, 237–257. https://doi.org/10.1007/978-3-030-80825-9_12

[20] Peter Eckersley. 2010. How Unique Is Your Web Browser? In *Privacy Enhancing Technologies*, Mikhail J. Atallah and Nicholas J. Hopper (Eds.). Lecture Notes in Computer Science, Vol. 6205. Springer Berlin Heidelberg, 1–18. https://doi.org/10.1007/978-3-642-14527-8_1

[21] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. 2014. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. *ACM Transactions on Computer Systems* 32, 2 (June 2014), 1–29. https://doi.org/10.1145/2619091

[22] Steven Englehardt and Arvind Narayanan. 2016. Online Tracking. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA), Edgar Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew Myers, and Shai Halevi (Eds.). ACM, 1388–1401. https://doi.org/10.1145/2976749.2978313

[23] Amin FaizKhademi, Mohammad Zulkernine, and Komminist Weldemariam. 2015. FPGuard. In *Data and Applications Security and Privacy XXIX*, Pierangela Samarati (Ed.). Lecture Notes in Computer Science, Vol. 9149. Springer International Publishing, 293–308. https://doi.org/10.1007/978-3-319-20810-7_21

[24] FingerprintJS Inc. 2022. *FingerprintJS Inc.* https://fingerprint.com/

[25] Global Research & Analysis Team. 2019. *Digital Doppelgängers: Cybercriminals cash out money using stolen digital identities.* Technical Report. Kaspersky Lab.

[26] Alejandro Gómez-Boix, Pierre Laperdrix, and Benoit Baudry. 2018. Hiding in the Crowd. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18* (New York, New York, USA), Pierre-Antoine Champin, Fabien Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis (Eds.). ACM Press, 309–318. https://doi.org/10.1145/3178876.3186097

[27] Imperva, Inc. 2022. *Imperva Advanced Bot Protection.* https://www.imperva.com/products/advanced-bot-protection-management/

[28] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. 2021. Fingerprinting the Fingerprinters:. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1143–1161. https://doi.org/10.1109/SP40001.2021.00017

[29] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. 2023. *FP-Inspector Dataset.* https://github.com/uiowa-irl/FP-Inspector

[30] George F. Jenks. 1967. The Data Model Concept in Statistical Mapping. In *International Yearbook of Cartography*, Vol. 7. 186––190. https://api.semanticscholar.org/CorpusID:215850874

[31] Nikhil Jha, Martino Trevisan, Luca Vassio, and Marco Mellia. 2022. The Internet with Privacy Policies: Measuring The Web Upon Consent. 16, 3 (2022), 1–24. https://doi.org/10.1145/3555352

[32] David Klein, Thomas Barber, Souphiane Bensalim, Ben Stock, and Martin Johns. 2022. Hand Sanitizers in the Wild: A Large-scale Study of Custom JavaScript Sanitizer Functions. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 236–250. https://doi.org/10.1109/EuroSP53844.2022.00023

[33] David Klein, Marius Musch, Thomas Barber, Moritz Kopmann, and Martin Johns. 2022. Accept All Exploits.

[34] Pierre Laperdrix, Nataliia Bielova, Benoit Baudry, and Gildas Avoine. 2020. Browser Fingerprinting. 14, 2 (2020), 1–33. https://doi.org/10.1145/3386040

[35] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the Beast: Diverting Modern Web Browsers to Build Unique Browser Fingerprints. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 878–894. https://doi.org/10.1109/SP.2016.57

[36] Hoan Le, Federico Fallace, and Pere Barlet-Ros. 2017. Towards accurate detection of obfuscated web tracking. In *2017 IEEE International Workshop on Measurement and Networking (M&N)*. IEEE, 1–6. https://doi.org/10.1109/IWMN.2017.8078365

[37] Victor Le Pochat, Tom van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczynski, and Wouter Joosen. 2019. Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation. In *Proceedings 2019 Network and Distributed System Security Symposium* (Reston, VA), Alina Oprea and Dongyan Xu (Eds.). Internet Society. https://doi.org/10.14722/ndss.2019.23386

[38] Victor Le Pochat, Tom van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczynski, and Wouter Joosen. 2022. *Tranco List.* https://tranco-list.eu/list/N7QVW/full

[39] Tianyi Li, Xiaofeng Zheng, Kaiwen Shen, and Xinhui Han. 2022. FPFlow: Detect and Prevent Browser Fingerprinting with Dynamic Taint Analysis. In *Cyber Security*, Wei Lu, Yuqing Zhang, Weiping Wen, Hanbing Yan, and Chao Li (Eds.). Communications in Computer and Information Science, Vol. 1506. Springer Nature Singapore, 51–67. https://doi.org/10.1007/978-981-16-9229-1_4

[40] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA, 281–297.

[41] Jonathan R. Mayer. 2009. "Any person… a pamphleteer" Internet Anonymity in the Age of Web 2.0.

[42] Microsoft Inc. 2022. *Playwright.* https://playwright.dev/(visitedon2022-12-02).

[43] NETACEA. 2021. *Buying Bad Bots Wholesale: The Genesis Market.* Technical Report.

[44] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. 2013. Cookieless Monster. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 541–555. https://doi.org/10.1109/SP.2013.43

[45] Midas Nouwens, Rolf Bagge, Janus Bager Kristensen, and Clemens Nylandsted Klokmose. 2022. Consent-O-Matic: Automatically Answering Consent Popups Using Adversarial Interoperability. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts* (New York, NY, USA), Simone Barbosa, Cliff Lampe, Caroline Appert, and David A. Shamma (Eds.). ACM, 1–7. https://doi.org/10.1145/3491101.3519683

[46] Emmanouil Papadogiannakis, Panagiotis Papadopoulos, Nicolas Kourtellis, and Evangelos P. Markatos. 2021. User Tracking in the Post-cookie Era: How Websites Bypass GDPR Consent to Track Users. In *Proceedings of the Web Conference 2021* (New York, NY, USA), Jure Leskovec, Marko Grobelnik, Marc Najork, Jie Tang, and Leila Zia (Eds.). ACM, 2130–2141. https://doi.org/10.1145/3442381.3450056

[47] Philip Raschke and Axel Küpper. 2018. Uncovering Canvas Fingerprinting in Real-Time and Analyzing ist Usage for Web-Tracking. In *GI Edition Proceedings Band 285 Workshops der INFORMATIK 2018*, Christian Czarnecki, Carsten Brockmann, Eldar Sultanow, Agnes Koschmider, Annika Selzer, and Gesellschaft für Informatik e. V. Gesellschaft für Informatik e. V. Bonn (Eds.). Number 285 in GI-Edition. Proceedings. Köllen, 97–108.

[48] Ruby Documentation. YYYY. *Taint and Untaint in Ruby.* https://ruby-doc.com/docs/ProgrammingRuby/html/taint.html

[49] Iskander Sanchez-Rola, Leyla Bilge, Davide Balzarotti, Armin Buescher, and Petros Efstathopoulos. 2018. Rods with Laser Beams: Understanding Browser Fingerprinting on Phishing Pages. *Norton Research Group* (2018).
[50] SAP. 2022. *Project Foxhound GitHub Repository.* https://github.com/SAP/project-foxhound
[51] Alexander Sjösten, Daniel Hedin, and Andrei Sabelfeld. 2021. EssentialFP: Exposing the Essence of Browser Fingerprinting. In *2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW).* IEEE, 32–48. https://doi.org/10.1109/EuroSPW54576.2021.00011
[52] C. Staicu, D. Schoepe, M. Balliu, M. Pradel, and A. Sabelfeld. 2019. An Empirical Study of Information Flows in Real-World JavaScript. *CoRR* abs/1906.11507 (2019). http://arxiv.org/abs/1906.11507
[53] TAPTAP digital. 2023. *TAPTAP digital - intelligence for marketing.* http://www.taptapdigital.com
[54] Pelayo Vallina, Álvaro Feal, Julien Gamba, Narseo Vallina-Rodriguez, and Antonio Fernández Anta. 2019. Tales from the Porn. In *Proceedings of the Internet Measurement Conference* (New York, NY, USA). ACM, 245–258. https://doi.org/10.1145/3355369.3355583
[55] Wietse Venema. 2008. *Taint Support for PHP.* https://wiki.php.net/rfc/taint
[56] Larry Wall. YYYY. *Perl Security and Taint Mode.* http://perldoc.perl.org/perlsec.html
[57] Webshinker. 2023. *Webshinker.* https://www.webshrinker.com/

A APPENDIX

In this section, we provide supplementary assessments related to our study. In Tables 8 ,9 and 10 we present the list of sources and sinks supported by FP-tracer. We also present the count of the active sources and sinks distributed by the entropy levels of the collection in Tables 12 and 11.

A.1 Performance

To evaluate the performance of our modified browser, we follow a similar method to that of Li et al. [39]. We use the browser `window.performance.timing` API to measure the time between the start of navigation (`navigationStart`) until DOM loading is complete (`domComplete`). We performed this measurement using our modified Foxhound browser and an unmodified Firefox browser for each of the top 1000 web domains according to Tranco [38].

The measured overheads are shown in fig. 7. We measure the average overhead by fitting the distribution with the sum of two Gaussians. The mean of this combined fit yields an overhead of 11 ± 1%. This result is comparable to the overhead of 9.2% measured by Li et al. [39], who use a similarly instrumented browser. However, in comparison to that work, FP-traceradditionally provides fine-grained tainting information at character resolution and more details on domains loading scripts and sending fingerprinting attributes.

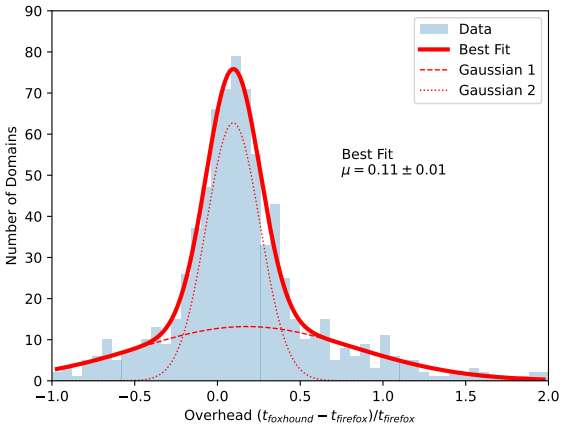

Figure 7: Overhead performance measurement.

```
hash = function(e, g) {
    g || (g = 0);
    if (!e || 0 == e.length)
        return g;
    for (var h = 0; h < e.length; h++) {
        var k = e.charCodeAt(h);
        g = (g << 5) - g + k;
        g &= g
    }
    return g
}

e = hash(screen.width + "x" + screen.height).toString()
```

Listing 2: Example of hashing function used by `bizible.com`.

```
function uuid(s) {
    seed_prng(s);
    var t = h("0123456789abcdef", "")
      , r = []
      , n = void 0;
    r[8] = r[13] = r[18] = r[23] = "-",
    r[14] = "4";
    for (var e = 0; e < 36; e++)
        r[e] || (n = 0 | 16 * rand(),
        r[e] = t[19 === e ? 3 & n | 8 : n]);
    return _(r, "")
}
```

Listing 3: Snapchat UUID using an attribute-seeded PRNG.

A.2 Aggregation Examples

In this section we provide some examples of the aggregation and obfuscation techniques summarized in section 6.6. All code snippets shown in this section have been edited for clarity and brevity.

Screen Resolution Encoding. We detected a number of fingerprinting strings which encode a combination of screen parameters to create a low-entropy aggregated identifier. In the simplest form, we found the screen resolution (`height` × `width`) on 222 domains. More common was the hex-encoded version (`e1000`) which was sent by 284 sink hosts across 759 domains. We also found the sum `height+width+colorDepth` on 836 domains. Another interesting example is shown in listing 2, where the screen width and height are first concatenated before being hashed into a number. We found 230 domains where this combination was collected by and sent to the `bizible.com` domain.

Snapchat Pixel SDK. We detected a total of 941 domains that load the Snapchat Pixel SDK from `sc-static.net`. The script collects a number of attributes, including `userAgent` and language, to form a medium entropy fingerprint. The concatenation of these attributes, together with a list of browser plugins, is used to seed a PRNG. The PRNG, which appears to resemble the Alea algorithm from

```
get: function() {
    var t = [];
    t.push(navigator.userAgent);
    t.push(navigator.language);
    t.push(screen.colorDepth);
    if (this.screen_resolution) {
        var e = this.getScreenResolution();
        void 0 !== e && t.push(e.join("x"))
    }
    t.push((new Date).getTimezoneOffset());
    t.push(this.hasSessionStorage());
    t.push(this.hasLocalStorage());
    t.push(!!window.indexedDB);
    document && document.body ? t.push(typeof
    ↪ document.body.addBehavior) : t.push("undefined");
    t.push(typeof window.openDatabase);
    t.push(navigator.cpuClass);
    t.push(navigator.platform);
    t.push(navigator.doNotTrack);
    return this.murmurhash3_32_gc(t.join("###"), 31)
}
```

Listing 4: Fingerprint created by Hubspot Analytics.

Johannes Baagøe[3], is called repeatedly to create a UUID as shown in listing 3. The resulting UUID is then sent via the `iframe.src` attribute to `snapchat.com`.

Piano Analytics. We also found 322 domains sending a high-entropy fingerprint to `piano.io`. This script creates a SHA1 hash from five attributes (`language`, `screen.height`, `screen.width`, `screen.colorDepth`, `userAgent`) which is sent as part of an XHR request body.

Hubspot Analytics. Listing 4 shows a script snippet loaded from `hs-analytics.net`. The script creates a high-entropy array of fingerprinting attributes, which is then `joined` and hashed using a variant of the MurMurHash3 algorithm [8] which produces a 32-bit integer output. The fingerprint is finally sent to `hubspot.com` via the `src` attribute of an `img` tag. We found this script on 2109 domains.

FingerprintJS. The FingerprintJS library also uses the MurMur algorithm to create a hexadecimal encoded string. We discovered FingerprintJS usage in a script loaded from `webgains.io`, which creates a very high entropy fingerprint and sends it via a `fetch` request body. This activity was found on 527 domains. An example of first-party FingerprintJS usage was found on 427 domains, including `pinterest.com` and `zoom.us`. In these cases, 10 attributes were combined to create a very high entropy fingerprint. Interestingly, the string value of the resulting fingerprint was the same across each of the 427 domains.

Obfuscation Techniques. We also discovered a number of techniques that attempt to obfuscate fingerprinting activities, mainly in use by bot protection services. For example, scripts from *Cheq* [16] send computed fingerprints to domains with seemingly unrelated and random names (e.g., `superpointlesshamsters.com` or `four-timessmelly.com`), making list-based blocking challenging. We also discovered fingerprinting activity which could be traced to *Imperva Advanced Bot Protection* [27]. These scripts were served from the first-party domain using a randomized 48-character path consisting of English words separated by hyphens. In addition, the scripts themselves were heavily obfuscated such that the attribute values were not visible in plaintext.

[3]https://github.com/coverslide/node-alea

A.3 Supporting Tables

In the following we present a number of tables which support our implementation and results from this paper. In particular, we provide lists of supported sources (table 8 and table 9) and sinks (table 10) implemented in this and related work. In addition, we also provide counts of attribute transmissions across all entropy levels, broken down by source (table 11) and sink (table 12). Finally, some examples of sink scripts leaking large numbers of unique attribute combinations are given in table 13.

Table 8: FP-tracer supports 62 taint-tracking sources (part 1).

Source	FP-tracer	[51]	[39]	Remarks
AudioContext				
baseLatency	✓	×	×	
currentTime	✓	×	×	via BaseAudioContext
outputLatency	✓	×	×	
sampleRate	✓	×	×	via BaseAudioContext
AudioNode				
channelCount	✓	×	×	
maxChannelCount	✓	×	×	via AudioDestinationNode
numberOfInputs	✓	×	×	
numberOfOutputs	✓	×	×	
BarProp				
visible	×	×	✓	booleans currently not supported
BatteryManager				
charging	×	×	✓	restricted use in Firefox
chargingTime	×	×	✓	restricted use in Firefox
dischargingTime	×	×	✓	restricted use in Firefox
level	×	×	✓	restricted use in Firefox
Document				
cookie	×	✓	✓	Used as sink, not as source
referrer	×	×	✓	low significance for fingerprinting
Location	×	×	✓	low significance for fingerprinting
History				
length	✓	×	✓	
HTMLElement				
offsetHeight	✓	×	×	
offsetWidth	✓	×	×	
M-AudioBuffer				
getChannelData	×	✓	✓	unspported in foxhound
M-HTMLCanvasElement				
toDataURL	✓	✓	✓	
MediaDevices				
enumerateDevices	×	✓	×	empty array returned in foxhound
Navigator				
appCodeName	✓	×	✓	
appName	✓	×	✓	
appVersion	✓	×	✓	
buildID	✓	×	✓	
cookieEnabled	×	×	✓	bool, unsupported by Firefox
DeviceMemory	×	×	✓	unsupported by firefox
doNotTrack	✓	✓	✓	
hardwareConcurrency	✓	✓	✓	
language	✓	✓	✓	
languages	×	✓	✓	unsupported as taint source
maxTouchPoints	✓	✓	✓	
mimeTypes	×	×	✓	empty array returned
oscpu	✓	✓	✓	
platform	✓	✓	✓	
plugins	×	✓	✓	empty array returned
product	✓	×	✓	
productSub	✓	×	✓	
userAgent	✓	✓	✓	
vendor	✓	×	✓	
vendorSub	✓	×	✓	

Table 9: FP-tracer supports 62 taint-tracking sources (part 2).

Source	FP-tracer	[51]	[39]	Remarks
Screen				
availHeight	✓	✓	✓	
availWidth	✓	✓	✓	
colorDepth	✓	✓	✓	
height	✓	✓	✓	
pixelDepth	✓	×	✓	
width	✓	✓	✓	
VisuaLViewport				
height	✓	×	✓	
offsetLeft	✓	×	✓	
offsetTop	✓	×	✓	
PageLeft	✓	×	✓	
pageTop	✓	×	✓	
scale	✓	×	✓	
width	✓	×	✓	
WebGLRenderingContext				
getParameter	✓	✓	×	
WebGLShaderPrecisionFormat				
precision	✓	✓	✓	
rangeMax	✓	✓	✓	
rangeMin	✓	✓	✓	
Window				
devicePixelRatio	✓	✓	✓	
indexedDB	×	✓	×	Contained Information not known
innerHeight, innerWidth	✓	×	✓	
localStorage	×	✓	×	Contained Information not known
locationbar	×	×	✓	requires BarProp:Visible
Menubar	×	×	✓	requires BarProp:Visible
outerHeight, outerWidth	✓	×	✓	
pageXOffset, pageYOffset	✓	×	✓	
personalbar	×	×	✓	requires BarProp:Visible
screenLeft	✓	×	✓	
screenTop	✓	×	✓	
screenX	✓	×	✓	
screenY	✓	×	✓	
scrollbars	×	×	✓	requires BarProp:Visible
scrollX	✓	×	✓	
scrollY	✓	×	✓	
sessionStroage	×	✓	×	Contained Information not known
statusbar	×	×	✓	requires BarProp:Visible
toolbar	×	×	✓	requires BarProp:Visible
top	×	×	✓	just another Window-object
WorkerNavigator				
platform	✓	×	✓	
userAgent	✓	×	✓	
XMLDocument				
location	×	×	✓	inherited from document
Storage				
quota	✓	×	×	
usage	✓	×	×	
Permissions				
state	✓	×	×	

Table 10: FP-tracer supports 25 taint-tracking sinks..

Sink	FP-tracer	[51]	[39]	Remarks
FetchAPI				
URL	✓	✓	✓	
body	✓	✓	✓	
headers	x	✓	✓	
XHR				
URL	✓	✓	✓	
username	✓	✓	✓	
password	✓	✓	✓	
body	✓	✓	✓	
headers	✓	✓	✓	
src Attributes				
audio	✓	✓	✓	
embed	✓	✓	✓	
iframe	✓	✓	✓	
img	✓	✓	✓	
input	x	✓	✓	
script	✓	✓	✓	
source	✓	✓	✓	
track	✓	✓	✓	
video	✓	✓	✓	
data Attributes				
object	✓	x	x	
Cookie				
Set Cookie	✓	✓	x	Cookie as a source [39]
Navigator.sendBeacon()				
URL	✓	✓	✓	
body	✓	✓	✓	
WebSockets				
URL	✓	x	✓	
Data	✓	x	✓	
HTMLFormElement				
setting attributes	x	✓	x	Not all forms result in requests
Window				
postMessage()	x	✓	x	Comms between windows

Table 11: Count of transmitted attributes for different entropy levels.

Source	Low entropy	Medium Entropy	High Entropy	Very High Entropy
AudioContext.baseLatency	0	1	0	28
AudioContext.outputLatency	0	0	0	28
AudioDestinationNode.maxChannelCount	0	6	5	195
AudioNode.channelCount	0	1	5	187
AudioNode.numberOfInputs	0	1	5	187
AudioNode.numberOfOutputs	0	1	5	187
BaseAudioContext.sampleRate	0	6	5	234
HTMLCanvasElement.toDataURL	0	917	2707	21211
HTMLElement.offsetHeight	3028	547	3769	1839
HTMLElement.offsetWidth	5520	515	3406	1290
History.length	23616	1921	166972	4350
Navigator.appCodeName	1416	1251	15685	5732
Navigator.appName	1977	1255	54521	6729
Navigator.appVersion	1598	270	13885	3557
Navigator.buildID	1252	30	928	6240
Navigator.doNotTrack	2048	397	43078	7377
Navigator.hardwareConcurrency	432	5509	30293	19215
Navigator.language	25861	306458	904318	37234
Navigator.maxTouchPoints	1548	1570	27144	14837
Navigator.oscpu	1345	179	1560	11219
Navigator.platform	5747	12549	106961	24724
Navigator.product	1738	57	3407	7446
Navigator.productSub	1253	72	16143	8558
Navigator.userAgent	188358	55100	407462	46829
PermissionStatus.state	0	9	26	369
Screen.availHeight	42645	2956	189650	20407
Screen.availWidth	40180	2765	190649	21853
Screen.colorDepth	47706	57281	711702	30872
Screen.height	209790	249337	1180128	49163
Screen.pixelDepth	12634	4257	13656	6912
Screen.width	233107	249991	1184912	48304
StorageEstimate.quota	0	0	34	217
StorageEstimate.usage	0	0	34	29

Table 12: Count of vector transmitted per sink for different entropy levels.

Sink	Low	Medium	High	Very High
WebSocket	858	159	4424	21
WebSocket.send	6522	1361	3694	1613
XMLHttpRequest.open(url)	135021	27766	1066413	24169
XMLHttpRequest.send	106008	119767	379113	124922
XMLHttpRequest.setRequestHeader(value)	3409	892	636471	12224
document.cookie	71127	6888	166299	40061
fetch.body	20914	23979	21335	36430
fetch.url	6330	1178	9524	4490
iframe.src	82606	13641	264536	7261
img.src	263857	204190	1497281	76409
img.srcset	0	0	6	0
navigator.sendBeacon(body)	33446	17073	105593	27336
navigator.sendBeacon(url)	13629	496644	876524	5590
script.src	109072	41671	241840	47017
source.srcset	0	0	2	16

Table 13: Sink scripts leaking unique attribute combinations.

Script URL	Domains	Attr comb
https://www.google-analytics.com/analytics.js	38620	28
https://www.clarity.ms/eus-c-sc/s/0.7.6/clarity.js	1276	26
https://www.clarity.ms/s/0.7.6/clarity.js	1123	24
https://cdn.bizible.com/scripts/bizible.js	259	26
https://eum.instana.io/eum.min.js	230	25
https://az416426.vo.msecnd.net/scripts/a/ai.0.js	228	35
https://script.hotjar.com/modules.69d367ac7af64e17f043.js	176	21
https://www.datadoghq-browser-agent.com/datadog-rum-v4.js	141	33
https://script.hotjar.com/modules.f0ba8b655d2d90cf7a94.js	113	26
https://script.hotjar.com/modules.76304821fe35d593f0f4.js	84	27
https://cdn.noibu.com/collect.js	79	25
https://www.datadoghq-browser-agent.com/datadog-rum.js	33	22
https://cdn.appdynamics.com/adrum/adrum-latest.js	26	27
https://cdn.trackjs.com/agent/v3/latest/t.js	26	21
https://cdnjs.cloudflare.com/ajax/libs/rollbar.js/2.4.6/rollbar.min.js	17	22