Double-Edged Shield: On the Fingerprintability of Customized Ad Blockers

Saiid El Hajj Chehade[§], Ben Stock[‡], and Carmela Troncoso[§†]

§ *EPFL,* † *Max-Planck Institute for Security and Privacy (MPI-SP)* ‡ *CISPA Helmholtz Center for Information Security*

Abstract

Web tracking is expanding to cookie-less techniques, like browser fingerprinting [41, 72], to evade popular privacy-enhancing web extensions, namely ad blockers. To mitigate tracking, privacy-aware users are motivated to optimize their privacy setups by adopting proposed anti-fingerprinting configurations and customizing ad blocker settings to maximize the number of blocked trackers.

However, users' choices can counter-intuitively undermine their privacy. In this work, we quantify the risk incurred by modifying ad-blocker filter-list selections. We evaluate the fingerprintability of ad-blocker customization and its implications on privacy. We present three scriptless attacks that evade *SoTA* fingerprinting detectors and mitigations. Our attacks identify 84% of filter lists, capture stable fingerprints with 0.72 normalized entropy, and reduce the relative anonymity set of users to a median of 48 users (0.2% of the population) using only 45 rules out of 577K. Finally, we provide recommendations and precautionary measures to all parties involved.

1 Introduction

To achieve web-browsing privacy, users can combine defenses from a wide set of privacy-enhancing technologies (Web-PETs). Web-PETs can ship as browser extensions [8, 12, 31, 67] or privacy-focused browsers [19]. Ad blockers, the most popular form of these Web-PETs, focus on filtering and blocking activity from advertisement and tracking services (`ATS`) using filter lists curated by experts.

Traditionally, `ATS` use web cookies to link multiple user visits together, which current Web-PETs are blocking effectively [48]. When cookies are not available, `ATS` leverage browsing fingerprints, i.e. unique features associated with the user's browser setup, to re-identify users [41, 72]. One well-studied fingerprinting vector is extension fingerprinting [72], in which the features used represent *whether the user browser has certain extensions installed or not*. `ATS` extract this information by exploiting browser APIs and extension interactions with web pages. Prior work shows that *extension enumeration attacks* can identify a significant portion of the global user population (Section 2).

To defend against browser fingerprinting [2, 23, 33, 62], the community proposes many web PETs [24, 39, 51], and ad blockers make available filter lists tailored to defending against malicious scripts like fingerprinting. *Privacy-aware users* enable many of these PETs (*e.g.*, use the Tor browser and/or disable JavaScript) and fine-tune their ad blocker filter lists to block more (fingerprinting-oriented) trackers. These additional defenses would block fingerprinting scripts and protect these users from most existing browser fingerprinting (like *extension enumeration*), allowing them to hide in large anonymity sets. Previous work showed that users overestimate the privacy protection they obtain from the layering of defenses [66]. In this work, we study whether customizing filter-list choices, on top of employing existing protections, gives a false sense of safety to *privacy-aware* users.

Unlike the *extension enumeration* literature (Section 2), which examines attacks maximizing the coverage of *detected extensions* in a *global user population*, we examine attacks maximizing the coverage of *detected filter lists* in a *privacy-aware user subpopulation*. While prior work focuses on extracting one bit from many extensions (whether the extension exists or not), our attacks extract multiple bits (whether filter lists exist or not) from only one extension. This means that even if *privacy-aware users* install a limited homogeneous set of extensions, their distinctive configurations of ad-blockers might uniquely identify them through our attacks. We believe that the **discriminate** risk of ad-blocker configuration fingerprinting on *privacy-aware users* is noteworthy, given that ad-blockers single-handedly cover 164 Million installs (23% of installs among the top 20 Chrome extensions [9]) and serve 31.5% of web users [25]. In addition, the subpopulation of ad blocker users who configure their ad blockers, which we consider to be *privacy aware*, is significant; it is at least 18% for AdGuard – a prominent ad-blocker (Section 6) compared to only 4% of global users who disable their JavaScript [46] and target to the established field of stylistic extension fingerprinting [44].

Concerns about filter-list fingerprinting have been a topic in multiple non-academic discussions [1, 20], and some implemented ad-hoc attack proof-of-concepts [21]; but this risk, nevertheless, remains unquantified.

Our work quantifies the privacy risk stemming from customizing filter-list configurations in two popular ad blockers. Specifically, we evaluate the trade-off between blocking more ATS– by enabling more filter lists in ad-blocker settings – and the risk of fingerprinting. Lin et al. [44] first investigated this tradeoff and argued that privacy extensions' benefits outweigh their fingerprinting risks; our results directly contest their unchallenged conclusions by uncovering novel multi-bit fingerprinting attacks (Section 6). To our knowledge, we are the first to present **practical attacks** to recover ad-blocker configurations, allowing us to quantify how much specialized configurations undermine users' privacy.

Our primary contributions are as follows:

- We establish a baseline attack – that assumes maximum coverage of filter lists – and discover three scriptless attacks to fingerprint ad-blocker configurations.
- We introduce the concept of *equivalence sets* to minimize the number of rules needed to maximize fingerprintability.
- Unlike many extension enumeration studies that stop at extension coverage [28], we evaluate the effectiveness of fingerprinting on two real-world *privacy-aware* user datasets from prominent ad blockers, AdGuard [12] and uBlock Origin [67]. Our attacks can identify 69/87 and 63/68 respectively of these ad-blocker filter lists. With these lists, 50% of *privacy-aware* AdGuard users have an anonymity set smaller than 48 users (0.2%).
- We discuss and evaluate possible detection and mitigation techniques and conclude with precautionary recommendations to users, ad blockers, and browsers.

2 Related Work

In this work, we study whether fingerprinting ad-blocker settings, specifically filter lists, can reduce the privacy of ad-blocker users. Our work goes beyond web-extension fingerprinting studies, as they only determine whether ad blockers and extensions exist. Also, our motivation to fingerprint ad-blocker functionality comes from the anti-adblock movement: websites that detect and restrict ad-blocker users from accessing their pages. Finally, our attacks rely less on JavaScript, which aligns with stylistic (or scriptless) fingerprinting studies.

Web-Extension Fingerprinting. Extension fingerprinting is a well-studied field studying how attacker-controlled sites can uncover the user's installed browser extensions [72]. Many studies do not perform any user studies, reporting only on the number of fingerprintable extensions [28, 56, 60, 63]. Studies that perform a user study report different degrees of fingerprint uniqueness and quality – between 14.10% of 854 users [62] and 39.29% of 7,643 users [33] – because of user datasets sizes, distribution, and collection methods (*e.g.*, in-lab testing, crowdsourcing, and public forum data). Earlier studies detect extensions based on web resources they expose to the page (*e.g.*, icons, images, *etc.*) called *web accessible resources (WAR)* [33, 55]. Only 28% of Chrome extensions and 6.73% of extensions are detectable with WARs [33], so newer studies check if an extension exists by their modification of the webpage DOM(*e.g.*, inserting buttons, removing ad containers, *etc.*) or load-time impact [15, 40, 52, 62, 63] or by exploiting inter-extension message passing [15, 28]. Other studies extend this by modifying or interacting with the page to trigger extension-specific actions [60]. Our attacks also look at the modifications made by the ad-blocker by removing ATS-related elements or blocking requests. However, unlike this line of work, we aim to extract more than one bit from the extension and not simply know whether it exists. Additionally, since most of these attacks require a fingerprinting script, they can be thwarted by disabling javascript or script-blocking employed by *privacy-aware* users. We show that in such conditions, our attacks still deanonymize these users despite their prior sense of protection.

In concurrent work, Solomos et al. [61] proposed Hecate, a multi-bit fingerprinting against extension configurations. Specifically, the authors fuzz configuration options, primarily related to extension storage, to elicit fingerprintable behavior and create a database of the fingerprints of different extension configurations. To identify a configuration in a given browser, they compare the measured fingerprint to the configuration database. Using an MTurk pilot study they find that 25% of the 375 participants could be uniquely identified.

Our work differs from Hecate in three main areas: (1) we specifically focus on *privacy-aware* users, (2) we strictly rely on scriptless attacks, and (3) we do not need a database of fingerprints to recover a particular configuration; our attacks directly extract the configuration from the measurements.

Scriptless Fingerprinting Since ad-blockers or other proposed defenses [39] can block the fingerprinting scripts, newer studies suggest fingerprinting attacks that deliver the fingerprint to the attacker using other page features (mainly CSS) without using JavaScript [44] The literature distinguishes between JS-based fingerprinting and CSS (or stylistic) fingerprinting. We modify this distinction by grouping attacks that do not use dedicated scripts into "scriptless attacks" (even if some need JavaScript enabled) and others as "scripted" attacks. This distinction comes from the vast difference in detecting and mitigating scripted attacks versus scriptless attacks by *SoTA* defenses [39]. In general terms, these attacks exploit differences in devices (*e.g.*, display size, available fonts, *etc.*) and implement CSS styles that execute for one device but not the other. For example, the rule `@media (min-width:300px) #probe {background: url(a.com/img.png)}` sends a re-

Table 1: Most Popular ATS-blocking tools. Total users are computed from the Chrome web store that serves popular Chromium browsers like Chrome, Edge *etc*., and the Firefox add-ons store.

Ad Blocker	Integration	Decision Method	Blocking Issue Reporting	Total Users	N. Filter Lists
Adblock Plus	Extension	Filter Lists	Proprietary Forum [30]+	Over 47 Million	31
uBlock Origin	Extension	Filter Lists	GitHub Forum [68]	44 Million	68
Brave Shields	Browser-Native	Filter Lists + Heuristics	Proprietary Forum [18]	25.3 Million	51
AdGuard	Extension	Filter Lists	GitHub Forum [13]	15 Million	87
Ghostery	Extension	Filter Lists + Tracker List	GitHub Forum [6]	3 Million	12*
Privacy Badger	Extension	Heuristics	GitHub Forum	2 Million	-

* Ghostery allows to customize each list further at the ATS domain level.
+ User reporting mechanism is private, *i.e.*, ad-blocker metadata is not shared with the public.

quest to a.com only if the window size is at least 300px [44]. As you can see, the fingerprinting signal is delivered directly through CSS and not collected by a script as with prior techniques. Our scriptless attacks expand on this concept to probe and send signals about active ad-blocker settings.

Anti-Adblock. Today, sites that depend on ATS to operate borrow techniques from extension fingerprinting to detect and block access from visitors with ad-blockers [37]. 30.5% of the Alexa top-10K websites use anti-adblock scripts [73] that check whether certain DOM elements display correctly or get blocked. These elements can be real Ads or "baits" added by the script [73]. We got inspired to use multiple baits that trigger only for specific configurations of the ad-blocker to fingerprint the ad-blocker user's settings. We found only one mention of the ad-blocker configuration fingerprinting in a blog post by FingerprintJS employee [1], which showcases inserting DOM elements that get hidden only for specific ad-blocker rules and verifying if they get hidden through a script similar to anti-adblock. This work presents an in-depth analysis of ad-blocker filter-list attacks that do not require dedicated scripts and cover more ad blockers and configurations.

3 Background

First, we present preliminary information relevant to our proposed ad-blocker fingerprinting attacks. We introduce ad blockers, their inner workings, and how users customize them.

3.1 Ad Blockers

Ad blockers aim to protect web users from ATS by *blocking* ATS-related network requests [10, 14] and altering web pages to hide ad-related content (referred to as *cosmetic* [14] or *content* [10] changes). Ad blockers are popular among web users (Table 1) for their dual benefits: protecting user privacy by limiting ATS activity and improving user experience by decluttering the page. Most ad blocking is provided by browser extensions, *e.g.*, Adblock Plus [31], uBlock Origin [67], Ghostery [5], AdGuard [12], Privacy Badger [8], while some come directly bundled in browsers, *e.g.*, Edge [47], Firefox [45], and Brave browser [19]. Widely-adopted ad-blockers detect ATS activity predominantly through **block-listing**, while a few rely on **heuristics** (*e.g.*, Privacy Badger [8]) – see Table 1. **Block-listing** works by checking each request and element against a preset list of known ATS-related requests and patterns. Expert maintainers curate these lists in public repositories and update them regularly [13, 30, 68]. In this work, we focus on block-listing ad blockers as they are the most popular and have parsable configurations.

3.2 Filtering Rules

If block lists simply enumerate all possible ATS resources, the list would be too large to be practical. Instead, maintainers extract common patterns from ATS and write them into *filtering rules*, *e.g.*, blocking all requests that contain */advert. The ad blocker compares every request or web page element against all active filtering rules. AdBlock Plus [31] first introduced the syntax governing these rules, which has since been adopted and extended by newer ad blockers [14]. In addition to the *pattern*, rules include *modifiers* prefixed by a $ sign. *Modifiers* add granularity in defining precisely what to block and when, *e.g.*, the $img modifier blocks only image requests. As a demonstration, rule 1 in Listing 1 blocks all image requests that start with the domain.com/contact URL prefix.

```
1 ||domain.com/contact^$img
2 ##.ad
3 ##.ad$remove()
```

Listing 1: AdGuard Filtering Rules Examples

Rules can be categorized in many ways based on their syntax, *e.g.*, whether it is an exception or not. To our purposes, we focus on two filter-rule attributes: its *scope* and *type*.

Rule Scope. Rules can either be *generic*, *i.e.*, active regardless of the website the user is visiting, or *specific*, *i.e.*, active only on specific websites. For example, for requests containing /market/ as ATS on domain.com but not on wikipedia.com, maintainers can restrict the rule to domain.com to avoid false

positives. *Specific* rules can be restricted to a list of domains [1] by appending the `$domain` modifier.

Rule Type. Rules that block network requests are called *blocking* rules; and rules that alter the web-page HTML or CSS (*e.g.*, hiding ad containers) or inject JavaScript are called *cosmetic* rules [14] – prefixed by the `##` symbols (rules 2 and 3 in Listing 1). For instance, rule 2 hides all DOM elements tagged with the class `ad` by altering CSS styles *on any page*. Rule 4 goes beyond rule 1 and removes the element from the DOM completely *i.e.*, from the rendered web page.

3.3 Filter Lists

As the ad-blocking community grew, so did the number and diversity of filtering rules. Today, ad blockers contain hundreds of thousands of rules for specific purposes like language-specific rules (*e.g.*, rules for German websites), cookies-blocking rules, social-media-specific rules, *etc*. [12]. Maintainers group filtering rules of the same purpose into **filter lists** hosted on GitHub or by 3rd-parties on standalone websites [7]. Maintainers fine-tune rule granularity (scope, type, modifiers, *etc*.)and organize them into dedicated lists with five goals in mind: (1) easing maintenance, (2) allowing users to activate lists based on their preferences [11], (3) maximizing the coverage of blocked `ATS`, (4) optimizing list sizes to reduce web-page loading time, (5) preventing web-page breakage due to false positives [26, 58]. For example, the *AdGuard Social Media filter list* specializes in removing social media buttons such as "Like" and "Share" buttons available in both AdGuard and uBlock Origin [11]. Maintainers frequently update their lists following user reports of new `ATS` resources, existing `ATS` evasively changing their patterns [22], or web pages breaking [26, 58] into dedicated forums or proprietary reporting mechanisms (Table 1). Note that two distinct filter lists can still contain duplicate rules due to independent maintainers (*e.g.*, *AdGuard Base* [11] and *EasyList* [4]) or similarity in purpose between the lists (*e.g.*, *Austrian* and *German* filter lists).

3.4 User Customization of the Ad Blocker

In addition to proprietary filter lists, ad blockers provide users with a set of third-party filter lists to activate or deactivate, *e.g.*, the famous *EasyList* [4] developed by Adblock Plus [31] and adopted by most major ad blocking extensions. Updating the filter-list subscriptions is straightforward: users can navigate to the ad blocker's settings from the extension view and click on the list toggles to activate/deactivate them. We identify two reasons users would personalize their filter-list subscriptions: (1) subjective browsing patterns and expectations (*e.g.*, some users might want to block adult content, some might dislike social media buttons, and others might find "debatably" annoying web-page elements with "AdGuard Annoyances"), and (2) different languages (*e.g.*, "AdGuard Arabic"). When the users activate multiple lists, the ad-blocker extension will take the **union** of the activated lists, *i.e.*, it will check the behavior against all rules in the concatenation of the activated lists.

[1] Domains are primarily expressed as "regular domains", *e.g.*, `domain.com` or "any TLD domain", *e.g.*, `domain.*`, with the star as a wildcard to match any suffix like `domain.co.uk` [14].

4 Fingerprinting Attacks

Since ad-blockers can be extensively personalized (Section 3), adversaries can use the user's filter-list subscriptions as a fingerprint for cross-site tracking. In this section, we describe the threat model, propose client-side attacks to identify active filter rules, and present the methods to optimize fingerprinting.

4.1 Threat Model

The considered adversary is interested in (fully or partially) uncovering the user's active filter lists and mapping them to a constant user fingerprint across websites – without relying on stateful information like cookies. Following prior fingerprinting attacks [2, 23, 33, 44], we assume the adversary cannot control the browser and cannot directly query the filter lists from the browser storage. Instead, the adversary can ask interested websites to embed JavaScript, CSS, or HTML components to conduct tests for active filter rules only by observing the impact of these rules on the page: *e.g.*, specific requests that get blocked or page elements that disappear. We refer to this act as *filter-rule detection attacks*. To arrive at optimized user-level fingerprints, the adversary must also *decide which filter rules to test for* and design a *mapping from in-page test results to user fingerprints*. Tracking Services that integrate directly into the first-party domain already exist [39], *e.g.*, *Fingerprint.com* [29] and *Simple Analytics* [54]. Also, adversaries embedding HTML elements in the page source is a reasonable assumption for the scriptless literature [35, 44] similar to our work (Section 2). However, we also present a baseline attack that does not require first-party modifications, which is more detectable by *SoTA* fingerprinting detectors.

4.2 Filter-Rule Detection Attacks

As shown in Figure 1 (1), the attacker adds page components that behave differently if a target rule is active. For example, a simple test for a rule hiding elements with the class `advert` inserts an element with this class and checks if it's visible or not using JavaScript. Each attack is composed of a **collection step** (the reaction of embedded components to the rule) and a **sending step** for the results to reach the adversary's domain.

In this work, we present three attacks that offer varied guarantees/assumptions compromises, as they exploit distinct collection and sending mechanisms. While we only present the most powerful attack in-depth, we emphasize with the other attacks the inherent ability of CSS features to leak the behavior of

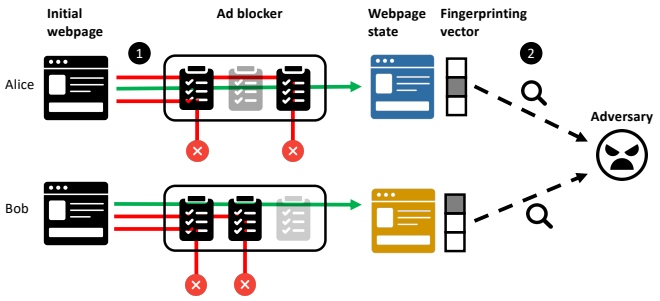

Figure 1: Overview of the fingerprinting attack framework. The adversary implements two steps: (1) triggering web activity that specific filter lists in the *fingerprinting template* can block, and (2) sending the signals back.

ad blockers. You can refer to Table 3 for a summary of testable rules for each attack. We also provide a proof-of-concept demo of the attacks at flfp-demo.github.io.

Baseline attack. This is the most aggressive attack, in which we assume the attacker has control over select third-party domains loaded as iframes in the target website and a dedicated fingerprinting script. This attack represents a (noisy) baseline for the maximum number of detectable ad-blocker filter rules without access to the user's browser. *Generic* rules are straightforward to detect. For *generic blocking* rules, the attacker script can use the `fetch` API to check whether the request is blocked or not [2], and for *generic cosmetic* rules, the attacker can monitor whether HTML elements that match blocking rules are visible or not (blocked) [1]. However, domain-*specific* rules trigger for specific domains, potentially different from the attack website. So, the attacker needs to load the domain within an `iframe` in the parent page and send the requests from that context. Still, the browsers' *same-origin* policy prevents scripts in the parent domain from accessing elements in a 3rd-party `iframe`'s domain and only allows for communication through the `window.postMessage` API [3]. The attacker can use this API to send the attack results from a script in the `iframe` of the colluding domain to the script of the parent domain. As we later show in Section 7.3, the `window.postMessage` API is already used by some websites, and the adversary could set up such a system with a few `ATS` domains to expand their tracking network. In Section 6.4, we show that generic rules sufficiently cover most lists and that the adversary needs less than 12 domains to cover all the filter lists.

CSS animation attack. CSS animations help integrate dynamism in the behavior of web elements by controlling the value of CSS properties over time, like loading bars, changing element background colors, *etc*. [3]. One property that CSS can animate is the `background-image`. Moreover, modern browsers implement many optimizations to reduce the performance costs of loading web pages: we experimentally find that the browser will not fetch the background image request if the element is hidden (with the `display:none` CSS style). Both ad blockers hide blocked images and elements with `display:none`, so the attacker can rely on this background image request to determine whether the element is blocked. However, uBlock Origin applies this style with a slight delay, during which the background image request goes through. To resolve it, we trigger the background image style after a 1s delay using CSS animations to wait for the `display:none` to be applied. Overall, the attack can detect `generic blocking` and `cosmetic` rules. The advantages are that CSS animations work even with JS disabled and that requests get triggered only for unblocked rules, reducing the attack footprint.

Summary of other attacks. In addition to the CSS animation attacks, we also discovered two other ways to infer the presence of a filter rule. Given their identical coverage of rules and slight limitations, we only briefly sketch them here and refer the interested reader to the appendix for a more verbose discussion.

These attacks do not depend on animations, which showcases the abundance of methods to detect active filter rules without scripts. The first attack relies on "lazy loading" images, where browsers would not request an image until it is close enough to the user's viewport to reduce the web page's loading overhead. An attacker could add an element to be blocked by a specific rule on top of an image element that sends a request to the attacker's server. If the element is blocked, the attacker receives the image request earlier than others in the same row. However, this attack can be subject to timing imprecision, depending on the user's scroll speed. The second attack exploits a new feature proposed by the W3C [70]: `@container` queries. Styles in this `@container` context would only execute if a target "container" element is rendered with "computed" styles or values *e.g.*, height, opacity, *etc*.. "Computed" styles and attributes for a DOM element are values determined by the browser after parsing all CSS stylesheets targeting this element, JavaScript, and web-page environmental changes (*e.g.*, screen width), such as the ad-blocker styles that hide the element. The attacker can use these queries to infer whether an ad-blocker rule acted on this element or not. All browsers fully support querying attributes like container width. However, more advanced "style" queries, like opacity, are still a work in progress. However, `@container` queries only have 6% adoption (Table A2), making it easier to patch by browsers.

4.3 Optimizing Filter-Rules to Test

By running multiple tests on different rules, the attacker can uncover the *fingerprinting vector* pertaining to the user's active filter rules (*e.g.*, [rule 1: Yes, rule 2: No, $\cdots$]). Given that the user enables and disables filter lists (**??**) not specific rules, the presence or absence of tested rules can reveal whether complete filter lists are active or not, *e.g.*, if `amazon.de` is blocked, and only *German* filter list contains a rule that blocks it, then knowing that `amazon.de` is blocked leaks the presence of the

[2] Adding `{mode: "no-cors"}` allows us to distinguish between blocking due to CORS (returns a failed response) and ad blockers (throws a `TypeError`)

complete list. So, rather than maximizing the detectable rules' count, the adversary has to test for the fewest rules that maximize the number of detectable filter lists for users, referred to as **filter-list coverage**. Maximizing **filter-list coverage** extracts most info about users and improves fingerprint quality, while minimizing rules reduces the number of tests, limits detectability, and increases the attack efficiency. Constraining the number of tests aligns with the goals of prior work [33, 44].

We study two ways to map rules to fingerprints and evaluate their **filter-list** coverage in Section 6.1.

(A) Unique Rules per Filter List. A straightforward method is to choose one unique rule for each filter list and test for them.

(B) Rule-Filter-List Equivalence Sets. While (A) restricts us to an m-to-one mapping between one list and all m rules unique to it, consider here a more general case. Any filtering rule r belongs to a set of one or more lists l, refer to it as the *prefix set* of r $F_r = \{l_1, l_2, \cdots\}$. The rule r is active if and only if one or more lists in F_r are active. We draw the equivalence $r \leftrightarrow F_r$. We can find multiple rules r_i that belong to the same set of lists, and establish an identical equivalence, *i.e.*, all r_i are active if and only if one or more lists li are active. We refer to the sets of rules and sets of lists as the *equivalence set*. Note that rules unique to filter-lists used in (A) are a subset of equivalence sets of the form $\{r_1, r_2, \cdots\} \leftrightarrow \{l_i\}$; we refer to all other equivalence sets (that map to two or more lists) as "complex" equivalence sets.

4.4 Fingerprinting Metrics

We rely on multiple standard metrics from the literature [24, 32, 33] to evaluate the success of the proposed fingerprinting attacks on user populations. Previous work measures the fingerprinting vectors' quality and utility with two metrics: the *normalized Shannon entropy* of the fingerprints [33, 44], and the *proportion of k-small anonymity sets*.

Normalized Shannon entropy. The normalized Shannon entropy measures the ratio between the number of bits an average fingerprint introduces about a user compared to the number of bits needed for the fingerprint to be unique [65]. It is an appropriate metric to compare fingerprinting datasets with different supports as in Table 2. We define it in Equation 1.

$$H(D) = -\frac{1}{\log_2 N} \sum_{a_i \in A} P_D(a_i) \log_2 P_D(a_i) \qquad (1)$$

D is the set of user fingerprints, $N = |D|$, $P_D(\cdot)$ is the frequency of a given fingerprint, and A is the set of all possible fingerprints (all possible values of the fingerprint template).

Proportion of k-small anonymity sets. We call the set of users with the same fingerprint as the *anonymity set*. Users in the same anonymity sets are indistinguishable, so the attacker's goal is to minimize the sizes of these anonymity sets. The k-small anonymity sets are all anonymity sets of size less or equal to K. Let $U_k(D)$ be the fraction of these sets out of all anonymity sets in D. For example, $U_1(D)$ is the *proportion of unique users*. In our study, these attacks are meant to be used along with other established fingerprinting techniques, so reducing the anonymity-set size to $k \leq 100$, for example, can still uniquely identify users when considering additional signals. While traditional fingerprinting attacks group all users implementing appropriate fingerprinting defenses into large anonymity sets, the attacks we propose will further segment them into smaller anonymity sets since the same users will heavily customize their ad blockers.

4.5 Mapping to User Fingerprints

To optimize for the metrics in Section 4.4, not all filter lists are equally valuable for fingerprinting users, *e.g.*, the *AdGuard* base list should be active by default for all users. So, the attacker should choose which lists to test for that yield the best fingerprinting utility. We refer to this subset of lists as the *fingerprinting template*. In our work, we consider the two popular fingerprinting goals in browser fingerprinting literature: targeted fingerprinting and general fingerprinting [32, 33, 34].

Targeted Fingerprinting. Here, the attacker is interested in distinguishing whether the website visitor is a targeted user or not, *i.e.*, they aim to maximize the likelihood of distinguishing a specific individual from the remaining population [33]. So, the adversary searches for the smallest possible fingerprinting template, such that the target's fingerprint is unique or has a small anonymity set. Note that the fingerprint template for each target user can be different.

General Fingerprinting. Here, the attacker aims to find a global fingerprint template that uniquely identifies the largest number of users. Gulyas et al. [33] propose a near-optimal algorithm by choosing the least number of features that divide the population into anonymity sets of the smallest sizes.

Gulyas et al. [32] shows that the optimization problems for both settings is an NP-hard problem, and they suggest near-optimal greedy algorithms – with an approximation ratio of $(1-1/e) \approx 0.6$ (*i.e.*, the algorithm will cover at least 60% of the optimal elements of the template). We adapt their algorithms for our settings. For more details, we refer the reader to their paper [32] and our code-base (Section 9).

5 Ad Blockers and User Datasets

In this section, we present the data collection methods and the data sources used for user ad-blocker configurations.

5.1 Collecting Ad-Blocker Configurations

To study the uniqueness of filter list combinations activated by users, we require a dataset of real-world ad-blocker users' configurations. However, crowdsourcing efforts comes with many challenges, namely: (1) Automatically processing and storing

the web-extension profiles[3] privately and ethically reduces participation rates [33]; (2) crowdsourcing such configurations would not yield the audience of interest for our analysis (the subpopulation of privacy-aware users), as privacy-aware users are unlikely to participate in crowdsourcing due to the possibly invasive surveyed information; (3) we cannot guarantee that participants are everyday ad-blocker users (participants may install the extension just to get the reward).

As finding study subjects is challenging, we opt for another data source of publicly accessible configurations, similar to prior work [40]. We scrape ad-blocker forums, where users post feedback and requests to ad-blocker and filter-list maintainers, to gather user ad-blocker configurations at scale. We only include "valid" posts where we can reliably extract the filter-list subscriptions from the metadata of the user post. We acknowledge the representativeness limitations of our datasets in Section 5.3; however, we also argue in that section that it is a sufficient *realistic* sample for our demographic.

We investigate the forums of the most popular ad-blockers (Table 1) and collect datasets from forums that have more than 5,000 users with "valid" posts. While we only study two ad blockers, many of their filter lists overlap with those used by other ad-blockers and privacy browsers like Brave [19] – which we could not obtain public forum datasets. We give more details about our dataset filtering approach in Appendix C.1.

5.2 Investigated Ad-blocker Forums

We investigate the most popular ad-blocking tools (Table 1) and their respective forums to gather user ad-blocker configurations at scale. We exclude Adblock Plus, Brave Shields, and Ghostery as their forums are not publicly accessible or do not contain structured user posts – more details in Appendix B.

uBlock Origin [68]. uBlock Origin is the second most popular ad-blocker extension with 44 million installs (Table 1). Extension maintainers provide a public forum on GitHub where users can report issues and request new features. Despite including many posts, only recent posts start including structured information about the user's filter lists that can be automatically scraped. So, we are limited to 5,890 posts (52% of total posts). We refer to the user settings dataset as `UBLOCK`.

AdGuard [13]. AdGuard is the third most popular ad-blocker extension with 15 million installs (Table 1). Similar to uBlock Origin, they host a public issues forum on GitHub. The advantage of this forum is the consistent structure of posts since 2022, which allows us to automate the user ad-blocker configuration extraction: each post features a table containing the names of the user's active filter lists. A limitation, however, is that all posts are submitted by the AdGuard bot account, preventing us from connecting multiple issues to the same user. As an approximation, we assume that each post has a unique user. Our approximation should not impact the dataset utility or uniqueness results, as we show in Section 6.3 that 90.4% of uBlock Origin users rarely post multiple issues with different configurations. Hence, this dataset provides a lower bound on the uniqueness of users' configurations, as we over-estimate the number of people with identical configurations. We include the 18,494 AdGuard user settings we extracted. We refer to the user filter-list subscription dataset as `ADGUARD`.

Thus, we argue that our approximation is not an issue for our study. At most, it means that we overestimate the number of people with identical configurations, and thus, our results constitute a lower bound of the users' uniqueness.

5.3 Representativeness of Forum Users

Given the challenges of collecting live samples of ad-blocker users (Section 5.1), we use ad-blocker forum authors as a proxy for *privacy-aware* ad-blocker users. While some *privacy-aware* individuals might not interact with the forum (low recall), we argue that most forum users are privacy-aware (high precision). More so, we believe forum users represent the best *practical* sample of proactive ad-blocker (and *privacy-aware*) users for

(1) Low bar of entry to forum. At first glance, forum authors can seem like a biased tech-savvy sample, as laypersons are unlikely to venture into GitHub to post issues. However, both AdGuard and uBlock Origin offer in-app feedback reporting interfaces – easily accessible to most users. For example, AdGuard relies on a bot to deliver the posts from the app interface to the GitHub forum. We observe that the vast majority of the posted issues reach the GitHub forums through these interfaces, and raise non-technical concerns (*e.g.*, requests to unblock specific pages, block specific Ads, *etc.*).

(2) AdGuard forum authors mention using advanced privacy features. Among the metadata shared in AdGuard forum posts, we can find whether authors specifically enable advanced privacy features. We find that at least 40% of posts mention using "Advanced protection" or "Stealth" features conforming with the behavior of advanced (*privacy-aware*) users.

(3) AdGuard internal statistics corroboration. We contacted Ad-blocker teams for filter-list configuration datasets. AdGuard's team replied that they do not collect per-user configurations and also pointed out that using the forum is the way to get practical and meaningful per-user data for our purposes. Fortunately, AdGuard's team shared internal stastistics of filter-list popularities which support the representativess of the forum dataset. Filter-list popularity in their internal statistics is close to the forum's list popularity (0.7 Rank Biased Overlap and 0.009 EMD). The only forum lists over-represented are those related to aggressive blocking (reinforcing that these are privacy-aware users) (Figure A1).

Finally, the size and representativeness of fingerprinting datasets is a common issue across all fingerprinting studies [23, 33, 44, 62]. In the future, we hope ad blockers can utilize

[3] Web extension data containing user settings stored in the browser

Table 2: Normalized Shannon Entropy of our collected datasets compared to previous work.

Dataset	Size	Entropy	Unique Users
FPJS[1][2]	1,848	0.53[2]	29.1%
StylisticFP [44]	1,848	0.58[2]	29.4%
Audio Fingerprint [23]	2,093	0.25	02.3%
Extension Fingerprinting			
XHOUND [62]	856	0.44	14.1%
Gulyas et al. [33]	7,643	0.64	39.3%
CARNUS [3][40]	9,286	-	94.5%
Hecat [61]	375	-	25.6%
Our Work			
ADGUARD	18,494	**0.75**	28.5%
UBLOCK	5,890	**0.66**	28.4%

[1] The metrics for FPJS were most recently evaluated in the StylisticFP study [44].
[2] The paper reports entropy per feature. We select the maximum entropy.
[3] The authors report multiple uniqueness values by considering different number of extensions. They focus on uniqueness for users installing 4 extensions, reported in the table.

our methodology to evaluate the impact of our attacks on their internal user datasets.

6 Evaluation

In this section, we evaluate the viability of the fingerprinting attacks proposed in Section 4.2 in re-identifying filter lists and users. We also evaluate the fingerprint stability over time and explore the detectability of the attacks. We make the code and data for this work available (instructions in Section 9).

6.1 Rule and List Coverage

Cosmetic rule tests do not require sending ATS requests, so they can be more attractive to adversaries in reducing network footprint. So, We evaluate how many filter-list and equivalence sets (Section 4.3) each attack and subset of rules can re-identify. Higher coverage is linked to higher-quality user fingerprints.

Results. We summarize our results in Table 3. First, we notice that a substantial decrease in rule coverage does not lead to a strong reduction in list coverage – across all attacks. For instance, in AdGuard, generic cosmetic rules account for less than 16% of all available rules, yet they can identify 58 (75%) lists. The reason is intuitive: to identify a list, the adversary needs no more than one unique rule for this list. Second, we observe that while the number of detectable lists is more robust to variance, the number of available equivalence sets varies more strongly with respect to the rule coverage. Generic cosmetic rules only expose 143 equivalence sets, 2.8 times less than that of Baseline attack at 406. On top, we find that "complex"

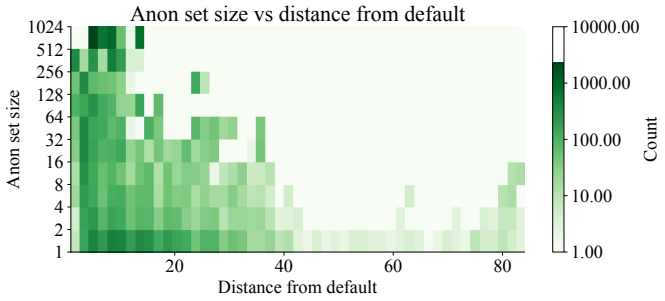

Figure 2: The distribution of users across their anonymity set size and the distance of their filter list configurations from the default. Dataset=ADGUARD, Browser Attack=CSS animation attack, and Attack Goal=Targeted Fingerprinting. For all other combinations, we observe similar trends.

equivalence sets (containing more than one list) constitute 53% of the fingerprinting template on average across attacks for general fingerprinting on ADGUARD. We conclude that, while the attacks target restricted subsets of rule types, they still can reliably leak a large portion of information (lists and equivalence sets) from the user's ad-blocker configuration.

6.2 Reducing User Anonymity

To measure the power of the attacks in re-identifying users, we evaluate the entropy and uniqueness of the attacks (Section 4.4) before applying targeted and general fingerprinting – to measure the power of an unconstrained attack to identify users – and after applying them. Regarding general fingerprinting, we report the smallest fingerprinting template size (between one and 90), at which entropy and uniqueness stop increasing. Regarding targeted fingerprinting, since each user is treated separately, we measure the portion of unique users and the maximum (worst-case) fingerprinting template size. Finally, we study how the anonymity set size varies with the degree of customization of the filter lists (more about computing filter-list distance in Appendix C.5), to answer our primary question: *"to what extent does customizing your filter-list subscriptions increase your fingerprintability?"*

Results. We summarize the results in Table 4. First, the unconstrained entropy of the attacks is comparable to the overall dataset entropy in Table 2 for ADGUARD and 0.1 lower for UBLOCK; this means that the attacks reliably extract the overall user ad-blocker customization info. On top, the uniqueness results are comparable to those achieved by prior attacks in Table 2, with a maximum of 26.66% for ADGUARD and 20.22% for UBLOCK– albeit we target an audience of privacy-aware users and assume the general population is already susceptible to prior attacks [2, 23, 62]. Comparing attacks, we find that all proposed scriptless attacks, *e.g.*, the CSS animation attack, achieve the highest entropy (0.73 for ADGUARD and 0.56 for UBLOCK) and approach the uniqueness of our baseline Baseline

Table 3: The number of detectable lists, equivalence sets, and testable rules for each attack.

	AdGuard			uBlock Origin		
	Lists	**Equiv. Sets**	**Rules**	**Lists**	**Equiv. Sets**	**Rules**
All rules	77	497	1,047,019 (100%)	66	282	585,536 (100%)
Generic rules	71	418	604,868 (57.7%)	65	216	354,840 (60.6%)
Generic network rules	70	393	444,878 (42.5%)	63	197	293,390 (50.1%)
Generic cosmetic rules	58	143	159,990 (15.3%)	54	98	61,450 (10.5%)
Attacks						
Baseline attack	76	494	1,044,270 (99.7%)	65	280	583,267 (99.6%)
CSS animation attack (+ others)	69	406	577,494 (55.2%)	63	212	336,332 (57.4%)

Table 4: Statistics of attacks on collected datasets. m is the size of the fingerprinting template (number of rules to test for), and m_{max} is the maximum fingerprinting template across different users. We group all attacks exploiting all generic rules into one row similar to CSS animation attack under "+ others".

Dataset	**Attack**	**N, Equiv. Sets**	**Entropy**	**Unique Users**	**Targeted Fingerprinting**		**General Fingerprinting**		
					Unique Users	m_{max}	Entropy	Unique Users	m
ADGUARD	Baseline attack	494	0.73	4931 (26.66%)	3865 (20.90%)	27	0.63	3865 (20.90%)	87
	CSS animation attack (+ others)	406	0.72	4605 (24.90%)	3570 (19.30%)	30	0.62	3570 (19.30%)	87
	Generic cosmetic rules only	143	0.69	4114 (22.25%)	2855 (16.92%)	29	0.58	2,855 (16.92%)	74
UBLOCK	Baseline attack	280	0.56	1191 (20.22%)	977 (16.59%)	39	0.46	977 (16.59%)	59
	CSS animation attack (+ others)	212	0.56	1151 (19.54%)	932 (15.82%)	37	0.45	932 (15.82%)	62
	Generic cosmetic rules only	98	0.51	881 (14.96%)	675 (11.46%)	35	0.40	675 (11.46%)	54

attack (1.76% less for ADGUARD and 0.68% less for UBLOCK), even while only requiring *generic* rules. For both general and targeted fingerprinting, the entropy and uniqueness decrease (by around 0.1 for entropy in both datasets) as expected – with added constraints on the template size – but remain relatively high, similar to prior attacks [33, 44]. Seeing that entropy decreases less sharply than uniqueness means that many small-size anonymity sets are still present in the dataset – which is supported by looking at Figure 2.

Finally, looking at the distribution of anonymity set size compared to the distance of the user's filter list configuration from the default in Figure 2, we can distinguish three major user communities: (Upper Left) most users who slightly customize their filter lists (less than 15 changes) and remain mostly anonymous (anonymity size $\geq$ 128 users); (Lower Left) some users who become more identifiable (anonymity size $\leq$ 128 users) despite editing only a few filter lists – this mainly includes users that choose very rare filter list combinations; and (Lower Right) advanced users fully customizing their ad-blockers for optimal privacy and incurring high identifiability. The downward arc defining the relationship between anonymity set size and filter-list customization confirms our hypothesis that users with advanced blocking are more susceptible to fingerprinting.

6.3 Fingerprint Stability

A unique fingerprint is not useful if it is not persistent over time. The stability of a fingerprint is essential for an attacker

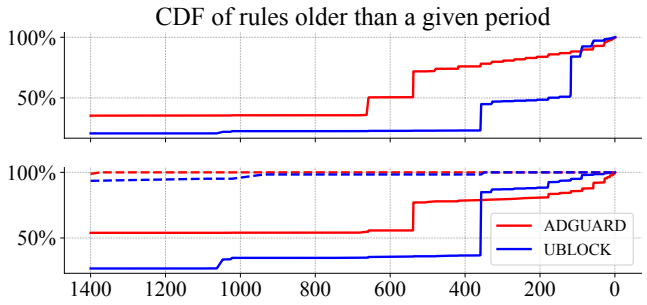

Figure 3: Cumulative Distribution of the age of filter rules usable in general fingerprinting. (Top) All usable rules. (Bottom) Only for CSS animation attack. The dashed lines represent equivalence set age distribution.

to link visits of the same user across time and monitor their browsing history. Two predominant factors can change a user's ad-blocker fingerprint over time: (A) maintainers updating the rules in filter lists and (B) users changing their subscriptions. For every filter-list update (A), the attacker must re-run their fingerprinting algorithm to update their templates (Section 4). For (B), at best, the adversary can use proposed techniques to link evolving fingerprints of the user [69] with varying success.

(A) Filter-Rule Stability. We track the stability of the rules chosen by the general fingerprinting algorithm (later called GF rules) during the last 1,400 days since data collection (three years and 10 months). Checking every commit would be pro-

hibitively expensive given their high frequency and variability – 21 minutes inter-commit time on average for AdGuard (σ = 1.32 hrs) and 30 minutes for uBlock Origin (σ = 43 mins). As a lower bound on stability, we sample filter-list versions at specific dates – with intervals between dates increasing further back in time – and check if the rules have changed. Furthermore, rules can be older than our estimation, as imperfect or non-existent versioning of the filter-list repositories can prematurely stop tracking them. More details in Appendix C.2.

Results. First, a large portion of GF rules survives the 1,400 days: 35% (97.9K rules) for AdGuard and 21% (45.6K rules) for uBlock Origin across all attacks; and 54% (50.9K rules) for AdGuard and 26% (10.8K rules) for uBlock Origin across attacks. We plot the age distribution of GF rules that disappear in Figure 3. For AdGuard, 72% of GF rules are at least 1.5 years old. For uBlock Origin, as updates are more frequent, 90% of GF rules are at least 3 months old and only 23% older than a year. Most rules disappear at specific time points, corresponding to plot jumps. Manual investigations show that these jumps are generally due to file renames and GitHub mirrors not existing before those dates (examples in Appendix C.2). Focusing only on the CSS animation attack in Figure 3 (Bottom), we see more stability in GF rules, which exclude the effect of domain-specific rules exploitable by Baseline attack in Figure 3 (Top): 79% of AdGuard's GF rules and 85% of uBlock Origin's GF rules survive for at least 357 days (almost a year). This result aligns with our ad-blocker forum observations: maintainers avoid changing generic rules and introduce exceptions and site-specific rules.

Moreover, the adversary only cares about the stability of equivalence sets (Section 4.3). The best adversary strategy to identify the equivalence set is to choose the set's historically old rules. In our datasets, most equivalence sets are stable in the long term. For example, 99% of equivalence sets survive the 1,400 days for AdGuard and 93% for uBlock Origin with the CSS animation attack (Figure 3). Overall, this stability is long enough for the attacker to update their templates across the websites they control. The redundancy of rules in equivalence sets prolongs the adversary's use of the same fingerprinting template even if a specific rule expires.

(B) User Subscription Stability. Our datasets do not allow us to faithfully study users' subscription stability, as we only have snapshots of the subscriptions they post on the forums. Measuring the stability of user subscriptions would require a comprehensive longitudinal study in which we would survey the user's filter-list subscriptions.

6.4 Utility of *Domain-Specific* Filter Rules

So far, we have focused on the impact of *generic* rules on the fingerprinting template. In this section, we assume a *hypothetical worst-case adversary*, able to collaborate with arbitrary websites, to study the impact of domain-specific rules (exploitable by the Baseline attack) on filter-list coverage and fingerprint quality. We quantify the impact of domain-specific rules with the following approach: First, we extract from the domain-specific rules all "activating" domains of these rules. Then, we count the number of rules each domain can "activate" for each filter list. Iterating over domains is not straightforward, as we need to account for subdomains and top-level domain relationships. Finally, we exclude already identifiable filter lists with generic rules and count the number of new ones that become identifiable with domain-specific rules for selected domains. We share more about the method in Appendix C.3.

Results. We summarize the results for AdGuard in Table 5. The table showcases statistics over two main breakdowns. (1) breakdown by rule type. (2) breakdown by domain rank according to Tranco [42][4]. We do a breakdown by domain rank, as we observe that some popular domains cover more lists but might be unfit or not need this type of tracking (e.g., `translate.google.*`). The top 1K domains vaguely account for "popular" domains, as taken by multiple measurement studies [27, 38, 39, 53]. A significant result is that the adversary must control less than 13 domains to cover more than 99% of filter lists across all rule types and domain ranks. An interesting observation is that far more cosmetic rules are domain-specific compared to blocking rules, contrary to what we observe for generic rules (Table 3). So attacks relying on cosmetic rules can benefit more from domain-specific rules than those relying on blocking rules, *e.g.*, with only one domain, the adversary can detect 15 more lists with cosmetic rules compared to only 6 more for blocking rules. Finally, looking at Table 4 and Table 3, we see that while the Baseline attack can detect seven more lists than Lazy loading attack, they both achieve close entropy and uniqueness results. This signals that the attacker's benefit from domain-specific rules is limited, and that more realistic still adversaries achieve comparable success.

6.5 Attack Detectability

We analyze how well *SoTA* fingerprinting detectors can detect our attacks. The most popular form of detectors uses machine learning to classify specific JS scripts as fingerprinting scripts or not [72], *e.g.*, like *FP-Inspector* [39], *FP-RADAR* [17], and *Essential-FP* [57]. This method can only detect the Baseline attack, as the other attacks are scriptless.

Another type of detector monitors and classifies network requests as fingerprinting or not based on their content (*e.g.*,, URL parameters, headers, *etc.*), like *FingerprintAlert* [16]. As our adversary only monitors if attack requests reach or not as one-bit signals, they do not carry any content that could serve as input for the classifier. An alternative would be to use network volume as a proxy, but such an approach risks raising too many false positives: The interquartile range of requests per web page is 43 to 123; sites with 29 to 80 additional requests from the attack still overlap with 50% of benign sites. We

[4] Available on https://tranco-list.eu/list/24P99/1000

Table 5: The impact of controlling specific domains on the filter-list coverage for AdGuard, broken down by rule type.

Rule Type	*All*		*Blocking*		*Cosmetic*	
Domain Rank	All	$> 1K$	All	$> 1K$	All	$> 1K$
Unique Domains	137,760	137,147	27,007	26,609	128,123	127,541
Median Coverage[1]	2	2	1	1	2	2
Max. Coverage[2]	45	45	18	18	41	41
Baseline Identifiable Lists	71 / 84	71 / 84	70 / 84	70 / 84	58 / 84	58 / 84
Max. +L(1)[3]	+13 (100%)	+5 (90%)	+6 (90%)	+6 (90%)	+15 (87%)	+6 (76%)
Max. +L(∞)[4]	+13 (100%)	+12 (99%)	+13 (99%)	+13 (99%)	+22 (95%)	+22 (95%)
Min. Domains[5]	1	5	5	5	4	12

[1] The median number of filter lists for which a domain has rules.
[2] The maximum number of filter lists for which a domain has rules.
[3] The maximum number of additional filter lists identifiable after controlling **one** domain.
[4] The maximum number of additional filter lists identifiable after controlling as many domains as the adversary needs. The percentage refers to the portion of the total lists covered.
[5] The minimum number of domains the adversary needs to control to achieve *Max. +L(∞)*.

also note that the number of attack requests can be further reduced by changing the geometric construction of attacks (each request corresponds to a specific combination of blocked elements) [44], becoming more similar to benign sites' traffic.

Finally, some ML-based approaches (like *WebGraph* [53] and *AdGraph* [38]) model the relationships between web-page elements, requests, and scripts for stateful tracking. These approaches do not target fingerprinting in their current form. Yet, we posit that adapting them to classify fingerprinting scripts is a promising avenue for future work.

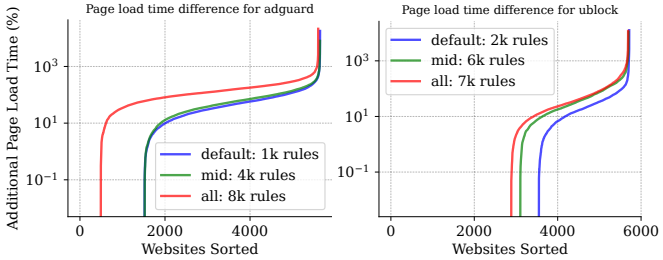

Figure 4: Page additional load time portion compared to loading it without ad blockers in (%) for AdGuard (**Left**) and uBlock Origin (**Right**).

7 Mitgation Resistance

While the proposed attacks can avoid detection by current means, in this section, we explore several techniques filter-list maintainers and browsers can perform to protect against potential filter-list fingerprinting attacks. Any fingerprinting mitigation technique falls into one of three categories: **uniformity** (all users have the same fingerprint), **randomization** (randomize the fingerprinting features for a single user), and **blocking** (preventing the attacker's access to the sources of the fingerprinting features) [72]. Contrary to other fingerprinting paradigms, ad blockers cannot arbitrarily randomize what requests they block or not. On the one hand, any allowed `ATS` request can have profound privacy implications. On the other hand, the usability and consistency of the ad-blockers are crucial for user adoption [50]. In a recent user survey, Nisenoff et al. [49] showed that most users faced with "abnormal" ad-blocker behavior take actions that expose them to more trackers: 69% of participants would disable the ad-blocker on the page, 8% would disable it globally, and 7% would uninstall the ad-blocker completely. So, we disregard randomization and only explore uniformity and blocking techniques. Another easy-to-dismiss filter-list maintainer mitigation is adding ad-blocker rules to block the attacker: Blocking attacker resources is a common ad-blocker strategy against evasive `ATS` and anti-ad-block providers [43]. Such strategies quickly become a cat-and-mouse game of never-ending updates (of maintainer filter rules and attack domains or parameters by the adversary) [59].

7.1 Disallowing Filter-List Customization

One seemingly obvious solution is limiting filter-list customization altogether, either by restricting users to fewer filter lists than before (*e.g.*, the default filter lists) or more filter lists than they would like (*e.g.*, all filter lists). This approach causes all users to have the same fingerprint, abolishing the threat of filter-list fingerprinting. However, we identify three drawbacks pertinent to the ad-blocker usability: (1) breakages due to bad rules in some filter lists will affect all users and decrease ad-blocker adoption [26, 49, 58]; (2)specific filter lists tackle needs that are not universally accepted – like the *Fanboy Annoyances* list, which removes additional elements that clutter the page, although they are not ads – and certain users would consider them examples of breakage or against their browsing expectations; (3) enabling all lists carries a performance overhead for the ad-blocker and the user's browser for every new page [59]. As users already expressed dissatisfaction with unexpected ad-blocker behavior [49], we leave the quantifica-

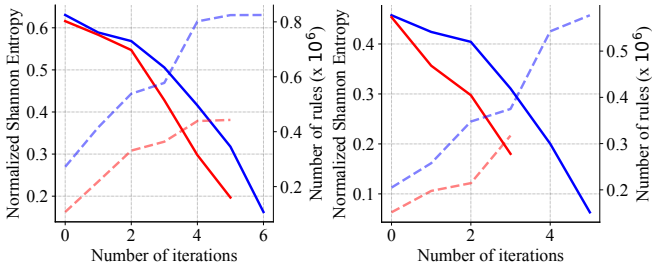

Figure 5: Iterations of increasing ad-blocker robustness by fixing rules for all users leading to a decrease in fingerprint entropy for AdGuard (**Left**) and uBlock Origin (**Right**). *Legend: Normalized entropy in solid lines, Number of rules in dashed lines, Baseline attack, and CSS animation attack.*

tion of the impact on user adoption and satisfaction through a comprehensive user study for future work. However, the performance overhead is something we can quantify. We did not find any study that measures the impact of increasing filter-rule count on the page load time. So, we adapt the methodology from Roongta and Greenstadt [50], who measure various performance metrics over multiple ad blockers, and conduct the analysis on their same set of sites from Tranco [42] – the details about the setup are in Appendix C.4.

Results. We report load time distribution across websites for both ad blockers in Figure 4. First, we observe that uBlock Origin handles the increase in rule count better than AdGuard : runtime only increases by a median of 0.052 seconds for uBlock Origin and 514K additional rules, compared to 2.8 seconds for AdGuard and 765K additional rules. We also note that the increase in load-time is not linear, *e.g.*, for AdGuard, using 119K or 417K rules is almost identical. We attribute this to the fact that less popular lists – when included in the "all" filter-list configuration – possess less optimized rules (*e.g.*, more complex regex rules) due to less scrutiny from the community. Overall, this approach might not impact uBlock Origin users as much as AdGuard users in terms of performance, but it will still significantly impede user experience and cause breakage.

7.2 Increasing Filter-List Robustness

Rather than limiting users' filter-list choices, maintainers could modify the lists to be more robust to fingerprinting by removing rules valuable for the attacks and globally enabling them for all users. Since the attacker is not exhaustive and uses constrained templates, if they re-run the algorithm, they might be able to extract new rules from the updated lists. To evaluate this defense's success, we simulate iterations of this game between the attacker and the maintainers, with the strategies described earlier, until the attacker utility (entropy or uniqueness) drops below a certain threshold or no more rules to extract.

Results. We plot the results in Figure 5. First, AdGuard requires enabling far more global rules than uBlock Origin to reduce entropy below 0.2. Against general fingerprinting with the Baseline attack, AdGuard requires 820K rules, which is 6.8 times the maximum filter-list size, while uBlock Origin requires 577K rules, which should not cause any problem with load-time, as we saw in the load-time experiment Figure 4. The CSS animation attack requires less than 442K rules for AdGuard and 317K for uBlock Origin, which is still a significant overhead but manageable. Also note that the same concerns about breakage and user experience from the prior defense (Section 7.1) still apply, although less severely. *Overall, this mitigation is a promising candidate for future work.*

7.3 Disabling Browser Features

Browsers can try to block browser features essential for the success of the proposed attacks. However, limiting browser functionality is an aggressive approach that can harm the user experience and cause breakage [39, 41]. We measure the prevalence of these features on the web to evaluate the impact of disabling them, similarly to Lin et al. [44]. We adopt the setup from Section 7.1 and successfully crawl 22,745 web pages initially listed by Roongta and Greenstadt [50].

Results. For the complete overview of our findings, we refer the reader to Table A2 in the appendix. In summary, the attacks' high-level features are highly prevalent: 77% of pages use CSS animations in their implementations. Looking at combinations of features and low-level attributes for some attacks, a small share of pages utilize them for usual purposes. We find 83 pages that call HTTP requests in animation sequences – usually to animate the rendered background in loops. However, blocking these combinations requires case-by-case filtering at runtime, which can be computationally expensive, *e.g.*, checking if the animation changes the background image URL requires enumerating all animation sequences (33 σ = 132 on average).

As a takeaway, limiting browser features is a possible option, especially for experimental features. Nevertheless, it can have significant overhead on page load time for granular features.

8 Discussion

In this section, we discuss the implications of presented results in Section 6 regarding the impact of filter-list fingerprinting attacks on the relationship between ad-blocking and privacy.

8.1 Revisiting the Privacy Extensions Dilemma

Our work underscores the conflicting duality within privacy protections, where efforts to *enhance* privacy can have *detrimental* effects on privacy. In our case, enhancing tracking protection through custom filter list blocking can reduce user anonymity through filter list fingerprinting.

This conflict is becoming increasingly relevant as users rely on multiple PETs with high configurability to protect themselves [66]. To our knowledge, only the work by Gulyas

et al. [33] has discussed this trade-off, in the context of web-extension fingerprinting. The authors conclude that the benefit of `ATS` blocking still outweighs the harm of fingerprinting ad-blocker extensions. This conclusion relies on the fact that excluding the ad-blocker from the set of fingerprinted extensions does not decrease the fingerprint entropy. This might be true when extension fingerprinting attacks extract at most one bit from the presence or absence of extensions. However, our work shows that it is possible to obtain multiple bits, as each enabled filter list provides another bit of entropy. This increase in information increases the privacy cost of installing the extension, especially for privacy-aware individuals with more unique configurations (see Figure 2). On one hand, this risk cannot be brushed off easily (Section 7), users must pay an inevitable cost every time they deviate from the popular ad-blocker configurations to more specialized ones. On the other hand, limiting all users to the same filter lists is not straightforward (Section 7.1): users do not agree on the boundary between content that should be blocked or not, hence the fragmentation into multiple filter lists. For example, the `Fanboy's annoyances` list removes "annoying" elements but is not necessarily `ATS`-related like "Share on Facebook" buttons. Given the active maintenance and large following of this list, we understand that disabling it for all users will negatively impact the ad-blocking community. Thus, addressing the privacy trade-off is non-trivial and requires further exploration and attention from the community. In sum, contrary to Gulyas et al. [33], we argue that the dilemma is relevant and research looking into the privacy of ad blockers should keep this angle in mind. We expand on our recommendations to the community in Section 8.4.

8.2 Applicability beyond ad-blockers

In this work, we have focused on studying the information extractable from ad blockers – concretely from their filter-rule syntax. To this end, we have developed CSS and HTML attacks that reveal whether a rule (with a given syntax, scope, or type) is active in the browser. Our evaluation of two popular ad-blockers (AdGuard and uBlock Origin) proves that configurable extensions leave a sufficiently fingerprintable footprint.

We now investigate whether our multi-bit fingerprinting paradigm and attacks apply to non-ad-blocker extensions and privacy browsers. For this, we identified ten extensions with granular configurations among the most popular extensions on the Chrome Web Store in three categories: **(A) Security and phishing protection**: NoScript (100K users), MalwareByte Guard (8M users), Netcraft (60K users). **(B) Productivity and distraction blocking**: NBlocksite (1M users), LeechBlock NG (80K users), MetaCert (10K users), Mindful Browsing (10K users), StayFocused (500K users). **(C) Browser theming**: Stylus (700K users) and Tampermonkey (11M users).

Most of these extensions do not fit our threat model for fingerprinting: *MalwareByte Guard* has coarse and ill-defined settings (*e.g.*, "blocking malicious scripts") that are not connected to deterministic, fingerprintable page manipulations; *StayFocused* only allows page modifications for specific sites (YouTube and TikTok) and thus does not enable tracking on the web as a whole; *Netcraft, Blocksite, LeechBlock, MetaCert, CleanBrowsing, Mindful Browsing* only block pages when they are directly visited. While, theoretically, the presence could be detected if their blocking occurred for subresources such as iframes, given the extensions' choice of blocking only top-level navigation means our threat model cannot be applied.

For the remaining two extensions, *Tampermonkey* and *Stylus*, the general principle of inferring multiple bits of entropy based on their configuration applies. Although the concrete implementation of the attacks is orthogonal to what our paper proposes, we nonetheless outline here how this could be done.

Tampermonkey. allows to set and execute custom user scripts when loading web pages. These scripts can impact CSS and HTML in many more different ways than filter-list blocking because JS code is more expressive than filter rule syntax. A possibility to automate the script-detection test is the following: (1) first find all element selector statements (*e.g.*, `document.querySelector(".main")`), (2) run the script on a test page containing elements for each selector, and (3) compare the element's attributes with and without the script and use those that vary to test whether the script is active. This attack can be made scriptless by setting CSS rules to check for specific element states (*e.g.*, add a background image hosted by the adversary only for `.main` if the element has small height).

Stylus. allows users to set custom CSS style sheets to modify the browser theme. *Stylus*' sheets allow for more arbitrary modifications to web elements than filter lists. Checking whether specific styles are applied with JS is possible by querying the `.style` dictionary of the element. Alternatively, it is possible to detect certain stylesheets without a script: Either by using the CSS container queries (Appendix A) to trigger background image URLs pointing to the attacker domain, only if a specific style on the element is applied (*e.g.*, `style(color: red)`); or, if the stylesheet contains element hiding statements, applying the CSS animation attack on elements with this feature.

Finally, privacy browsers also contain particular configurations. This is the case of the Brave Browser [19], which uses filter lists, they highly overlap with both AdGuard and uBlock Origin, and therefore our attacks apply seamlessly and would likely obtain similar results.

8.3 Limitations and Future Work

The experimental limitations that we faced included approximation decisions to make the analysis feasible. Notably, while studying the change in filter-list versions (Section 6.3), some lists do not have a versioning system (like GitHub), so we either find mirrored versions on GitHub if possible or omit them completely. Regarding comparing rules, we only rely on syntactical similarity (whether they look the same or not). However, some rules might be functionally similar but look

different (*e.g.*, both `##.ad` and `##div.ad` block `div` elements with class `.ad`). We do not see a straightforward way to functionally compare two rules other than attempting to apply them on a test website, which would make the analysis prohibitively expensive. Regarding studied filter lists, we omit any third-party list the user might install from a URL in the ad blocker, as the analysis would not be tractable. However, in such cases, users would become more identifiable, and the analysis remains a valid lower bound on uniqueness. Finally, regarding our measurement studies in Section 7, we are subject to the same limitations regarding the choices of sites as Roongta and Greenstadt [50]. We would like to see future studies conduct more comprehensive crawls to validate the impact of mitigation on website usability, breakage, and performance.

8.4 Recommendations

In this section, we detail how the various parties involved in the ad-blocking space can benefit from our method and results to implement precautionary measures against the attacks presented in Section 4.2.

Ad-blocker Maintainers. Ad-blocker maintainers could add our fingerprinting analysis pipeline described in Section 4 to continuously evaluate the fingerprinting susceptibility of their user base and filter lists. They then could use the iterative robustness defense proposed in Section 7.2 to reinforce their filter-list against identifying rules. As a more general guideline, filter-list maintainers should limit the number of generic network rules as they are the easiest to fingerprint (Section 6) or at least allow any image endpoint of blocked domains if the case permits to prevent attacks relying on `<img>` elements (Appendix A). Regarding the CSS animation attack, the success of the attack relies on the fact that ad blockers set the `display` attribute to `none`, so they can experiment with different ways to hide the element like playing around with `visibility`, `z-index`, and `pointer-events`. Finally, AdGuard should account for the leak of browser history associated with automatically enabling filter lists.

Privacy-Aware Users. Users aiming for better privacy protection should not have the false idea that more tools mean more privacy. While choosing PET web extensions and configuring them, users should be cautious in making their web behavior stand out completely, *e.g.*, including custom user rules or third-party lists could become a user identifier.

Browsers. Browsers should be careful in adopting new CSS directives that can trigger conditionally based on network events or granular web page states. For instance, the optimizations regarding fetching `background-image` requests based on the visibility of the DOM element should be more carefully investigated. Also, browsers should further investigate the CSS `container` query attack as it forms a new covert channel between the CSS code and the rest of the browser's environment. We describe such an attack further in Appendix A.

9 Conclusion

By subscribing to additional filter lists, users can optimize the ad-blocker to their personal browsing pattern (*e.g.*, page languages) and their tolerance of breakage versus more aggressive blocking. What at first appears to be beneficial for privacy through more aggressive filtering, however, as our work highlights, has detrimental effects on a user's privacy. We propose several attacks (Section 4.2) that allow uniquely identifying between 70% and 84% of the user's active filter lists without requiring easy-to-detect scripts.

We show that no more than 87 generic rules out of hundreds of thousands are sufficient to recover fingerprints with identical entropy to extracting all lists from original datasets (0.72 for `ADGUARD` and 0.58 for `UBLOCK`) (Table 4). In conclusion, customizing filter lists can reduce the anonymity sets of privacy-aware users to less than a median of 48 users (0.2% of the population) with just 45 rules with the CSS animation attack) further than traditional fingerprinting attacks for which they implement defenses.

We also show that it is hard to mitigate the proposed attacks without penalty: users have to either pay in privacy (*e.g.*, choosing to allow certain `ATS`) or performance and usability challenges (e.g., adding niche rules for all users results in breakage for users not benefiting from the rules). Yet, we propose precautionary measures (Section 8.4) that users, ad-blockers, and browsers can take to minimize the privacy risk.

We believe that the fingerprintability of ad-blocker customization is a practical example of the privacy harm that can arise when (ab)using Web PETs without appropriate care. Further work should investigate hidden privacy compromises in other privacy extensions, like *Privacy Badger* [8], VPNs, Link Scanners, *etc*., both separately and in combination. This will help the community better understand the privacy costs associated with layering customized web-PETs.

Acknowledgments. We thank Jannis Rautenstrauch for his early and detailed feedback on the manuscript and Sandra Siby for her fruitful discussions and feedback on the work. We also thank AdGuard's Team for sharing internal statistics.

Ethical Considerations

We acknowledge the importance of the ethical implications of conducting research, as reinforced by USENIX. Our work includes diverse experiments and data sources that we ensure are ethically valid. We detail the ethical considerations regarding our experiments in the following.

User data sources. we scrape publicly accessible GitHub forums for open-source (GPL-3) ad-blockers. Scraping GitHub forums is standard practice in the privacy and security community [26, 58], and only processes publicly accessible resources. To ensure that public information about the forum authors cannot be used to harm users with our proposed attacks, we

do not store any author's personal information in our pipeline and randomize the identifiers while ensuring that the identifiers for posts by the same user are the same. In addition, ad-blocker maintainers employ proprietary methods that reduce personal information in posts, such as moderation and submissions through bot intermediaries.

Proposed attacks. The attacks we propose do not rely on specific browser or ad-blocker vulnerabilities that were previously undetected. Instead, we present simple techniques to utilize CSS to demonstrate the inherent susceptibility of ad blockers to detection. We provide attacks similar to prior stylistic attacks previously known to the community, some of which are already implemented [1]. As other fingerprinting works, we provide these attacks as examples to raise awareness for the larger security and privacy community and ad-blocker users about the privacy cost of over-configuring ad blockers. As part of our ethical duty, we explore ways to detect these attacks, try to mitigate them and provide recommendations for appropriate protection against their potential risk. We also provide our pipeline to identify problematic filter rules and eliminate them. Overall, we believe the awareness benefit of disclosing this risk to the community outweighs the potential attack implementation – especially to ad-blocker maintainers and users. This is consistent with the tracking literature, which states that simply describing browser fingerprinting channels does not require a private disclosure to immediate stakeholders.

Web measurements. We adopt the same precautions as the large body of web measurement studies. We visit 22K web pages at most thrice in sequence, giving ample delay (30 seconds) between consecutive visits. We only visit publicly accessible pages, which do not require any prior consent from the servers. We use only 10 parallel crawlers to constrain the additional internet traffic volume. Also, we only load the pages passively by loading the page and any associated resources for about 10 seconds. Our crawlers do not interact with the web server in any other way. All additional operations happen on the front end in our browser instances without impacting the services. Finally, the data we generate from the pages is purely functional, including timing information and HTML element counting. Overall, we believe our crawling techniques do not harm servers, ad blockers, or other users.

Open Science

Adhering to the Open Science Policy, we make this work's artifacts (experiment code and data) available in zenodo.org/records/14736725. We also host the experiments' code on GitHub at spring-epfl/flfp. We also provide proo-of-concept attack implementations and host exemplary honey pages on flfp-demo.github.io. The source code at flfp-demo/flfp-demo-builder. We do not implement the Baseline attack as it requires collusion with 3rd-party domains. The attacks themselves use no JavaScript, but we include scripts to simulate the attacker server through *service workers* and visualize the output. We manually hard-coded the rules to test for demonstration.

References

[1] How ad blockers can be used for browser fingerprinting, . URL https://fingerprint.com/blog/ad-blocker-fingerprinting/.

[2] Fingerprintjs/fingerprintjs, . URL https://github.com/fingerprintjs/fingerprintjs.

[3] MDN Web Docs. URL https://developer.mozilla.org.

[4] EasyList. URL https://easylist.to/easylist/easylist.txt.

[5] Ghostery, . URL https://www.ghostery.com.

[6] Ghostery Github Issues, . URL https://github.com/ghostery/adblocker/issues.

[7] Peter lowe's hostname blocklist. URL https://pgl.yoyo.org/as/.

[8] Privacy Badger. URL https://privacybadger.org/.

[9] Ad Blocker Usage and Demographic Statistics in 2024, 2024. URL https://backlinko.com/ad-blockers-users.

[10] Adblock Plus. Adblock plus: How to write filters, 2021. URL https://help.adblockplus.org/hc/en-us/articles/360062733293-How-to-write-filters.

[11] AdGuard Software Ltd. AdGuard filters, . URL https://adguard.com/kb/general/ad-filtering/adguard-filters/.

[12] AdGuard Software Ltd. AdGuard AdBlocker Browser Extension, . URL https://adguard.com/en/adguard-browser-extension/overview.html.

[13] AdGuard Software Ltd. Adguard AdguardFilter Github Issues, . URL https://github.com/AdguardTeam/AdguardFilters/issues.

[14] AdGuard Software Ltd. How to create your own ad filters, . URL https://adguard.com/kb/general/ad-filtering/create-own-filters/.

[15] S. Agarwal, A. Fass, and B. Stock. Peeking through the window: Fingerprinting browser extensions through page-visible execution traces and interactions. *ACM CCS*, 2024.

[16] N. M. Al-Fannah. *Towards A Better Understanding of Browser Fingerprinting*. PhD thesis, Royal Holloway, University of London, 2019.

[17] P. N. Bahrami, U. Iqbal, and Z. Shafiq. Fp-radar: Longitudinal measurement and early detection of browser fingerprinting. *PETS*, 2021.

[18] Brave. Brave community, . URL https://community.brave.com/c/support-and-troubleshooting/ad-blocking/78.

[19] Brave. Advanced privacy, . URL https://brave.com/privacy-features.

[20] Brave Community. Possible browser fingerprinting using adblock filter lists, 2022. URL https://community.brave.com/t/possible-browser-fingerprinting-using-adblock-filter-lists/365908.

[21] BrowserLeaks. Content Filters and Proxy Detection. URL https://browserleaks.com/proxy.

[22] I. Castell-Uroz, R. Sanz-García, J. Solé-Pareta, and P. Barlet-Ros. Demystifying Content-Blockers: Measuring Their Impact on Performance and Quality of Experience. *IEEE TNSM*, 2022.

[23] S. Chalise, H. D. Nguyen, and P. Vadrevu. Your speaker or my snooper?: Measuring the effectiveness of web audio browser fingerprints. In *ACM IMC*, 2022.

[24] A. Datta, J. Lu, and M. C. Tschantz. Evaluating Anti-Fingerprinting Privacy Enhancing Technologies. In *ACM WWW*, 2019.

[25] DebugBear. Chrome Extension Statistics, 2024. URL https://www.debugbear.com/blog/chrome-extension-statistics.

[26] S. El Hajj Chehade, S. Siby, and C. Troncoso. Sinbad: Saliency-informed detection of breakage caused by ad blocking. In *IEEE S&P*, 2024.

[27] S. Englehardt and A. Narayanan. Online Tracking: A 1-million-site Measurement and Analysis. In *ACM CCS*, 2016.

[28] B. Eriksson, P. Picazo-Sanchez, and A. Sabelfeld. Hardening the security analysis of browser extensions. In *ACM SIGAPP*, 2022.

[29] Fingerprint.com. Server-side API. URL https://dev.fingerprint.com/.

[30] Free Software Foundation Inc. Open Adblock Plus forums. URL https://forum.adblockplus.org/viewforum.php?f=6.

[31] Free Software Foundation Inc. Adblock Plus, 2024. URL https://adblockplus.org/.

[32] G. G. Gulyas, G. Acs, and C. Castelluccia. Near-Optimal Fingerprinting with Constraints. In *PETS*, 2016.

[33] G. G. Gulyas, D. F. Some, N. Bielova, and C. Castelluccia. To Extend or not to Extend: On the Uniqueness of Browser Extensions and Web Logins. In *PETS*, 2018.

[34] A. Gómez-Boix, P. Laperdrix, and B. Baudry. Hiding in the Crowd: An Analysis of the Effectiveness of Browser Fingerprinting at Large Scale. In *ACM WWW*, 2018.

[35] M. Heiderich, M. Niemietz, F. Schuster, T. Holz, and J. Schwenk. Scriptless attacks: stealing the pie without touching the sill. In *ACM CCS*, 2012.

[36] HTTP Archive. HTTP Archive: Page Weight. Technical report. URL https://httparchive.org/reports/page-weight?lens=top1m&start=2018_09_01&end=latest.

[37] U. Iqbal, Z. Shafiq, and Z. Qian. The ad wars: retrospective measurement and analysis of anti-adblock filter lists. In *ACM IMC*, 2017.

[38] U. Iqbal, Z. Shafiq, P. Snyder, S. Zhu, Z. Qian, and B. Livshits. Adgraph: A machine learning approach to automatic and effective adblocking. *IEEE S&P*, 2020.

[39] U. Iqbal, S. Englehardt, and Z. Shafiq. Fingerprinting the Fingerprinters: Learning to Detect Browser Fingerprinting Behaviors. In *IEEE S&P*, 2021.

[40] S. Karami, P. Ilia, K. Solomos, and J. Polakis. Carnus: Exploring the privacy threats of browser extension fingerprinting. In *NDSS*, 2020.

[41] P. Laperdrix, N. Bielova, B. Baudry, and G. Avoine. Browser Fingerprinting: A Survey. *ACM TWEB*, 2020.

[42] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *NDSS*, 2019.

[43] S.-C. Lin, K.-H. Chou, Y. Chen, H.-C. Hsiao, D. Cassel, L. Bauer, and L. Jia. Investigating advertisers' domain-changing behaviors and their impacts on ad-blocker filter lists. In *ACM WWW*, 2022.

[44] X. Lin, F. Araujo, T. Taylor, J. Jang, and J. Polakis. Fashion Faux Pas: Implicit Stylistic Fingerprints for Bypassing Browsers' Anti-Fingerprinting Defenses. In *IEEE S&P*, 2023.

[45] MDN. Redirect tracking protection. URL https://developer.mozilla.org/en-US/docs/Mozilla/Firefox/Privacy/Redirect_Tracking_Protection.

[46] MediaWiki. No-JavaScript notes, 2024. URL https://www.mediawiki.org/wiki/No-JavaScript_notes.

[47] Microsoft Edge Team. Introducing tracking prevention, now available in Microsoft Edge preview builds. URL https://blogs.windows.com/msedgedev/2019/06/27/tracking-prevention-microsoft-edge-preview/.

[48] S. Munir, S. Siby, U. Iqbal, S. Englehardt, Z. Shafiq, and C. Troncoso. Cookiegraph: Understanding and detecting first-party tracking cookies. In *ACM SIGSAC*, 2023.

[49] A. Nisenoff, A. Borem, M. Pickering, G. Nakanishi, M. Thumpasery, and B. Ur. Defining "broken": User experiences and remediation tactics when Ad-Blocking or Tracking-Protection tools break a Website's user experience. In *USENIX*, 2023.

[50] R. Roongta and R. Greenstadt. From user insights to actionable metrics: A user-focused evaluation of privacy-preserving browser extensions. In *ACM Asia CCS*, 2024.

[51] T. Saito, K. Takahashi, K. Yasuda, K. Tanabe, M. Taneoka, and R. Hosoya. Tor fingerprinting: Tor browser can mitigate browser fingerprinting? In L. Barolli, T. Enokido, and M. Takizawa, editors, *NBiS*, 2018.

[52] I. Sanchez-Rola, I. Santos, and D. Balzarotti. Extension breakdown: Security analysis of browsers extension resources control policies. In *USENIX*, 2017.

[53] S. Siby, U. Iqbal, S. Englehardt, Z. Shafiq, and C. Troncoso. WebGraph: Capturing Advertising and Tracking Information Flows for Robust Blocking. In *USENIX*, 2022.

[54] Simple Analytics. Collect without JavaScript. URL https://docs.simpleanalytics.com/without-javascript.

[55] A. Sjösten, S. Van Acker, and A. Sabelfeld. Discovering browser extensions via web accessible resources. In *ACM CODASPY*, 2017.

[56] A. Sjösten, S. Van Acker, P. Picazo-Sanchez, and A. Sabelfeld. Latex gloves: Protecting browser extensions from probing and revelation attacks. In *NDSS*, 2019.

[57] A. Sjosten, D. Hedin, and A. Sabelfeld. EssentialFP: Exposing the Essence of Browser Fingerprinting. In *EuroS&PW*, 2021.

[58] M. Smith, P. Snyder, M. Haller, B. Livshits, D. Stefan, and H. Haddadi. Blocked or broken? automatically detecting when privacy interventions break websites. *PETS*, 2022.

[59] P. Snyder, A. Vastel, and B. Livshits. Who filters the filters: Understanding the growth, usefulness and efficiency of crowdsourced ad blocking. *ACM POMACS*, 2020.

[60] K. Solomos, P. Ilia, S. Karami, N. Nikiforakis, and J. Polakis. The dangers of human touch: Fingerprinting browser extensions through user actions. In *USENIX*, 2022.

[61] K. Solomos, N. Nikiforakis, and J. Polakis. Harnessing multiplicity: Granular browser extension fingerprinting through user configurations. In *ACSAC*, 2024.

[62] O. Starov and N. Nikiforakis. XHOUND: Quantifying the Fingerprintability of Browser Extensions. In *IEEE S&P*, 2017.

[63] O. Starov, P. Laperdrix, A. Kapravelos, and N. Nikiforakis. Unnecessarily identifiable: Quantifying the fingerprintability of browser extensions due to bloat. In *ACM WWW*, 2019.

[64] S. G. Stats. Browser Market Share Worldwide, 2024. URL https://gs.statcounter.com/browser-market-share.

[65] T. I. W. Stephenson. A Comparative Study on Analyses of Browser Fingerprinting. Bachelor thesis, Wesleyan University, 2023.

[66] P. Story, D. Smullen, Y. Yao, A. Acquisti, L. F. Cranor, N. Sadeh, and F. Schaub. Awareness, adoption, and misconceptions of web privacy tools. *PETS*, 2021.

[67] uBlock Origin. uBlock Origin, . URL https://ublockorigin.com/.

[68] uBlock Origin. uBlock Origin uAssets GitHub issues., . URL https://github.com/uBlockOrigin/uAssets.

[69] A. Vastel, P. Laperdrix, W. Rudametkin, and R. Rouvoy. Fp-stalker: Tracking browser fingerprint evolutions. In *IEEE S&P*, 2018.

[70] *CSS Containment Module Level 3*. W3C CSSWG, 2024. URL https://drafts.csswg.org/css-contain-3. Editor's Draft.

[71] WhoTracks.Me. Tracker list. URL https://whotracks.me/explorer.html.

[72] D. Zhang, J. Zhang, Y. Bu, B. Chen, C. Sun, T. Wang, and P.-M. Tardif. A survey of browser fingerprint research and application. *ACM WCMC*, 2022.

[73] S. Zhu, X. Hu, Z. Qian, Z. Shafiq, and H. Yin. Measuring and disrupting anti-adblockers using differential execution analysis. In *NDSS*, 2018.

A Extended Attacks

This section details other interesting attacks that exploit different parts of CSS features and achieve comparable results.

Lazy loading attack. Lazy loading is a browser feature that defers loading images until they are almost in view (*e.g.*, images in a blog as the user scrolls down) [3]. Each browser specifies a *loading boundary*: the maximum distance from the viewport where the image element should be to fetch the image request [3]. Our investigation also shows that, for each browser, this distance is constant and independent of the viewport size, *e.g.*, for Firefox, the distance is 585 pixels below the viewport. Relying on this functionality, the adversary can construct a group of HTML elements as follows: A "witness" element – an image, iframe, or block that will be hidden or blocked by the ad-blocker – on top of a "signal" element – an image with the malicious URL. The adversary positions the pair of elements such that the *loading boundary* goes through the "witness". Only if the rule is active, the "witness" size goes to 0, and the "signal" element crosses the *loading boundary*, triggering the malicious URL request. We can disguise "witnesses" as Ad containers or hide them with CSS, making the attack stealthy. It is efficient as it sends a request only if the rule is triggered. It can also work even if JS is disabled. However, this attack is limited to *generic* (*blocking* or *cosmetic*) rules.

Experimental details. In this attack, the adversary monitors which requests get triggered after an image crosses the `loading boundary`. In practice, however, a user scrolling through the page can trigger more images than expected. As one solution, the attacker should monitor the request timestamps to differentiate between the batch of requests initially above the boundary for blocked elements and the rest of the signal images (requested later as the user scrolls). The horizontal group signal images must always include a reference signal image ("leader") above the loading boundary, triggering first, to which the rest are compared. All requests close in time to the "leader" (modulo some threshold) make up the fingerprint signal (i.e., blocked elements). More advanced solutions could involve statistical tests to distinguish between blocked and non-blocked signal requests. For websites with scroll, multiple rows of such signal groups – with respective "leaders" – can split the immediate increase in request count over batches.

CSS `container` query attack. W3C [70] recently introduced container queries to allow developers to activate CSS rules of one element based on the "computed" values (*e.g.*, height or opacity) of other elements (containers). As the browser renders each element, it merges CSS rules from all loaded stylesheets to specify its' "computed" CSS rules; *e.g.*, in absence of a declared `width` style, an image's "computed" `width` style will be the rendered width. The CSS query `@container style(display: none) { ... }` executes styles only if the container is *rendered* hidden. The attacker can use this query to execute a `background-image` request only if a rule blocks the "container". Before `@container`, "computed" CSS values could not be used as conditions to activate other elements' styles. Container styling is still experimental. Nonetheless, we include the attack to show that new CSS features can have unforeseen privacy implications.

	Chromium	Firefox	Webkit
Market Share [64]	72.44%	2.74%	18.39%
Popularity in `ADGUARD`	80.73%	14.81%	0.94%
Popularity in `UBLOCK`	53.22%	39.02%	0.00%
Baseline attack	✓	✓	✓
Lazy loading attack	✓	✓	✓
CSS animation attack	✓	✓	
CSS `container` query attack	✓		experimental

Table A1: Susceptibility of different browser engines to the proposed attacks and the popularity of these engines in our forum datasets.

B Excluded Data Sources

Here, we present additional investigated but excluded ad-blocker forums, with the reasons for exclusion.

Adblock Plus Forum. Adblock Plus deploys a private issue reporting system. While issue submissions include detailed parameters about their browser and extension, we could not find accessible and ethical manners to collect this data.

Ghostery Forum. The Ghostery ad blocker blocks trackers based on the `WhoTracks.Me` tracker list [71] and third-party lists (EasyList [4], Peter Low's List [7], and uBlock Origin filer lists [67]). As with other ad blockers, users can activate/deactivate rules in their Ghostery extension. Though, unlike other extensions, Ghostery defines coarse categories (*e.g.*, "Advertising", "Site Analytics", *etc.*) and options to block/exclude specific `ATS`-domains rather than adding a switch for each filter list. We require a representative sample to understand if users actively edit Ghostery's unique settings, as opposed to AdGuard and uBlock Origin's filter lists. However, the extension does not forward issue posts to public issue forums but rather to the support team via e-mail.

Brave Browser Filter-lists Forum. Brave uses a browser-native ad blocker named "Shields" that relies on heuristics for advanced privacy measures (*e.g.*, CNAME uncloaking), on top of content blocking through proprietary and third-party filter lists (*e.g.*, EasyList [4]). Users can activate/disactivate filter lists in the browser settings. To post issue reports, most users use the brave community forum in the "Ad-Blocking" category [18]. These posts do not include the detailed browser filter-list configurations.

C Experimental Details

C.1 Building User Filter-List Datasets

We extract the filter list names from the configuration tables of each user post with custom parsers. However, filter lists can be referred to with multiple names depending on the post's creation method (*e.g.*, automatically by the ad-blocker, manually by the user, *etc*.). So, we manually build an alias table and normalize the names to the official names used by the Ad-blocker in question. We also record metadata such as the browser and OS if our keyword-based heuristics find any.

C.2 Estimating Filter-Rule Stability

Due to the high frequency of commits to filter-list repositories, we sample commits only at specific points in time and monitor which rules remain, marking a lower bound on the age of these rules. We face three main challenges in recovering prior filter list versions. (1) Some filter lists are compiled dynamically from many sources using a script and published on dedicated websites. (2) Some filter lists are not hosted on GitHub. Fortunately, we found GitHub mirrors for some lists in (1) and (2) that pull the latest version of the lists at constant intervals of time,*e.g.*, `thedoggybrad/easylist-mirror`, which we used as approximated history. In some cases, these approximations are not ideal, as ad blockers might lag slightly behind – by pulling older versions rather than the latest GitHub version. (3) Finally, many filter lists are not versioned at all, with only the latest version accessible through web endpoints (*e.g.*, *Peter Lowe's Blocklist* [7]). Moreover, for efficiency, we do not track whether filter lists were renamed, moved, split up, or recombined in the past; in all these cases, we consider the rule absent, in line with the lower bound on the rule lifetime. That leaves us with 75 (86%) available filter lists for `ADGUARD` and 62 (91%) for `UBLOCK`. We study the filter-list histories for 1,753 days before their download date. We start sampling day by day, then gradually increasing the intervals to months and years.

Examples of rule versioning limitations. Examples: Change in files lassekongo83/Freewits-filter-lists, uBlock-Origin/uAssetsCDN; Earliest "Easylist" mirror commit: thedoggybrad/easylist-mirror.

C.3 Domain Coverage Over Filter Rules

In addition to *generic* rules, we want to compute the added benefit of a Baseline attack adversary in triggering *domain-specific* rules by controlling the domains that trigger these rules and injecting elements or requests to be blocked. To determine the best domains to control, we implement the following steps. First, we extract all domains mentioned in *domain-specific* rules, *e.g.*, the rule `example.com##.ad1` would trigger only on `example.com`. Next, we count the number of rules each domain appears in and across filter lists. Counting requires care as (1) subdomains will trigger rules only specifying the root domain (*e.g.*, `|example.com$img` triggers for `blog.example.com`) and (2) "Any TLD" rules match all domains with a specified prefix (*e.g.*, `|example.*$img` triggers for both `example.com` and `example.org`). So, for each full domain, we add the rule count for all matching parent domains (*suffixes*) or "Any TLD" domains (*prefixes*). We adapt multiple data structures (like "tries") and optimizations to make the experiment more efficient. Next, to choose the optimal domains to control, we perform an iterative algorithm:

1. Start with the identifiable lists from the generic rules
2. *Repeat while not all lists identified:* (2.1) For each domain, find the number of unidentified lists where at least one rule exists for this domain. (2.2) Select the domain with rules from most unidentified lists. (2.3) Mark the domain and update the identified lists to include those identifiable with the domain. (2.4) If no new domains are identifiable by any domain, stop the loop.

By the end, the sequence of chosen domains minimizes the steps required to achieve maximum coverage of filter lists.

C.4 Ad-blocker Overhead Experiment

We extend the setup from Roongta and Greenstadt [50] by varying the active filter-list configurations for AdGuard and uBlock Origin between *default*, *mid* (activate the 30 most popular lists in our datasets), and *all* lists. We perform the measurements on 3K sites from the set using 20 crawlers (Docker containers) on an Intel Xeon Platinum 8160 with 192 cores. Similar to Roongta and Greenstadt [50], we use Chrome v.113. For more details about the methodology, kindly refer to their paper.

C.5 Filter-list Distance Metric

We compute this distance as the cardinality of the symmetric difference between two filter-list subscription sets $d(A,B) = |A \Delta B| = |A \cup B| - |A \cap B|$ – this distance keeps an intuition of the magnitude of altered filter lists.

D Attack Detectability Details

We describe the evasion characteristics of our proposed attacks to *SoTA* fingerprinting detection techniques proposed by the literature. We identify and discuss three detection strategies:

JavaScript-based detectors. These detectors classify whether specific JS scripts fingerprint the user or not [72]. Famous examples include *FP-Inspector* [39], *FP-RADAR* [17], and *Essential-FP* [57]. Out of our attacks, only the Baseline attack relies on a script and could be susceptible to this detection. Other attacks are scriptless and not applicable.

Table A2: In-the-wild usage of attack-relevant web features.

Feature	Nb. webpages
Total Successful Visits	22,745
CSS `container` query attack	
CSS includes `@container`	676 (2.97%)
CSS includes `@container` + `style`	6 (0.03%)
Lazy loading attack	
Image is lazy loaded	7,814 (37.33%)
CSS animation attack	
Site use CSS animations	17,116 (77.07%)
Animate background	7,289 (32.82%)
Animate background image	117 (0.53%)
Fetches Image in animation	83 (0.37%)
Baseline attack	
Site includes iframes	14,156 (62.24%)
Scripts have or call `postMesssage`	621 (2.73%)
Received `message` events	609 (2.68%)

Network-based detectors. These detectors flag the network requests emitted by browsing pages containing fingerprint data. While ad-blockers are rudimentary list-based network detectors, some studies like *FingerprintAlert* [16] introduce more advanced network monitoring. They flag a request as fingerprinting if the body or parameters includes values from 17 common fingerprinting values (*e.g.*, OS version, Resolution, *etc*.). In our attacks, individual requests contain no interpretable information. Instead, the attacker server extracts the fingerprint vector from the set of requests reaching them from the target (for blocked rules). So, detecting our attacks depends on whether the increase in request number is perceptible. According to an *HTTP Archive* [36] report over the top 1M sites within the last 3 years, the interquartile range of HTTP requests per page is between 43 and 123 requests with a cumulative size between 1,300 and 5,300 KB. Focusing on image requests, each site requests 13 to 45 requests, with a cumulative size between 300 and 2500 KB. The requests from our attacks do not contain data, so their impact on total network size is negligible. However, we can see that – depending on the mask size (Table 4) – attacks can send anywhere between 29 and 80 additional requests per page, and detectors tuned for the interquartile range would flag them. However, such detectors would still have a significant false positive rate as they overlap with the remaining 50% of web pages. The adversary can avoid such detection through two main techniques: (1) generate smaller masks with less uniqueness, or (2) use geometric constructions described in a related stylistic fingerprinting attack [44] and send only $m = k(\log(n/k)+1)$ requests for n tests, (*i.e.*, for 62 rules only 16 requests if $k = 3$).

Activity-graph detectors. These detectors use advanced techniques to model the execution of the webpage and interactions of its various components (like scripts, DOM elements, requests, *etc*.). Two prominent examples are *WebGraph* [53] and *AdGraph* [38]. Both techniques construct a graph where entities are web components, and edges between them represent a causal relationship. For example, if a script creates a new HTML image element that requests `domain.com/image.png`, the graph would include a *DOM create* edge between the script and the HTML element and a *request source* edge between the element and the network. The overall graph summarizes the activity happening on the page. These techniques are powerful at modeling logical representations of different activities like fingerprinting or adding tracking cookies, *etc*. because these representations are invariant, *e.g.*, any fingerprinting activity must trigger a network request. While *WebGraph* and *AdGraph* are more tailored for stateful tracking and JavaScript-based tracking, their graph representation can easily be appropriated to focus on relationships between DOM elements, CSS styles, and network requests invariant in our attacks.

In conclusion, no readily available technique can detect our proposed attacks in the wild with acceptable certainty. On the one hand, JS and network detectors can be circumvented through careful tuning of attack components. On the other hand, future graph-based techniques optimized to detect this type of fingerprinting require a comprehensive evaluation of CSS instrumentation, which we leave for future work.

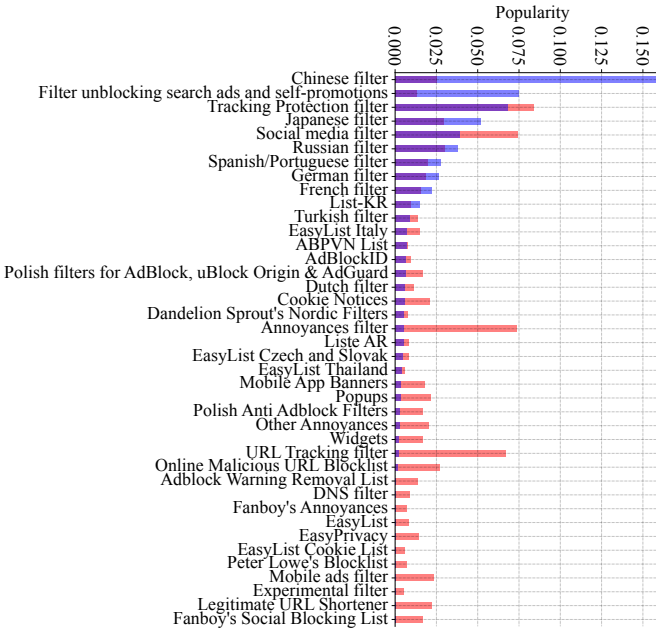

Figure A1: Filter-list popularity distribution among AdGuard users in the dataset provided by the AdGuard's team and our own `ADGUARD` forum dataset. We omit lists with a popularity of less than 0.005 in both datasets.