Evaluating Anti-Fingerprinting Privacy Enhancing Technologies

Amit Datta[*]
Snap Inc.
amit.datta@snap.com

Jianan Lu
University of California, Berkeley
amberljn@berkeley.edu

Michael Carl Tschantz
ICSI
mct@icsi.berkeley.edu

ABSTRACT

We study how to evaluate Anti-Fingerprinting Privacy Enhancing Technologies (AFPETs). Experimental methods have the advantage of control and precision, and can be applied to new AFPETs that currently lack a user base. Observational methods have the advantage of scale and drawing from the browsers currently in real-world use. We propose a novel combination of these methods, offering the best of both worlds, by applying experimentally created models of a AFPET's behavior to an observational dataset. We apply our evaluation methods to a collection of AFPETs to find the Tor Browser Bundle to be the most effective among them. We further uncover inconsistencies in some AFPETs' behaviors.

CCS CONCEPTS

• **Security and privacy → Pseudonymity, anonymity and untraceability**.

KEYWORDS

Privacy enhancing technologies; website fingerprinting; testing

ACM Reference Format:
Amit Datta, Jianan Lu, and Michael Carl Tschantz. 2019. Evaluating Anti-Fingerprinting Privacy Enhancing Technologies. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3308558.3313703

1 INTRODUCTION

Online data aggregators track consumer activities on the Internet to build behavioral profiles. In addition to using stateful methods, such as cookies, some trackers use stateless tracking mechanisms, also known as browser fingerprinting. A stateless tracker extracts fingerprints from consumers as a collection of several attributes of the browser, operating system, and hardware, typically accessed through Javascript APIs. Fingerprints collected on websites like panopticlick.eff.org and amiunique.org/fp demonstrate that they are sufficiently unique and stable for tracking purposes [15, 36]. The list of attributes that can be used in fingerprints is rapidly increasing [3, 4, 13, 18, 21, 41, 55]. Studies have also uncovered fingerprinting code on popular webpages [3, 4, 18].

Anti-Fingerprinting Privacy Enhancing Technologies (AFPETs) aim to protect consumers against fingerprinting by masking, or spoofing, the values of attributes. Our goal is to find attributes that AFPETs are not masking and to quantify their effects on privacy. We develop a method that compares the trackability of AFPET-modified fingerprints with those of the original fingerprints.

Depending on the goals, AFPET evaluation could depend on the context in which the AFPET is used, accounting for features of other users and non-users, or be a more theoretical assessment of the AFPET's potential, untied to the vagaries of today. For example, if the goal of evaluation is to determine which AFPET to use today, one would want to know how many other users of the AFPET there are since they will form the anonymity set – the group of other users one will blend in with. If instead the goal is to determine which AFPET to fund for further development, the user numbers of today may matter less than the technical or theoretical capabilities of the AFPET. Given that no one AFPET evaluation can match all goals, we will explore points in the space of possible evaluations while focusing more on prospective evaluations.

Methods. First, we consider a more theoretical, experimental analysis that directly looks at an AFPET's ability to mask attributes. This method runs browsers with and without an AFPET installed to determine which attributes the AFPET masks. For this purpose, we develop an experimental framework, PETInspector, which has three components: the *fingerprinting server* (FPServer), which collects fingerprints from visitors, the *client simulator* (ClientSim), which simulates consumers and drives them to FPServer with and without AFPETs, and the *analysis engine* (AnaEng), which compares fingerprints across clients to produce a *mask model* characterizing AFPET behaviors. This tool can be applied to new AFPETs that currently lack a user base. This experimental method does not require access to the source code of AFPETs. However, it does not tell us which attributes are the most important to mask.

Next, we consider a highly context-dependent, observational method. Websites like panopticlick.eff.org and amiunique.org/fp obtain large sets of real-world fingerprints, revealing which are the most trackable (i.e., unique and predictable). In principle, these datasets can be studied to evaluate an AFPET by selecting the fingerprints generated by users of that AFPET and, for each such fingerprint, checking how trackable they are compared to other fingerprints in the dataset. We have implemented the core task of measuring trackablity as a tool, FPInspector, which simulates a simple tracker and computes statistics quantifying anonymity, such as entropy. In practice, however, such observational datasets may contain too few users of an AFPET, especially for new ones, for FPInspector evaluate it. Furthermore, in some cases, it may be difficult to determine which fingerprints correspond to which AFPETs. Thus, utilizing such a dataset requires a more nuanced approach.

Then, we develop a hybrid method combining observational and experimental data to enable the evaluation of AFPETs with low or no usage within the context of browsers used today but without

[*]The majority of this author's contributions were made at Carnegie Mellon University.

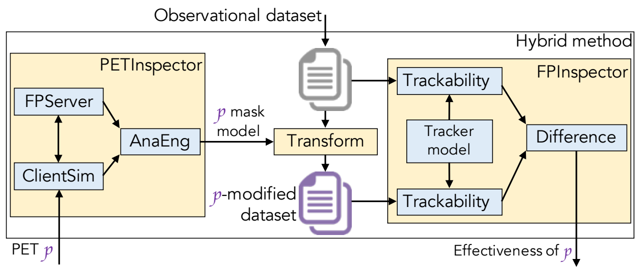

Figure 1: The hybrid AFPET evaluation method takes an observational dataset and a PET, and outputs its effectiveness.

access to the AFPET source code. Our hybrid method combines FPInspector with PETInspector as outlined in Figure 1. It contextualizes the mask model produced by PETInspector by applying it to an observational dataset of real-world fingerprints to produce a counterfactual dataset representing what the browsers would look like to trackers had everyone used the AFPET. By comparing the trackabilities on the two datasets, we evaluate the effectiveness of the AFPET. By parametrically leveraging data from ongoing, large-scale measurement studies, our methods may be adapted for the ever-changing landscape of browsers with little additional work.

Finally, we adjust the hybrid method to take into account the number of users an AFPET has. This shifts the analysis even further in the direction of examining the PET's current abilities.

Results. Using PETInspector, we resolved with high confidence whether 15 AFPETs explicitly claiming to protect against fingerprinting mask 20 attributes of Firefox and 18 attributes of Chrome. We also looked at 11 other popular *blacklisting PETs* (BLPETs), which operate by blacklisting domains known to engage in tracking. While they do not make a claim of protecting against fingerprinting, they should not make matters worse by giving browsers a more unique fingerprint, a property we check them for.

We found that all but the Tor Browser Bundle masked 9 or fewer of the resolved attributes, at least in their default configurations. In particular, we found that Tor left a single attribute, `platform`, unmasked while all others left at least 12 attributes unmasked. PETInspector also uncovered undocumented behaviors and inconsistencies in how some PETs modify various attributes:

- Brave Browser spoofs the `User-Agent` to appear like Chrome, but modifies the `Accept-Language` header, `language` and `plugins` differently than baseline Chrome. This can make Brave users stand out from other Chrome users. We have raised the issue with Brave developers and have received comments from them acknowledging the issue [9, 10].
- While both Privacy Badger and Firefox send the `Dnt` header, only Firefox sets the `doNotTrack` variable in JavaScript's navigator object. As a result, web-services which only use JavaScript to detect the Do Not Track choice will not be able to do so for Privacy Badger users. Furthermore, this inconsistency may make Privacy Badger's users stand out, making them easier to track. We raised this issue with Privacy Badger developers who have since fixed the issue [11].

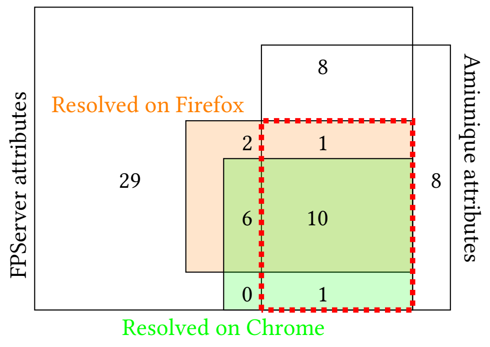

Figure 2: FPServer collects 49 attributes, of which 29 remain unexercised by ClientSim. Amiunique dataset has 28 unique attributes, of which 8 aren't collected by FPServer and 8 are collected but remain unexercised by ClientSim. The red dotted rectangle represents the intersection of attributes exercised by ClientSim and those present in the amiunique dataset, which are used for our hybrid evaluation.

- HideMyFootprint randomizes the `User-Agent` header, while not modifying the `platform`. This leads to inconsistencies like the `User-Agent` containing *Windows NT 10.0* on a *Linux x86_64* `platform`. Moreover, it sends an additional `Pragma` header, which can make users distinguishable.
- While the Tor Browser Bundle hides the OS by spoofing attributes like `User-Agent` and `cpu class`, one may still infer that information from the `javascript fonts` revealed.

We found 6 AFPETs which masked 4 attributes, but they did not all mask the same set of attributes. To break such ties, we used the hybrid method. We used a pre-existing dataset of over 25,000 real-world fingerprints collected on and provided by the website amiunique.org. [1] Of the 18–20 attributes resolved for each AFPET, only 12 appear in the `amiunique.org` dataset. Figure 2 provides an overview of how we selected attributes for our hybrid evaluation. For these 12 attributes, the hybrid method generates a set of PET-modified fingerprints from the original fingerprints and measures the effectiveness of the 15 AFPETs with FPInspector.

Our hybrid method finds that even with just 12 attributes, 13 AFPETs do not provide much protection over using no PET, decreasing the entropy revealed from about 13 bits without any PET to 11 bits with the AFPET. It finds Tor Browser Bundle (Tor BB) to be most effective, revealing under 3 bits of entropy. Given that some AFPETs only claim to protect a single attribute, these findings do not necessarily show any of them to be broken, but it does provide useful information for users and privacy advocates about their limitations. Furthermore, we found that the AFPET Trace did not mask two attributes that its documentation claimed it did (Table 3).

Recognizing that automation has its limitations, we manually analyzed some of the more interesting findings. We found that some AFPETs performed better when switched out of their default configuration. While we find that some do mask attributes labeled as inconclusive by PETInspector, we did not find any falsely labeled as masked or as unmasked.

[1] Graciously provided by Pierre Laperdrix, one of the creators of amiunique.org.

A source of entropy for Tor BB fingerprints is the revealed screen resolution, which is only partly masked. Tor BB reveals partial information about the screen resolution of its users using a spoofing strategy which depends on the true resolution for usability reasons. We explore a space of alternate spoofing strategies and find some to be just as effective according to our metrics despite being more usable, by utilizing more pixels on average for browsing than Tor.

Interpretation. AFPETs do not claim to protect against all forms of fingerprinting, and, thus, our results should not necessarily be interpreted as finding flaws. Nevertheless, our tool can be useful for AFPET developers. It can test whether they masked the attributes they intended to do so and help ensure that their documentation is correct. Indeed, we found that AFPETs Trace and Tor did not mask all the attributes that their documentation claimed they did.

BLPETs do not claim to protect against fingerprinting, but even they should avoid making browsers more fingerprintable than they already are. For example, we found that Privacy Badger made fingerprinting easier by modifying an attribute in a particular and undocumented way. Despite not making any anti-fingerprinting claims, its developers updated Privacy Badger.

For consumers and their advocates, our results are useful beyond the pre-existing, and sometimes flawed, documentation. In addition to double checking documentation, such consumers may be less concerned with whether PETs meet their specifications than their overall effectiveness, which our tools measure.

Our results are best understood as providing a lower bound on how much room for improvement remains for AFPETs. Our lower bound is sound in that when PETInspector claims that an AFPET leaves an attribute unmasked, it really is not masking it, is not varying the attribute often enough to be effective, or is not masking enough values of the attribute to protect our test browser platforms. Our bound is only a lower bound since, by resolving only the status of 18–20 attributes of each browsing platform, we might label some attributes in need of masking as inconclusive. More attributes can be added to our tools, but the set of possible attributes is open ended and finding platforms that differ in all attributes can be difficult.

Contributions. We make the following main contributions:

- We develop PETInspector to find how 15 AFPETs (and 11 BLPETs) mask 18–20 different attributes. By obtaining a more complete picture of PETs' behaviors, we uncover some inconsistencies and peculiarities (Section 4).
- We develop a hybrid method for evaluating AFPETs from an observational dataset of real-world fingerprints and apply it to evaluate 15 AFPETs. We find Tor BB to be the most effective AFPET among the ones we evaluate (Section 5).
- We adjust the hybrid method to also consider the current number of users each AFPET has (Section 6).
- We explore a space of alternate spoofing strategies for screen resolution by Tor BB and find some which have higher screen utilization than Tor BB, but are just as effective (Section 7).

2 PRIOR WORK

Prior work finds that various attributes are trackable by measuring the uniqueness and predictability of fingerprints collected from real-world browsing platforms [15, 36, 63]. However, few studies evaluate the effectiveness of AFPETs against fingerprinting.

Many prior studies have focused on BLPETs, which use blacklists to block known tracking domains and scripts. Since BLPETs try to prevent the consumer's browser from interacting with trackers, metrics suggestive of successful interactions (e.g., third-party requests sent, cookies placed, etc.) are good indicators of BLPET effectiveness. Studies have evaluated BLPETs by comparing these metrics between browsers with and without the BLPET when visiting popular websites [18, 26, 28, 31, 32, 38, 40, 50]. FPGuard takes a blacklisting strategy to protect against fingerprinting: it uses heuristics to identify fingerprinting domains and blocks them [20]. Gulyás et al. study the tradeoff between a BLPET suppressing some trackers but also leading to the browser having a more unique fingerprint by being a rare browser extension [25].

Most AFPETs protect against fingerprinting by spoofing browser, operating system and hardware characteristics, without blocking specific domains and scripts. AFPETs like the Tor Browser standardize various attribute values [48], whereas others like PriVaricator [45], FP-Block [59], Blink [35], and FPRandom [34] vary them. Metrics for evaluating BLPETs are not able to meaningfully evaluate AFPETs. Some studies have evaluated attribute-varying AFPETs by observing variations in fingerprints when using these AFPETs (e.g., [34, 45]). Vastel et al. look at how AFPETs can introduce inconsistencies between attributes leading to a more unique fingerprint [61]. Our work differs from these and uses a combination of experimental and observational data to more thoroughly evaluate AFPETs.

3 TRACKERS AND PETS

When a user visits a webpage, trackers can have the user's browser execute code that requests information about the user's browsing platform, including their hardware, operating system, and the browser itself. Table 3 provides a list of 49 attributes known to be good candidates for fingerprinting. The tracker can combine multiple attributes $a_1, \dots, a_n$ to compute a *fingerprint* $id(b) = \langle a_1(b), \dots, a_n(b) \rangle$ of the browsing platform b where $a_i(b)$ represents the value of attribute a_i for the platform b. A tracker can use fingerprints to identify browsing platforms visiting two websites as being the same one. The more unique the fingerprint is for each user, the fewer false matches the tracker will produce in linking two different users. The more predictable, or stable (ideally, unchanging), the fingerprint is as a user goes from website to website, the fewer matches the tracker will miss.

To protect themselves from fingerprinting, consumers can install AFPETs on their browsing platform to reduce the uniqueness or stability of the platform's fingerprints. Upon installing a PET p, the consumer's browsing platform b is modified to $p(b)$. As a result, the tracker interacts with $p(b)$ and extracts fingerprint $id(p(b))$.

In this study, we look at three types of PETs:

I **Attribute standardizing (AS).** These AFPETs reveal one (full standardization) or one of a small set of possible values (partial standardization) for an attribute. Full standardization makes all AFPET users appear identical, whereas partial standardization makes them appear from a few groups, with respect to that attribute. An AFPET may choose partial over full standardization if spoofing the attribute value has usability implications.

II **Attribute varying (AV).** These AFPETs vary the value of an attribute so that the values of each user varies across browsing activities. Such variations may affect both the predictability and the uniqueness of the revealed attribute. Laperdrix et al. [34] show that variation AFPETs can vary attributes in a manner that reduces their usability impact.

III **Interaction blocking (IB).** These BLPETs block some or all interactions between the browsing platform and trackers. They rely on a blacklist (e.g., EasyPrivacy) to block interactions matching known tracking patterns. Trackers interacting with browsing platforms with these PETs receive an error message instead of the true fingerprints.

We are primarily interested in evaluating AFPETs that modify the attribute values either by standardizing (I) or varying (II) their values. In some places, we comment on BLPETs that block interactions with known trackers (III). We do so even for BLPETs not claiming to be AFPETs since they are popular, have been the subject of past evaluation studies, have the potential to unintentionally make fingerprints more unique (as we find with Privacy Badger), and can be used as AFPETs. However, we do not directly compare them to the AFPETs since they do not purport to modify any attributes explicitly, and their quality depends upon the quality of their blacklists, necessitating a different form of evaluation.

We leave out of scope PETs that protect against fingerprinting by blocking scripts (e.g., NoScript [29] and ScriptSafe [6]) since they have considerable impact on usability [28]. We also leave out PETs like AdNauseum (adnauseam.io) that do not attempt to prevent tracking but rather to make it pointless by injecting noise into the user's history with fake clicks and website visits.

In this paper, we consider a total of 26 PETs. We assign each PET a unique abbreviation, which we use in some tables. We present the full list of PETs and their abbreviations in Table 1. 23 of the 26 PETs are extensions for Chrome and Firefox, two are full browsers, and one is a browser configuration. 15 of the 26 PETs are AFPETs and purport to either standardize or vary attribute values, while 11 others are popular BLPETs. Some PETs assume mixed strategies.

We went over the documentation of the PETs to uncover how they purport to modify attributes. For all PETs that explicitly document masking an attribute, we place a ☐ in the corresponding cell in Table 3. Next, we demonstrate how we use our experimental method can check whether the documentation is accurate.

Table 1: List of PETs we study, their abbreviation, and strategy to protection. Most PETs are browser extensions, * indicates full browsers, and ** indicates browser configurations. For AFPETs, we list its number of users.

PET	Abbr.	Strategy	AFPET	Users
Chrome PETs				
CanvasFingerprintBlock [8]	CFB	AS	✓	7.6K
Privacy Extension [54]	PE	AS	✓	915
Brave [12]	BR*	AS+IB	✓	N/A
Canvas Defender [43]	CD_C	AV	✓	20K
Glove [44]	GL	AV	✓	342
HideMyFootprint [1]	HMF	AV+IB	✓	177
Trace [2]	TR	AV+IB	✓	N/A
Adblock Plus [19]	AP_C	IB		
Disconnect [14]	D_C	IB		
Ghostery [23]	GH_C	IB		
Privacy Badger [17]	PB_C	IB		
uBlock Origin [27]	UO_C	IB		
Firefox PETs				
Blend In [49]	BI	AS	✓	858
Blender [39]	BL	AS	✓	1.8K
No Enum. Extensions [51]	NE	AS	✓	N/A
Stop Fingerprinting [46]	SF	AS	✓	1.8K
Tor Browser Bundle [48]	Tor*	AS	✓	≈4M
TotalSpoof [22]	TO	AS	✓	265
Canvas Defender [43]	CD_F	AV	✓	5.3K
CanvasBlocker [30]	CB	AV	✓	27K
Adblock Plus [19]	AP_F	IB		
Disconnect [14]	D_F	IB		
Ghostery [23]	GH_F	IB		
Privacy Badger [17]	PB_F	IB		
Tracking Protection [57]	TP**	IB		
uBlock Origin [27]	UO_F	IB		

4 EXPERIMENTAL EVALUATION OF AFPETS

We now consider an experimental, or test-based, approach to AFPET evaluation conducted with artificial users. These artificial users browse on platforms differing in whether they have an AFPET installed. By comparing fingerprints generated by these artificial users, we infer which attributes the AFPET is masking. We use the degree of masking by each AFPET as an evaluation metric.

4.1 Method

Our experimental framework, PETInspector, is composed of three parts. The *client simulator*, ClientSim, creates and drives experimental browsing platforms, with and without various AFPETs installed, to visit a server. The *fingerprinting server*, FPServer, plays the role of an online tracker and collects fingerprints when the browsing platforms, driven by ClientSim, visit it. The *analysis engine*, AnaEng, compares fingerprints across clients to detect whether an AFPET varies, standardizes, or does not mask the value of an attribute. To observe these behaviors, AnaEng compares the value of the attribute on the browsing platform without any AFPET (i.e., on the baseline browser) with the value when an AFPET is installed.

Client Simulator. ClientSim drives simulated clients using browsing platforms with different configurations to visit FPServer. For each base configuration and AFPET, ClientSim simulates a pair of clients only differing on whether the AFPET is installed, to allow the isolation of the AFPET's effects.

We choose the base configurations to exercise a wide range of attribute values in hopes of triggering an AFPET's masking behavior even when the masking is partial. To exercise more platforms than we have access to, ClientSim simulates browsing platforms either locally on a computer or on pre-configured VirtualBox virtual machines [62]. ClientSim configures different fonts, timezones, languages, and screen properties using OS and browser functionalities. Some attributes directly depend upon hardware (e.g., `max touch points`) or fixed OS libraries (e.g., `math` attributes) preventing their simulation. Avoid altering others due them possibly interfering the operation of PETs (`DNT enabled`, `openDB`, `indexedDB`, two `storage` attributes, and six `header` attributes), being antiqued (`plugins`), or being a function of lower-level attributes (such as `adBlock installed` and `has lied with`).

After setting up a simulated browsing platform, ClientSim drives browser instances on them using the Selenium Webdriver [52] to FPServer. The browser instances interact with FPServer in a specified pattern of reloads and idling to provide insights about the modification behavior of PETs. In hopes of triggering a PET's ability to mask by varying attribute values, ClientSim drives its browsers across various boundaries that may cause the PET to refresh its spoofed value: *reloads* of a single domain, visits to different *domains* (we give FPServer two domain names), and browsing across *sessions*. We define a session to browsing separated by 45 minutes of down time, following Mozilla's definition of a session as a continuous period of user activity in the browser, where successive events are separated by no more than 30 minutes [60].

Fingerprinting Server. FPServer collects attributes collected by the open-source fingerprinting projects FPCentral [33] and Panopticlick [16], often by reusing their code. We list these attributes in the first column of Table 3. Similar to websites like panopticlick.eff.org and amiunique.org/fp, any browser visiting FPServer's domain can view their fingerprint, while FPServer retains a copy.

Analysis Engine. To check for masking by a PET, AnaEng uses both the fingerprints collected by FPServer from the browsers driven by ClientSim and information directly from ClientSim stating which browser used which PETs and in which configurations. Figure 3 provides an overview of AnaEng. In short, the analysis looks for both masking by standardization and by variation. If it detects standardization or variation for an attribute, it models the attribute as masked in the mask model of the PET that it produces. It models an attribute as unmasked if it is able to thoroughly test it and find neither type of masking. The possible results of the analysis are

(1) Inconclusive: cannot test variation due the baseline browser varying the attribute
(2) Masked: detect AFPET-induced variation by seeing variation with the AFPET but not without the AFPET
(3) Masked: detect AFPET-induced standardization by seeing value change in a stable manner from baseline browser to browser with AFPET
(4) Inconclusive: cannot rule out partial standardization due to lack of browsing platforms that differ enough in the attribute
(5) Unmasked: rule out impactful standardization as unlikely

By *impactful* partial standardization, we mean standardization that affects at least a fraction f of the values. In general, ruling out partial standardization with experiments requires testing for all possible attribute values, a prohibitively expensive, if not impossible, task for many attributes. However, AnaEng can, in reasonable time and with reasonable confidence, rule out impactful partial standardization. To do so, AnaEng estimates the probability of seeing at least

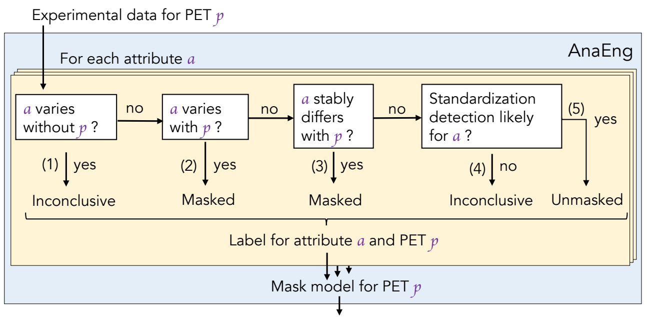

Figure 3: The Analysis Engine (AnaEng) of PETInspector consumes experimental data for a PET and outputs the corresponding mask model.

one changed value given that at least a fraction of them f are being standardized. If this probability is below some threshold α, AnaEng rejects the idea that tool is impactfully standardizing and labels the attribute as unmasked with confidence α. Otherwise, the result is inconclusive since not enough values of the attribute were tested. We use the geometric distribution to estimate likelihood of finding masking given that a fraction f is happening.

4.2 Experiment

Using PETInspector, we performed an initial experiment finding no additional spoofing from AFPETs crossing sessions. Thus, to save time, our main experiment uses only a single session and does not check for the masking of attributes by variation across sessions.

We use ClientSim to simulate six browsing platforms. Three of these are virtual machines running various versions of Linux. We introduce additional changes into these virtual machines to simulate differences in system configurations. Specifically, we install different fonts and browser versions, set up different timezones, and simulate different screen resolutions and languages, The remaining platforms run natively on a Linux desktop, Macbook Pro, and a PC laptop. We perform measurements on Firefox and Chrome browsers. More details on these configurations are in Table 2.

ClientSim drives these experimental browsing platforms to reach FPServer for five reloads of each of the two domain names of FPServer. On each platform, it performs these reloads a total of 28 times: one time each for 26 PETs and one time each for the two baseline browsers. All PETs are left in their default configurations.

4.3 Results

Before commenting on PETs, we make some observations about the baseline browsers. While we did not think of the choice of browsers as affecting the trackability of fingerprints, it turns out that the two browsers have small differences in the attributes shared by them. Aside from the expected difference in the browser name in `User-Agent`, Chrome sets the `cpu class` to *unknown*, the `screen.Depth` to 24, and the `buildID` to *Undefined*, unlike Firefox which reveals different values across browsing platforms. Firefox does not reveal any plugins, while Chrome does, and Chrome's plugins differ across Ubuntu, Debian, and macOS. PETInspector does

Table 2: Configurations of simulated browsing platforms in our main experiment. The last three were regularly used.

#	Type	OS	Addl. Fonts	Resolution	Locale & LANG	Timezone	Browser versions		Notes
							Firefox	Chrome	
1	VM	Ubuntu 16.04	Mordred	450×721×24	ru_RU.UTF-8	GMT+6	56.0	63.0	
2	VM	Debian 8.10	OldLondon	2000×2000×16	de_DE.UTF-8	GMT-3	56.0	63.0	
3	VM	Ubuntu 14.04	(none added)	6000×3000×24+64	ar_SA.UTF-8	GMT-11	56.0	63.0	
4	Local	Ubuntu 16.04	> 40	1920×1080×24	en_EN.UTF-8	GMT-8	56.0	70.0	
5	Local	macOS 10.13	> 145	1440×900×24	en_EN.UTF-8	GMT-8	56.0	70.0	
6	Local	Windows NT 10.0	> 145	1280×720×24	en_EN.UTF-8	GMT-8	56.0 beta	69.0	Touch screen

not find any baseline browser to vary any attributes itself (outcome (1) in Fig. 3).

Turning to PETs, PETInspector produces Table 3, which displays attributes masked or not by AFPETs. We comment on the BLPETs in text. Among the 15 AFPETs, three (Trace, Privacy Extension, and No Enum. Extensions) do not lead to any detectable masking in their default configurations. The remaining 12 AFPETs mask at least one of the collected attributes.

Our experiment also detects undocumented masking of attributes. For example, while Canvas Defender$_C$, Canvas Defender$_F$, CanvasFingerprintBlock, Glove, and CanvasBlocker claim to spoof only the `canvas fingerprint`, we also find them spoofing `webGL` attributes. Similarly, we find undocumented modifications by Brave, Stop Fingerprinting, and TotalSpoof. We also find inconsistencies in the behavior of Brave, Privacy Badger$_C$, Privacy Badger$_F$, HideMyFootprint, and Tor BB, which we discussed in Section 1.

Among the 11 BLPETs, 4 (Disconnect$_F$, Disconnect$_C$, Ghostery$_C$, and Ghostery$_F$) do not lead to any detectable modifications of attributes, 4 (Adblock Plus$_C$, Adblock Plus$_F$, uBlock Origin$_C$, and uBlock Origin$_F$) modify the attribute `adBlock installed`, and 3 (Privacy Badger$_C$, Privacy Badger$_F$, and Tracking Protection) modify Do Not Track attributes. As discussed in the introduction, these BLPETs are not presented as AFPETs, but their modifications can actually make their users more identifiable. Indeed, Privacy Badger was updated in response to our finding.

4.4 Discussion and Limitations

The ranking above may not be suitable for some evaluation goals. For example, some AFPETs were designed to mask a single attribute and does in fact mask it (e.g., Canvas Defender$_C$). Our findings that such AFPETs (or BLPETs) do not mask all attributes should not be interpreted as the PET having a bug. Nevertheless, consumers and advocates seeking effective PETs may find our results useful.

As mentioned above, we may miss some masking of attributes for not testing values that an AFPET standardizes away. Furthermore, we may not detect an AFPET varying an attribute across a boundary that we do not test. Thus, while we can be sure of masking when we find it, we cannot be sure we have found all masking.

Since FPServer extracts fingerprints using first-party scripts, we do not detect masking that is triggered only for third-party scripts.

To an extent, these limitations can be mitigated with more comprehensive experiments using PETInspector. For example, one can modify FPServer to collect additional attributes in both first-party and third-party contexts. Moreover, one can modify ClientSim to detect variations across other boundaries and use more diverse browsing platforms to be more confident about not missing standardization modifications. We will make PETInspector freely available for more extensive experimentation and further development. Our current evaluations demonstrate the benefits of an experimental evaluation method for AFPETs within the current boundaries.

Our experiments may dispute claimed masking (☒ in Table 3) due to the above limitations rather than documentation making spurious claims. AFPETs may mask more attributes when appropriately configured, but users find it difficult to change defaults [37], suggesting our experiments may capture typical use. Next, we perform a manual analysis to understand the effects of configuration and why our results conflict with some AFPETs' documentation.

4.5 Additional Manual Analysis

To address some of the limitations mentioned above, we manually analyze some AFPETs. Specifically, we analyze AFPETs for which we found no evidence of any masking (Trace, Privacy Extension, No Enum. Extensions) and those which made a claim rejected by PETInspector (Trace, Privacy Extension, Stop Fingerprinting, Tor).

PETInspector rejects two claims of masking by Trace. We could not find the source code for Trace, but we installed the extension and manually examined it. Both the documentation and settings panel show canvas fingerprinting being masked by default, despite our studies concluding the opposite. As far as we can tell, Trace really does not mask this attribute despite claiming to. Since running our tests, Trace has been updated from version 1.0.2 to 1.8.6 and it now randomizes the canvas fingerprinting.

As for masking the user-agent, the settings panel of Trace shows that user-agent randomization is off by default, explaining our finding. Turning it on does randomize the user agent.

All of Privacy Extension's masking abilities are off by default. Turning them on does result in standardizing the two attributes in question: the canvas fingerprint and the user-agent.

To analyze No Enum. Extensions, we examined both its source code and documentation. The documentation of No Enum. Extensions only claims to mask plugins, and we found evidence of plugin masking in No Enum. Extensions's source code. PETInspector was inconclusive for this attribute since it was unable to exercise the plugin list for Firefox due to Firefox making the loading of any plugins a manual process. Thus, this instance does not represent a false negative, and instead represents a failure to find a positive.

Table 3: AFPET masks as purported and observed by PETInspector. □ indicates AFPET's documentation purports that the attribute is masked. The remaining symbols represent the possible outputs of PETInspector: + indicates observed masking, × indicates no masking found even when it is likely to detect it, and · indicates inconclusive results. For the results that we manually double checked, we include the outcome of that check as a superscript. Here, × denotes that the PET really does not mask the attribute, + that it does, ×/+ that it is not masked by default but can be with configuration, and ? that the manual analysis was inconclusive. Not shown are attributes that had all inconclusive results and not purported masking (nothing but ·): `DNT enabled`, `IE addBehavior`, `adBlock installed`, `h.Connection`, `h.Dnt`, `h.Up.-Ins.-Req.`, `indexedDB`, `math.acosh(1e300)`, `math.asinh(1)`, `math.atanh(05)`, `math.cbrt(100)`, `math.cosh(10)`, `math.expm1(1)`, `math.log1p(10)`, `math.sinh(1)`, `math.tanh(1)`, **and** `openDB`**.**

	Chrome							Firefox							
Attribute	BR	CD$_C$	CFB	GL	HMF	PE	TR	BI	BL	CD$_F$	CB	NE	SF	Tor	TO
buildID	·	·	·	·	·	·	·	⊞	⊞	×	×	×	×	⊞	+
canvas fingerprint	⊞	⊞	⊞	⊞	⊞	⊠$^{\times/+}$	⊠$^{\times}$	×	×	⊞	⊞	×	×	⊞	×
cookies enabled	·	·	·	·	·	⊡	·	·	·	·	·	·	·	·	·
cpu class	·	·	·	·	·	·	·	⊞	⊞	×	×	×	+	⊞	+
h.Accept	·	·	·	·	·	⊡	·	·	·	·	·	·	·	⊡	·
h.Accept-Encoding	·	·	·	·	·	·	·	·	·	·	·	·	·	⊡	·
h.Accept-Language	+	×	×	×	×	×	×	×	⊞	×	×	×	×	⊞	×
h.Pragma	·	·	·	·	+	·	·	·	·	·	·	·	·	·	·
h.User-Agent	⊞	×	×	×	⊞	⊠$^{\times/+}$	⊠$^{\times}$	⊞	⊞	×	×	×	×	⊞	⊞
javascript fonts	×	×	×	×	×	×	×	×	×	×	×	×	⊠$^{?}$	⊞	×
language	+	×	×	×	×	×	×	×	⊞	×	×	×	×	⊞	×
local storage	·	·	·	·	·	⊡	·	·	·	·	·	·	·	·	·
platform	×	×	×	×	×	×	×	⊞	⊞	×	×	×	×	⊠$^{\times}$	+
plugins	⊞	×	×	×	×	×	×	·	·	·	·	⊡$^{+}$	⊡$^{+}$	⊡	·
screen.AvailHeight	×	×	×	×	×	×	×	×	×	×	×	×	⊞	⊞	×
screen.AvailLeft	×	×	×	×	×	×	×	×	×	×	×	×	⊞	⊞	×
screen.AvailTop	×	×	×	×	×	×	×	×	×	×	×	×	⊞	⊞	×
screen.AvailWidth	×	×	×	×	×	×	×	×	×	×	×	×	⊞	⊞	×
screen.Depth	·	·	·	·	·	·	·	×	×	×	×	×	⊞	⊞	×
screen.Height	×	×	×	×	×	×	×	×	×	×	×	×	⊞	⊞	×
screen.Left	·	·	·	·	·	·	·	·	·	·	·	·	⊡	⊡	·
screen.Pixel Ratio	×	×	×	×	×	×	×	×	×	×	×	×	⊞	⊞	×
screen.Top	·	·	·	·	·	·	·	·	·	·	·	·	⊡	⊡	·
screen.Width	×	×	×	×	×	×	×	×	×	×	×	×	⊞	⊞	×
session storage	·	·	·	·	·	⊡	·	·	·	·	·	·	·	·	·
timezone	×	×	×	×	×	×	×	×	×	×	×	×	×	⊞	×
touch.event	·	·	·	·	·	·	·	×	×	×	×	×	×	⊞	×
touch.max points	×	×	×	×	×	×	×	·	·	·	·	·	·	·	·
touch.start	·	·	·	·	·	·	·	×	×	×	×	×	×	+	×
webGL.Data Hash	⊞	+	+	+	+	×	×	×	×	+	+	×	×	⊞	×
webGL.Renderer	⊞	+	+	+	+	×	×	×	×	+	×	×	×	⊞	×
webGL.Vendor	⊞	+	+	+	+	×	×	×	×	+	×	×	×	⊞	×

We manually tested Stop Fingerprinting and found that, like No Enum. Extensions, it masks plugins despite PETInspector's inconclusive finding. As for the rejected claim of masking javascript fonts, Stop Fingerprinting may be doing something with the fonts, but not enough to defeat the way FPServer fingerprints them.

Examining Tor BB leads us to believe that a recent update (after Version 7.0.11) accidentally affected its masking of the `platform` attribute. We also find that during the same time frame, `cpuclass` and `h.User-Agent` went from being fully masked to partially masked. We found user complaints about this change in version 8.0a10 [7].

We also confirmed that Privacy Badger did not set the doNotTrack field of the navigator object to match the Dnt header. The code was fixed after we notified the developers of the issue [5].

During our manual examinations, we also looked for artifacts introduced by VMs simulating platforms and found none.

5 HYBRID EVALUATION OF AFPETS

Our experimental method provides a model of how various AFPETs mask fingerprints as well a ranking of AFPETs based on the number of attributes they mask. However, it does not consider how important masking each attribute is. We develop a hybrid method that combines the benefits of the experimental method with an observational method. We start by considering a completely observational method and then discuss how combining it with our experimental method allows us to overcome each of their limitations.

5.1 Sampling

We cannot, in practice, see all the world's browsing platforms and instead must work with a sample. The quality of the metrics computed from the sample depends upon both the nature of the metric and the sample. For example, a random sample will provide a reasonable estimation of the entropy (e.g., [47]). However, estimating the proportion of users in small anonymity sets from even a random sample proves difficult since the length of the tail of the distribution may be unclear from a random sample.

Furthermore, in practice, we must approximate truly random samples of browser platforms from available datasets since we cannot force all users to participate. We do so by using a convenience sample provided to us by the `amiunique` website This sample comprises 25,984 real-world fingerprints collected over a period of 30 days (10/02/2017 to 11/02/2017). Each fingerprint has 32 attributes.

Determining the representativeness of this sample is difficult since it can only be compared to other possibly unrepresentative samples. We compare our sample's distributions to GlobalStat's for desktop users [56]. We find that our sample has a higher proportion of Firefox users (42% vs. 12%) and of Linux users (19% vs. 1%).

5.2 Metrics of Trackability

To measure trackability of fingerprints, we build FPInspector, which consumes a dataset and characterizes how trackable its members are. One such characterization is the anonymity set. An *anonymity set* comprises browsing platforms with identical fingerprints that are, thus, indistinguishable from each other. Thus, the smaller and numerous the anonymity sets, the higher the uniqueness. FPInspector implements various proposed functions (in [15, 63]) over the distribution of anonymity sets for measuring uniqueness.

The first metric which we use to measure uniqueness is entropy. For a set of browser platforms $\mathcal{D} = \{b_i\}_i$, such as those using a particular AFPET, let $\mathcal{D}[id(\cdot)]$ denote the multiset of fingerprints $\{id(b_i)\}_i$ where $id(\cdot)$ is the fingerprinting mechanism. The entropy of these fingerprints is given by

$$\mathsf{ent}(\mathcal{D}[id(\cdot)]) = -\sum_{id_k \in \mathcal{D}[id(\cdot)]} \Pr[id_k] \log_2(\Pr[id_k])$$

where $\Pr[id_k]$ is the probability of observing the fingerprint id_k, which we estimate from the frequency of id_k in $\mathcal{D}[id(\cdot)]$. The higher the entropy, the higher the uniqueness of the fingerprints.

FPInspector also measures the proportion of users in anonymity sets of size less than or equal to 1 ($\mathsf{p}_{-\leq 1}$) and 10 ($\mathsf{p}_{-\leq 10}$). These metrics measure the proportion of browsing platforms hiding in anonymity sets of sizes at most 1 and 10. Higher values of these metrics indicate higher uniqueness of the fingerprints.

FPInspector measures effectiveness of a PET p against fingerprinting mechanism $id(\cdot)$ from the dataset of fingerprints $\mathcal{D}[id(\cdot)]$ in terms of a metric f in $\{\mathsf{ent}, \mathsf{p}_{-\leq 1}, \mathsf{p}_{-\leq 10}\}$ as

$$\mathsf{eff}_{\mathsf{f}}(p, id, \mathcal{D}_p, \mathcal{D}_{\bar{p}}) := \mathsf{f}(\mathcal{D}_{\bar{p}}[id(\cdot)]) - \mathsf{f}(\mathcal{D}_p[id(\cdot)]) \qquad (1)$$

where $\mathcal{D}_p$ is a subset of $\mathcal{D}$ using the PET and $\mathcal{D}_{\bar{p}}$ is the rest of $\mathcal{D}$.

5.3 Limitations of Observations Alone

In principle, a highly-context dependent, completely observational method could function by comparing the fingerprints produced by users of each AFPET to determine which are the least trackable. In practice, we face difficulties with obtaining a representative sample of AFPET users and determining which users run which AFPETs.

PET determination. Determining PET use from fingerprints not explicitly containing the information is difficult. This limitation can be overcome by a fingerprinting server designed to collect information about PET use. One approach is to ask visitors about their PETs, but users can be unaware of their own browser's configurations. In some cases, PETs have a distinctive fingerprint that gives away their use, but this would only help us with a subset of PETs. Alternatively, fingerprint collection websites can use automated methods to detect browser extension PETs (e.g., [53, 55]). Unfortunately, our observational data lacks this information.

PET sampling. Even with a fingerprinting server collecting PET information, getting a representative sample of real users with AFPETs to visit the website may be difficult, since there are few AFPET users. This is especially true for new and not yet popular AFPETs. Furthermore, users of AFPETs may be systematically different from users without AFPETs, thereby introducing confounding factors influencing the trackability metrics.

Due to these limitations, we cannot apply FPInspector directly to our dataset. Moreover, the PET sampling limitation may prevent application of this method directly to data collected on even fingerprinting servers designed for PET determination. Thus, we instead use FPInspector in a hybrid evaluation method that avoids the PET determination and sampling problems altogether.

5.4 Overcoming Limitations of Observations

To overcome the difficulty of getting a sample $\mathcal{D}_p$ of browser platforms using a PET p, we construct our own from a sample $\mathcal{D}_{\bar{p}}$ of browser platforms not using p. We then provide both to FPInspector, to evaluate the PET p, as show in Figure 1.

This approach requires that we first get a sample of platforms not using p. We start with the `amiunique` dataset. To convert that dataset of fingerprints into one of platforms, we need a mapping of fingerprints to unique browsing platforms. We approximate this mapping using cookie IDs associated with each fingerprint, similarly to Eckersley [15]. In the dataset, 21,395 fingerprints have a cookie associated with them, of which, 18,295 are unique.

To obtain $\mathcal{D}_{\bar{p}}[id(\cdot)]$, we sanitize the dataset by removing fingerprints with obvious signs of PET use, specifically those with JavaScript disabled and illegitimate screen resolutions. Additionally, we only retain fingerprints from desktop browsers (with Windows, Mac, or Linux OSes) since we only study PETs for desktops. These sanitizations leave 9,493 Chrome and 6,516 Firefox browser fingerprints. We find that these fingerprints reveal 13.002 and 12.359 bits

Table 4: Metrics of trackability (from Section 5.2) for AFPETs

PET	ent	$p_{=\leq 1}$	$p_{=\leq 10}$
Chrome PETs			
no mask	13.002	0.892	0.983
base mask, Privacy Extension, Trace	12.914	0.829	0.982
Canvas Defender$_C$, CFB, Glove	12.306	0.641	0.893
HideMyFootprint	11.77	0.497	0.825
Brave	8.108	0.072	0.262
Firefox PETs			
no mask	12.359	0.875	0.96
base mask, No Enum. Extensions	12.177	0.797	0.949
Blend In, TotalSpoof	12.049	0.747	0.936
CanvasBlocker	12.002	0.7	0.941
Blender	11.875	0.678	0.924
Stop Fingerprinting	11.778	0.726	0.919
Canvas Defender$_F$	11.263	0.483	0.833
Tor BB	4.766	0.01	0.038

of entropy for Chrome and Firefox respectively. These and other metrics are presented in Table 4 in the 'no mask' row.

The mask model from the experimental method provides a way to transform these original fingerprints. We apply the mask model $\hat{p}$ of an PET p produced by PETInspector to the sample $\mathcal{D}_{\bar{p}}$ of platforms without a PET to generate a sample of fingerprints $\mathcal{D}_{\hat{p}}[id(\cdot)]$. This generated sample estimates what the original fingerprints would had looked like had the platforms used the PET p. We use FPInspector to calculate the trackability metrics of the modified fingerprints and unmodified fingerprints. By comparing the metrics of the original and $\hat{p}$-modified fingerprints, we estimate the effectiveness of p.

Of the 49 original attributes, PETInspector provides conclusive characterization for 18 attributes on Chrome and 20 attributes on Firefox. Of these, 12 appear in the `amiunique.org` dataset. For a given PET, we mask these 12 attributes according to the model generated by PETInspector and fully mask the remaining 16 attributes for which the experiment is inconclusive. By fully masking inconclusive attributes, we overestimate the effectiveness of PETs. Thus, we generate PET-modified fingerprints (i.e., $\mathcal{D}_{\hat{p}}[id(\cdot)]$) from the original fingerprints to measure the effectiveness of 15 AFPETs.

5.5 Results

We present the trackability metrics from Section 5.2 in Table 4. The original fingerprints reveal 13.002 and 12.359 bits of entropy for Chrome and Firefox respectively. Applying a *base mask* of all inconclusive attributes reduces them to 12.914 and 12.177 bits.

Our evaluations reveal that all AFPETs but Brave and Tor BB reveal over 11 bits of entropy and hence are marginally better than not using any AFPET at all. For these AFPETs, fewer than 20% of the fingerprints are in anonymity sets of size greater than 10. Brave does better, leaking just over 8 bits of entropy and having over 70% of fingerprints in anonymity sets of size greater than 10. Tor BB performs best since it modifies all the 12 attributes we consider.

5.6 Remaining Limitations

While the hybrid method helps us perform a fine-grained evaluation of AFPETs with few users, it inherits some limitations of the methods on which it builds. From the observational method come the limitations that samples may be biased and that no one metric fully captures the quality of an AFPET. From the experimental method, it inherits the approximate nature of mask models.

In particular, our analysis overestimates the effectiveness of all AFPETs, since we assume any modifications of an attribute by an AFPET renders that attribute useless to a tracker. This may not be the case. For example, Brave spoofs the `User-Agent` and the `Accept-Language` headers to different values than Chrome. Similarly, Tor BB also reveals spoofed values of screen resolution.

We can carry out a tighter evaluation by considering a tracker which can take advantage of the spoofed values. This evaluation requires knowledge of how an AFPET spoofs the attribute. For Tor BB, we perform a manual code analysis to determine how exactly Tor BB deals with screen resolution attributes.[2] We rerun the hybrid analysis on a hand crafted mask model capturing this behavior instead of using the rough model produced by PETInspector. This provides a tighter evaluation for Tor BB that will serve as the basis for our analysis in Section 7.

Finally, the above evaluations are performed on the same set of fingerprints and applies the mask to every fingerprint in the dataset, simulating total adoption of the AFPET. This approach is appropriate evaluations with a long-term prospective, such as selecting an AFPET to fund, since a properly promoted AFPET could become nearly universal in the future. However, those looking to select a AFPET for usage today should be concerned with the number of users each AFPET has since it will affect the size of the anonymity set the AFPET produces. In the next section, we consider a modification of the above method for dealing with this issue.

6 ADJUSTING FOR NUMBER OF USERS

To observe the consequences of having user bases of different sizes, we also evaluate the AFPETs taking into account their popularity. Ideally, we would do this by analyzing fingerprints of all the users of an AFPET. However, not having access to this set of fingerprints, we simulate them by drawing random samples of fingerprints from the `amiunique.org` dataset of size equal to the number of AFPET users and estimate uniqueness metrics on the samples.

Table 1 displays the number of users of each AFPET in our list as of Dec. 2017. The popularity of extensions were obtained from the Firefox add-on library [42] and the Chrome extensions webstore [24]. Tor's popularity was obtained from Tor Metrics [58]. For AFPETs with an undisclosed number of users, such as Brave and Tracking Protection, we are unable to perform this evaluation.

We also do not perform these evaluations for AFPETs with a user base greater than 17,109 (like Tor BB, Canvas Defender$_C$ and CanvasBlocker), since we cannot draw a sample from our dataset of sufficient size. Attempting to draw such a sample by allowing the same fingerprint to be sampled multiple times will overestimate the effectiveness of the PET since such repeats will surely be in the same anonymity set even for PETs that do nothing.

[2] https://gitweb.torproject.org/tor-browser.git/commit/?h=tor-browser-45.8.0esr-6.5-2&id=7b3e68bd7172d4f3feac11e74c65b06729a502b2.

Table 5: Metrics of trackability (from Section 5.2) for AFPETs on samples scaled according to their popularity

PET	#users	ent	$p_{-\leq 1}$	$p_{-\leq 10}$
Chrome PETs				
HideMyFootprint	177.0	7.343	0.901	1.000
Glove	342.0	8.277	0.886	1.000
CanvasFingerprintBlock	7630.0	11.559	0.313	0.899
Firefox PETs				
TotalSpoof	265.0	7.904	0.889	1.000
Blend In	858.0	9.401	0.777	0.983
Stop Fingerprinting	1754.0	9.994	0.641	0.939
Blender	1816.0	10.200	0.614	0.960
Canvas Defender$_F$	5274.0	10.656	0.252	0.845

For all other AFPETs, we compute the mean of the trackability metrics from 100 random samples. Table 5 displays the effectiveness metrics for these AFPETs, sorted according to the entropy. We can see that CanvasFingerprintBlock scores better than HideMyFootprint due to its high popularity, contrary to the original evaluations in Table 4. We also see that the effectiveness of tools with identical effects increases with popularity. For example, TotalSpoof and Blend In both identically modify 12 attributes, but Blend In is more effective than TotalSpoof due to its popularity.

7 APPLICATION: INFORMING AFPET DESIGN

With the ability to accept handcrafted mask models, our hybrid method can help AFPET developers make an informed choice while designing AFPETs. By measuring the effectiveness of hypothetical AFPET designs, developers can compare masking strategies to balance utility with trackability. We carry out such an exploration of alternate designs of Tor BB that mask attributes differently.

Tor BB leaks some information about the screen resolution by only partially standardizing it. Specifically, it resizes new browser windows in quanta (step/bucket sizes) of 200×100 pixels, while capping the window size at 1000×1000 pixels, and uses the client content window size as screen dimensions [48]. As a result all Tor BB users get placed into one of 50 anonymity sets based on the revealed screen dimensions, as long as they do not change the window dimensions manually. We explore the impact of the cap and quanta parameters on the effectiveness of Tor BB.

We use the number of unutilized screen pixels due to a spoofing strategy as a measure of utility loss. We measure two variants: the total number of unutilized pixels (average absolute loss), and the number of unutilized pixels as a percentage of the available pixels (average percentage loss). Increasing the cap parameters and decreasing the quanta parameters reduces this loss.

With the Firefox fingerprints in the `amiunique.org` dataset, we explore parameter settings with the goal of finding a strategy that reduces the utility loss while increasing the effectiveness. We consider alternative cap widths of 1000, 1350, 1550, and 1600 since a higher percentage of fingerprints (25%, 47%, and 51% respectively) have screen widths less than these caps. We retain the cap height of 1000 pixels as more than 50% of the fingerprints remain below that cap. We exhaustively search for all 10,201 quanta in the range 200×100 to 300×200 for all three cap parameters. We set an upper bound of 300×200 as the loss may be too high for low-resolution displays for very high quanta parameters. We find 786 and 291 quanta parameters for cap widths of 1350 and 1550 respectively for which the losses are lower than Tor BB's, but the effectiveness is higher. We display strategies with the least quanta parameters in Table 6. As we increase the cap width to 1600, none of the quanta parameters lead to a higher measure of effectiveness than Tor BB.

Table 6: Comparison of effectiveness and utility-loss of Tor BB's original spoofing strategies with alternate strategies

Cap	Quanta	ent	$p_{-\leq 1}$	$p_{-\leq 10}$	Abs. Loss	% Loss
1000×1000	200×100	2.902	0.001	0.010	870k	50.3%
1350×1000	200×193	2.715	0.001	0.009	729k	42.6%
1350×1000	269×160	2.901	0.000	0.009	728k	42.3%
1550×1000	222×197	2.899	0.000	0.009	666k	40.3%
1550×1000	295×160	2.882	0.000	0.010	636k	37.2%

8 CONCLUSION

We end with some suggestions for AFPET developers and evaluators. We recommend that developers address any attribute that PETInspector flags as unmasked. The entropy results from our hybrid method can aid in determining the order in which to address various unmasked attributes. Given our experimental results, we expect this task will keep the developers of most AFPETs busy. Next, they might want to consider any attributes that PETInspector labeled as inconclusive. After addressing these attributes, they can consider improving how an AFPET spoofs an attribute. As shown in Section 7, not all spoofing is equal. Developers should consider using Tor BB as a starting point for their development and carefully consider the default settings of their AFPET.

The set of fingerprintable attributes are open-ended and will never be fully enumerated, but new attributes can be added to our tools. AFPET evaluators should keep in mind that any one-time evaluation of PETs will quickly become out of date, necessitating the use of automated tools like ours. We encourage developers and advocates (e.g., the EFF) to use automated tools to regularly test the trackability of PETs. Our tool can fill this need, and to this end we open-source the tool here:

https://github.com/tadatitam/pet-inspector

ACKNOWLEDGEMENTS

We thank Pierre Laperdrix for providing us a dataset of real-world fingerprints from amiunique.org, and Milan Ganai for investigating how to automate the use of PETs on Windows. We thank Lay Kuan Loh and Zheng Zong for assistance in exploring the application of information flow experiments to evaluate PETs. We thank Anupam Datta for discussions about this work. We gratefully acknowledge funding support from the National Science Foundation (Grants 1514509 and 1704985). The opinions in this paper are those of the authors and do not necessarily reflect the opinions of any funding sponsor or the United States Government.

REFERENCES

[1] Absolute Double. 2017. HideMyFootprint: Protect your privacy. https://hmfp.absolutedouble.co.uk. (2017). Accessed Dec. 25, 2017.

[2] Absolute Double. 2018. Trace: Browse online without leaving a Trace. https://absolutedouble.co.uk/trace/. (2018). Accessed Jan. 12, 2018.

[3] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. 2014. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 674–689.

[4] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. 2013. FPDetective: dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 1129–1140.

[5] Alexei "ghostwords". 2018. Support navigator.doNotTrack. Pull request #1861 for the EFForg/privacybadger project on GitHub: https://github.com/EFForg/privacybadger/pull/1861. (July 2018).

[6] Andrew. 2017. Scriptsafe: andryou. https://www.andryou.com/scriptsafe/. (2017). Accessed Dec. 25, 2017.

[7] Anonymous. 2018. Comment 276687 on "New Release: Tor Browser 8.0a10". Tor Blog: https://blog.torproject.org/comment/276424#comment-276424. (Aug. 2018). See responses as well.

[8] appodrome.net. 2017. CanvasFingerprintBlock: Chrome Web Store. https://chrome.google.com/webstore/detail/canvasfingerprintblock/ipmjngkmngdcdpmgmiebdmfbkcecdndc?hl=en. (2017). Accessed Dec. 25, 2017.

[9] Blinded for anonymization. 2018. A bug report to Brave. (Jan. 2018).

[10] Blinded for anonymization. 2018. Another bug report to Brave. (Jan. 2018).

[11] Blinded for anonymization. 2018. A bug report to Privacy Badger. (Jan. 2018).

[12] Brave Browser. 2017. Fingerprint Protection Mode. https://github.com/brave/browser-laptop/wiki/Fingerprinting-Protection-Mode. (2017). Accessed Dec. 19, 2017.

[13] Yinzhi Cao, Song Li, and Erik Wijmans. 2017. (Cross-)Browser Fingerprinting via OS and Hardware Level Features. In *24th Annual Network and Distributed System Security SymposiumNDSS*. http://www.yinzhicao.org/TrackingFree/crossbrowsertracking_NDSS17.pdf

[14] Disconnect. 2017. Disconnect. https://disconnect.me. (2017). Accessed Jan. 12, 2017.

[15] Peter Eckersley. 2010. How unique is your web browser?. In *Privacy Enhancing Technologies*, Vol. 6205. Springer, 1–18.

[16] Electronic Frontier Foundation. 2017. Panopticlick. https://panopticlick.eff.org. (2017). Accessed Dec 12, 2017.

[17] Electronic Frontier Foundation. 2017. Privacy Badger. https://www.eff.org/privacybadger. (2017). Accessed Jan. 13, 2017.

[18] Steven Englehardt and Arvind Narayanan. 2016. Online tracking: A 1-million-site measurement and analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1388–1401.

[19] eyeo GmbH. 2017. Adblock Plus: Surf the web without annoying ads! https://adblockplus.org. (2017). Accessed Dec. 27, 2017.

[20] Amin FaizKhademi, Mohammad Zulkernine, and Komminist Weldemariam. 2015. FPGuard: Detection and prevention of browser fingerprinting. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 293–308.

[21] David Fifield and Serge Egelman. 2015. Fingerprinting web users through font metrics. In *International Conference on Financial Cryptography and Data Security*. Springer, 107–124.

[22] fonk. 2017. TotalSpoof Add-on Homepage. http://fonk.wz.cz/totalspoof. (2017). Accessed Dec. 25, 2017.

[23] Cliqz International GmbH. 2017. Ghostery Makes the Web Cleaner, Faster and Safer! https://www.ghostery.com. (2017). Accessed Dec. 27, 2017.

[24] Google. 2017. Chrome web store. https://chrome.google.com/webstore/category/extensions. (Dec. 2017).

[25] Gábor György Gulyás, Dolière Francis Somé, Nataliia Bielova, and Claude Castelluccia. 2018. To Extend or Not to Extend: On the Uniqueness of Browser Extensions and Web Logins. In *Proceedings of the 2018 Workshop on Privacy in the Electronic Society (WPES'18)*. ACM, New York, NY, USA, 14–27. DOI: http://dx.doi.org/10.1145/3267323.3268959

[26] Raymond Hill. 2015. uBlock and others: Blocking ads, trackers, malwares. https://github.com/gorhill/uBlock/wiki/uBlock-and-others%3A-Blocking-ads%2C-trackers%2C-malwares. (May 2015). Accessed July 5, 2017.

[27] Raymond Hill. 2017. uBlock Origin: An efficient blocker for Chromium and Firefox. https://github.com/gorhill/uBlock. (2017). Accessed Dec. 27, 2017.

[28] Muhammad Ikram, Hassan Jameel Asghar, Mohamed Ali Kaafar, Anirban Mahanti, and Balachandar Krishnamurthy. 2017. Towards seamless tracking-free web: Improved detection of trackers via one-class learning. *Proceedings on Privacy Enhancing Technologies* 2017, 1 (2017), 79–99.

[29] InformAction. 2017. NoScript: JavaScript/Java/Flash blocker for a safer Firefox experience! https://noscript.net. (2017). Accessed Dec. 27, 2017.

[30] kkapsner. 2017. CanvasBlocker: A Firefox Plugin to block the canvas-API. https://github.com/kkapsner/CanvasBlocker/. (2017). Accessed Dec. 25, 2017.

[31] Georgios Kontaxis and Monica Chew. 2015. Tracking protection in Firefox for privacy and performance. *arXiv preprint arXiv:1506.04104* (2015).

[32] Balachander Krishnamurthy and Craig E Wills. 2006. Generating a privacy footprint on the internet. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. ACM, 65–70.

[33] Pierre Laperdrix. 2017. Fingerprint Central. https://fpcentral.irisa.fr/. (2017). Accessed Oct 31, 2017.

[34] Pierre Laperdrix, Benoit Baudry, and Vikas Mishra. 2017. FPRandom: Randomizing core browser objects to break advanced device fingerprinting techniques. In *9th International Symposium on Engineering Secure Software and Systems (ESSoS 2017)*.

[35] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2015. Mitigating browser fingerprint tracking: multi-level reconfiguration and diversification. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press, 98–108.

[36] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. 2016. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 878–894.

[37] Pedro Leon, Blase Ur, Richard Shay, Yang Wang, Rebecca Balebako, and Lorrie Cranor. 2012. Why Johnny can't opt out: a usability evaluation of tools to limit online behavioral advertising. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 589–598.

[38] Jonathan R Mayer and John C Mitchell. 2012. Third-party web tracking: Policy and technology. In *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 413–427.

[39] meh. 2017. Blender: Blend in the crowd by faking to be the most common Firefox browser version, operating system and other stuff. https://github.com/meh/blender. (2017). Accessed Dec. 25, 2017.

[40] Georg Merzdovnik, Markus Huber, Damjan Buhov, Nick Nikiforakis, Sebastian Neuner, Martin Schmiedecker, and Edgar Weippl. 2017. Block me if you can: A large-scale study of tracker-blocking tools. In *Proceedings of the 2nd IEEE European Symposium on Security and Privacy (IEEE EuroS&P)*.

[41] Keaton Mowery and Hovav Shacham. 2012. Pixel perfect: Fingerprinting canvas in HTML5. *Proceedings of W2SP* (2012), 1–12.

[42] Mozilla. 2017. Firefox Add-ons. https://addons.mozilla.org/en-US/firefox/. (Dec. 2017).

[43] Multiloginapp. 2017. How Canvas Fingerprint Blockers Make You Easily Trackable. https://multiloginapp.com/how-canvas-fingerprint-blockers-make-you-easily-trackable/. (2017). Accessed Dec 19, 2017.

[44] Net-Comet. 2017. Glove: Chrome Web Store. https://chrome.google.com/webstore/detail/glove/abdgoalibdacpnmknnpkgnfllphboefb?hl=en. (2017). Accessed Dec. 25, 2017.

[45] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. 2015. Privaricator: Deceiving fingerprinters with little white lies. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 820–830.

[46] NiklasG. 2017. Stop Fingerprinting: Add-ons for Firefox. https://addons.mozilla.org/en-US/firefox/addon/stop-fingerprinting/. (2017). Accessed Dec. 25, 2017.

[47] Liam Paninski. 2003. Estimation of entropy and mutual information. *Neural computation* 15, 6 (2003), 1191–1253.

[48] Mike Perry, Erinn Clark, Steven Murdoch, and Georg Koppen. 2017. The Design and Implementation of the Tor Browser. https://www.torproject.org/projects/torbrowser/design/#privacy. (2017). Accessed Jul 21, 2017.

[49] Reşat. 2017. Blend In: Add-ons for Firefox. https://addons.mozilla.org/en-US/firefox/addon/blend-in/. (2017). Accessed Dec. 25, 2017.

[50] Franziska Roesner, Tadayoshi Kohno, and David Wetherall. 2012. Detecting and Defending Against Third-party Tracking on the Web. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI'12)*. USENIX Association, Berkeley, CA, USA, 12–12. http://dl.acm.org/citation.cfm?id=2228298.2228315

[51] Samy Sadi. 2017. No Enumerable Extensions: Firefox addon that lets you hide installed extensions and avoid being fingerprinted based on them. https://github.com/samysadi/no-enumerable-extensions. (2017). Accessed Jan. 13, 2017.

[52] Sagar Shivaji Salunke. 2014. *Selenium Webdriver in Python: Learn with Examples* (1st ed.). CreateSpace Independent Publishing Platform, USA.

[53] Iskander Sanchez-Rola, Igor Santos, and Davide Balzarotti. 2017. Extension Breakdown: Security Analysis of Browsers Extension Resources Control Policies. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 679–694. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/sanchez-rola

[54] Martin Springwald. 2017. Privacy-Extension-Chrome: Provides Privacy for Chrome. https://github.com/marspr/privacy-extension-chrome. (2017). Accessed Dec. 25, 2017.

[55] Oleksii Starov and Nick Nikiforakis. 2017. Xhound: Quantifying the fingerprintability of browser extensions. In *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE, 941–956.

[56] StatCounter. 2018. StatCounter Global Stats. http://gs.statcounter.com/. (2018). Accessed Feb. 12, 2018.

[57] Mozilla Support. 2017. Tracking Protection. https://support.mozilla.org/en-US/kb/tracking-protection. (2017). Accessed Dec. 27, 2017.

[58] The Tor Project. 2017. Users. Tor Metrics page: https://metrics.torproject.org/userstats-relay-country.html. (Dec. 2017).

[59] Christof Ferreira Torres, Hugo Jonker, and Sjouke Mauw. 2015. FP-Block: usable web privacy by controlling browser fingerprinting. In *European Symposium on Research in Computer Security*. Springer, 3–19.

[60] Hamilton Ulmer. 2010. Browsing Sessions. Mozilla's Blog of Metrics: https://blog.mozilla.org/metrics/2010/12/22/browsing-sessions/. (Dec. 2010).

[61] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. Fp-Scanner: The Privacy Implications of Browser Fingerprint Inconsistencies. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 135–150. https://www.usenix.org/conference/usenixsecurity18/presentation/vastel

[62] Jon Watson. 2008. VirtualBox: Bits and Bytes Masquerading As Machines. *Linux J.* 2008, 166 (Feb. 2008). http://dl.acm.org/citation.cfm?id=1344209.1344210

[63] Ting-Fang Yen, Yinglian Xie, Fang Yu, Roger Peng Yu, and Martin Abadi. 2012. Host Fingerprinting and Tracking on the Web: Privacy and Security Implications.. In *NDSS*.