FP-Inconsistent: Measurement and Analysis of Fingerprint Inconsistencies in Evasive Bot Traffic

Hari Venugopalan
hvenugopalan@ucdavis.edu
UC Davis

Shaoor Munir
smunir@ucdavis.edu
UC Davis

Shuaib Ahmed
shuahmed@ucdavis.edu
UC Davis

Tangbaihe Wang
monwang@ucdavis.edu
UC Davis

Samuel T. King
kingst@ucdavis.edu
UC Davis

Zubair Shafiq
zubair@ucdavis.edu
UC Davis

ABSTRACT

Browser fingerprinting is used for bot detection. In response, bots have started altering their fingerprints to evade detection. We conduct the first large-scale evaluation to study whether and how altering fingerprints helps bots evade detection. To systematically investigate such evasive bots, we deploy a honey site that includes two anti-bot services (DataDome and BotD) and solicit bot traffic from 20 different bot services that purport to sell "realistic and undetectable traffic." Across half a million requests recorded on our honey site, we find an average evasion rate of 52.93% against DataDome and 44.56% evasion rate against BotD. Our analysis of fingerprint attributes of evasive bots shows that they indeed alter their fingerprints. Moreover, we find that the attributes of these altered fingerprints are often inconsistent with each other. We propose FP-Inconsistent, a data-driven approach to detect such inconsistencies across *space* (two attributes in a given browser fingerprint) and *time* (a single attribute at two different points in time). Our evaluation shows that our approach can reduce the evasion rate of evasive bots by 44.95%-48.11% while maintaining a true negative rate of 96.84% on traffic from real users.

1 INTRODUCTION

The prevalence of bots on the web is on the rise [15]. Per recent reports, bots constitute around 49.6% of online traffic [31], with 64.5% of those being bots that engage in malicious activity. Bad actors employ bots to launch a multitude of attacks [11, 13, 14, 47, 56]. To counter such attacks, anti-bot services aim to detect and block bot traffic. Prior research has shown that anti-bot services use browser fingerprinting to detect bots without disrupting the user experience of legitimate users [5, 63]. Browser fingerprints capture attributes of the web browser sending web requests and anti-bot services attempt to use differences in these attributes to distinguish bots from real users [67].

Blackhat marketplaces [6, 52, 55], however, advertise realistic and undetectable bot traffic as a service. The traffic from such services constitute impression fraud and serve to artificially boost website engagement for monetization [18, 35, 56]. To evade detection, bots from these services are likely altering their fingerprint attributes that are used by anti-bot services for detection [26, 37]. We refer to such bots as *evasive* bots. It is important to characterize evasive bots and their fingerprints to improve the effectiveness of bot detection.

Prior research has studied bot fingerprints by employing their own bots [3, 63] or studying naturally discovered bots on their honey sites [40]. Thus, this line of work is not geared towards capturing the evasive fingerprints used by bots seeking to evade detection in the wild. Wu et al. performed a large-scale characterization of the differences between human and bot fingerprints in the wild [67]. However, they did not specifically characterize evasive bots since they treat their bot detection system decisions as ground-truth to distinguish between the fingerprints of bots and real users.

To fill this gap, we perform the first large-scale measurement of evasive bots that evade anti-bot services. To this end, we drive traffic from different bot services from blackhat marketplaces to different instances of our honey site. That way, the requests recorded at each honey site instance can be attributed to a bot service from whom we purchased traffic. These operators advertise their traffic as being realistic and natural, indicating that they likely employ evasive bots to ensure that they do not get detected. We integrate two commercial bot detection services (DataDome and BotD) on our honey site for bot detection. We also instrument our honey site to collect a range of fingerprint attributes.

We collect 507,080 requests from 20 different bot services, with DataDome and BotD detecting 55.44% and 47.07% of these requests respectively. We analyze fingerprint attributes from different bot services to identify different sets of attributes that are effective at evading DataDome and BotD individually as well as attributes that are effective at evading both anti-bot services. Our analysis reveals spatial inconsistencies (among the different attributes of a given request) and temporal inconsistencies (across requests originating from the same device). These include obvious inconsistencies that cannot exist for real users, thereby making them useful signatures to detect bots.

We use observations from our analysis to develop FP-Inconsistent, a data-driven approach to discover inconsistencies in fingerprint attributes for bot detection. FP-Inconsistent relies on the insight that real devices can only have a limited number of hardware and software configurations that are reflected in fingerprint attributes. Evasive bots, in their attempt to evade detection, emulate a large number of invalid or extraneous configurations. FP-Inconsistent leverages this mismatch between the expected and observed number of configurations to identify potential inconsistencies among evasive bot fingerprints. It does so by calculating the number of configurations for pairs of fingerprint attributes from evasive bots and identifying inconsistencies among attribute pairs that exhibit a higher-than-expected number of configurations.

Using this approach, we generate inconsistency rules that can be readily deployed by anti-bot services. Prior research focusing on the use of inconsistencies for bot detection [62, 63] has predominantly relied on one-off anecdotes to define inconsistencies that are not data-driven and hence do not scale. FP-Inconsistent systematizes the generation of inconsistency rules for bot detection.

Our evaluation shows the rules generated by FP-Inconsistent are able to achieve 48.11% and 44.95% reduction in traffic that evades DataDome and BotD respectively while maintaining a true negative rate of 96.84% on real user traffic. Our experiments also show that FP-Inconsistent does not incur false positives with most privacy-enhancing technologies. We open-source our honey site architecture and inconsistency rules for public use at this link.

Our work makes the following contributions:

- A **novel honey site architecture to establish reliable ground-truth** for evasive bots.
- A **large-scale measurement and analysis of fingerprint attributes** for evasive bots that are able to evade anti-bot services.
- A **data-driven approach to discover inconsistencies in fingerprint attributes** for detecting evasive bots.

2 BACKGROUND AND RELATED WORK

2.1 Evaluation of bot detection services

Anti-bot services on the web generally employ machine learning to determine if an incoming request was sent by a human or a bot[9]. These services rely on several signals captured through different browser fingerprinting APIs, request headers, and behavior characteristics on a website[63]. Prior research has attempted to measure the accuracy of anti-bot services and understand their detection techniques. Azad et al. [3] analyzed 15 different anti-bot services, 14 of which used modern fingerprinting techniques such as WebGL and Canvas-based fingerprinting. While these services rely on inconsistencies in fingerprint attributes to detect bots, we show how they can be more extensive in using them to improve bot detection (Section 7).

Azad et al. also tried to evaluate the performance of these services by deploying their own bots and measuring their evasiveness. In contrast, we evaluate anti-bot services on requests from bots in the wild.

2.2 Analysis of bot traffic in the wild

Xigao et al. [40] studied the prevalence of "malicious" bots in the wild. They use the behavior of bots (indulging in credential stuffing, not honoring bots.txt, etc) to characterize them as malicious. Such characterization is not applicable for bots indulging in impression fraud since these bots don't exhibit any explicit behavior that can be leveraged for detection. Further, their approach draws traffic from bots in general and they do not include mechanisms to isolate evasive bots visiting their honey sites that seek to evade detection.

Wu et al. [67] analyzed browser fingerprints from 36 billion requests on 14 commercial websites. Their analysis shows that adversarial bots (bots that change their fingerprints to avoid detection) have significantly different properties compared to benign bots. While they conducted the largest study (at the time of writing) of bots in the wild, their ground-truth relies on decisions by F5 Inc.[17], a commercial anti-bot service. Thus, without a more robust mechanism to collect ground-truth, their approach cannot analyze bots that can evade commercial anti-bot services such as F5.

Browser Polygraph [38] employs machine learning to detect bots that indulge in account takeover fraud (ATO). They predict if the fingerprint attributes in a request are consistent with the request's reported `User-Agent`. In contrast, our work proposes a data-driven and semi-automatic technique to discover inconsistencies between any pair of fingerprint attributes (which includes but is not limited to the `User-Agent`) to combat impression fraud. Further, similar to the work of Wu et. al, their approach could be bolstered with more robust ground-truth since they rely on tags from FinOrg (a financial organization) to provide ground-truth for evaluation. In our work, our novel honey site architecture provides ground-truth to isolate traffic sent from different bot services.

2.3 Challenges in bot detection

We discuss some of the common techniques used by bots to evade detection.

Polymorphism: Certain bots morph their User-Agent or other attributes (i.e., fingerprints) to appear as benign website visitors for evasion [3, 38]. Iliou et. al [30] showed that while machine learning algorithms can detect simple bots with a precision and recall of 95% and 97% respectively, more advanced bots, i.e. bots that change their fingerprints, result in a drop in accuracy to only 55%.

Behavioral Mimicry: Bots also simulate human-like behavior to evade behavioral analysis systems, including mimicking mouse movements, keystrokes, browsing patterns, and human text input[4]. Bot detection systems use these movements as "Human Interactive Proofs (HIPs)"[22, 23] to determine if a website visitor is a bot or a human. Jing et. al. [36] developed a bot framework for bots to generate keystrokes and mouse clicks that closely resemble human actions to evade detection.

3 THREAT MODEL

In this paper, we focus on bots committing impression fraud [56]. Web publishers who seek to artificially inflate the engagement on their websites indulge in this type of fraud. Inflating engagement allows these publishers to monetize and profit from their websites through ads, even when they cannot guarantee visits to their website from legitimate users. Advertisers pay publishers for impressions of their ad (views, clicks, etc) on the publisher's website. However, only impressions recorded from legitimate users are useful to advertisers. Publishers who do not receive traffic from legitimate users could employ bots to record these impressions to get paid by advertisers without delivering any useful impressions to them. We focus on bots indulging in impression fraud over other types of fraud (such as credential stuffing, account takeover, etc), since these bots do not have a need to perform specific actions [3, 40] to reach their goal, thereby making it more challenging to detect them.

In our threat model, we consider publishers who incorporate anti-bot services on their websites to provide assurance of traffic from legitimate users, while employing evasive bots to evade detection.

4 MEASUREMENT INFRASTRUCTURE

In this section, we describe our measurement infrastructure including the design of our novel honey site architecture. We design our measurement infrastructure to satisfy three requirements that enable us to reliably characterize evasive bots: first, we need reliable ground-truth that we only record requests from evasive bots of interest and no other entities (real users or other bots). Second, we need decisions from bot detection services on each request to isolate requests that evade detection. Third, we need to collect browser attributes that constitute browser fingerprints in these requests to analyze attributes that help with evasion.

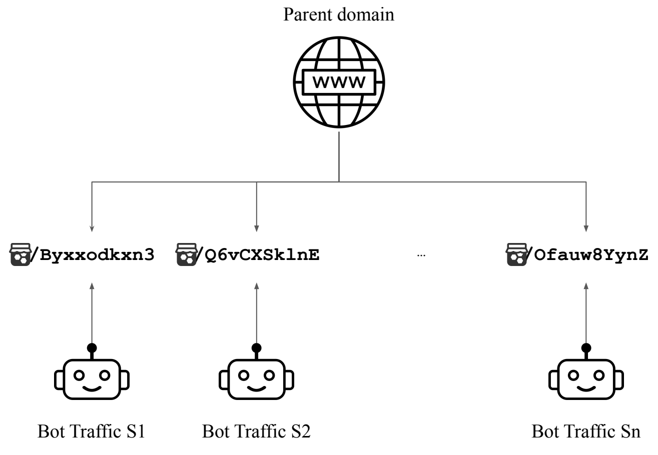


Figure 1: To collect requests from different bot services, we create multiple versions of the same honey site under the same domain. The only difference between these versions is the presence of different random strings in their URL. We then drive traffic from different bot services to different versions of the honey site.

4.1 Honey site architecture

Using obscure domain names for honey sites [40] cannot guarantee that the honey sites only receive requests from evasive bots. Bots that automatically send requests to such domains are typically indexing bots that visit new websites added to domain registries and other sources of DNS records. Examples of such bots include search engine bots that do not have a need to conceal their identities. In fact, Google's bots announce their identity through their User-Agent [24]. While evasive bots may also send requests to such domains, the absence of a mechanism to isolate those requests makes it challenging to analyze them. Evasive bots indulging in impression fraud do not have a need to perform specific actions to record views or impressions. Hence, such bots cannot be detected based on their actions/behavior [3, 40].

To overcome the challenge of only recording requests from evasive bots, we deploy multiple versions of the same honey site under the same domain. These versions only differ in terms of the presence of arbitrarily chosen strings in their URL. We do not record requests that do not contain one of these strings in the URL to ensure that we do not record requests from real users or generic bots that discover our domain. We also share URLs having different arbitrary strings with different bot services. Thus, these URL strings enable the isolation of requests received from different bot services. As a concrete example, if *example.com* is the domain of our honey site, *example.com/XXXXX*, *example.com/YYYYY*, and *example.com/ZZZZZ* would constitute different versions of the honey site. We then purchase traffic from 3 different bot services to each send requests to one of these URLs. Real users and other generic bots who may stumble upon our site, will not know these strings and hence cannot include such strings in their requests. Thus, we can ensure that we only record requests from the bot services where we made our purchases using these URL strings. Figure 1 shows an overview of the honey site architecture.

4.2 Anti-bot services

We integrated two popular commercial anti-bot services on our honey site: DataDome [12] and BotD [20]. Both DataDome and BotD provide real-time decisions on requests received on a website. DataDome advertises real-time decisions for a request in under 2 milliseconds with an overall accuracy of 99% and a false positive rate of 0.01%. Prior research on bot detection has explored DataDome [3, 69]. BotD is a bot detection service from the developers of the popular open-source fingerprinting library FingerprintJS [19] that is widely used in industry and academia [3, 38, 40, 63]. BotD claims to use "the most advanced device fingerprinting technology", and reports a detection accuracy of 99.5%.

We integrate JavaScript libraries of both these services on our honey site[1]. These libraries collect browser fingerprints of the browser visiting the honey site and relay them to their own servers. The servers then respond with the decision of whether a real human or a bot originated the request.

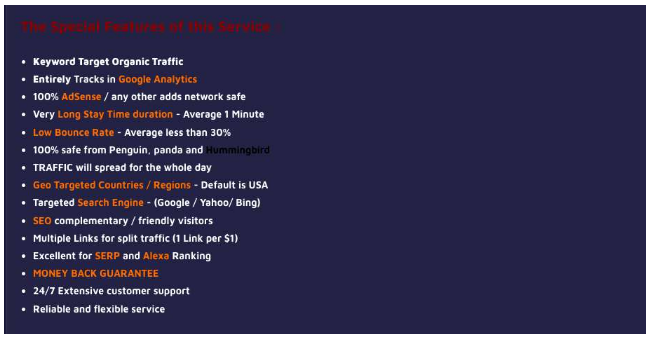
These services are black-boxed and do not provide information on fingerprint attributes they use as features to decide if a request originates from a bot. To determine this information, we crawl our honey site using OpenWPM [48]. OpenWPM is an open-source tool to track the behavior of different web elements, including scripts, on a webpage.

Table 5 in Appendix B highlights the different browser APIs accessed by DataDome and BotD. Both services access a number of fingerprinting APIs such as `navigator.plugins`, `HTMLCanvasElement.getContext`, `navigator.userAgent`, and more. We find that DataDome collects more attributes from each request than BotD. In Section 5, we see that DataDome has higher bot detection accuracy than BotD, which could potentially stem from these additional attributes.

4.3 Bot services

We made purchases from multiple online bot services to send traffic to different versions of our honey site. We make our purchases from the SEOClerks [52], an underground marketplace for web traffic where bot services advertise their traffic as being real, organic, and Adsense safe to boost website engagement. Their claims of being able to send real and organic

[1]As required by DataDome, for each request, we also make an API call from our server to get their decision



The Special Features of this Service :

- Keyword Target Organic Traffic
- Entirely Tracks in Google Analytics
- 100% AdSense / any other adds network safe
- Very Long Stay Time duration - Average 1 Minute
- Low Bounce Rate - Average less than 30%
- 100% safe from Penguin, panda and Hummingbird
- TRAFFIC will spread for the whole day
- Geo Targeted Countries / Regions - Default is USA
- Targeted Search Engine - (Google / Yahoo/ Bing)
- SEO complementary / friendly visitors
- Multiple Links for split traffic (1 Link per $1)
- Excellent for SERP and Alexa Ranking
- MONEY BACK GUARANTEE
- 24/7 Extensive customer support
- Reliable and flexible service

Figure 2: Screenshot from a bot service on SEOClerks making claims about sending organic traffic to drive engagement on websites. The claims likely suggest that the bot service employs evasive bots to took real users.

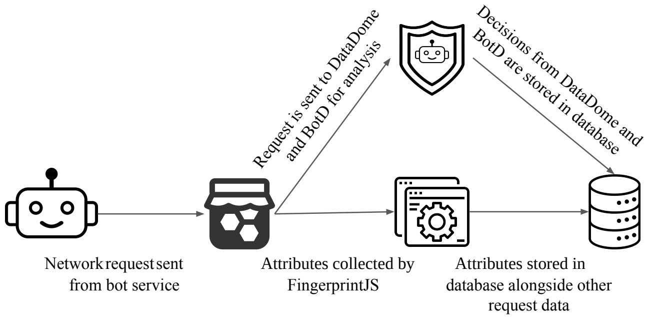

Request is sent to DataDome and BotD for analysis
Decisions from DataDome and BotD are stored in database
Network request sent from bot service
Attributes collected by FingerprintJS
Attributes stored in database alongside other request data

Figure 3: Overview of our data collection pipeline.

traffic indicate that they are likely using evasive bots that alter their fingerprints to look like real users. Figure 2 captures a screenshot from a bot service on SEOClerks making such claims about their traffic. We share URLs with different version strings with different bot services to identify the bot services of each request on our honey site.

4.4 Data Collection

To characterize the differences in the fingerprint attributes of evasive bots, we extract information from different browser APIs and properties upon loading our honey site in the browser. We send this information to our server in an http request. We use FingerprintJS [19], a widely deployed browser fingerprinting library to capture this information. FingerprintJS captures over 30 different fingerprint attributes including the list of fonts installed on the browser, the number of CPU cores on the device running the browser, the amount of memory on the device, and the languages supported by the browser. While we focus on the attributes captured by FingerprintJS in this paper, both our measurement analysis (Section 5) and our methodology to discover inconsistencies

(Section 7) are compatible with other fingerprint attributes too.

5 ANALYSIS

Table 1: Overview of different bot services sending traffic to our honey site and their evasion rates against DataDome and BotD.

Bot Service	Num. Requests	DataDome Evasion Rate	BotD Evasion Rate
S1	121500	44.01%	71.58%
S2	63708	42.99%	72.29%
S3	54746	74.91%	10.26%
S4	47278	38.65%	73.85%
S5	40087	23.86%	72.65%
S6	32447	71.81%	5.45%
S7	28940	2.56%	39.99%
S8	26335	80.43%	28.9%
S9	23412	78.29%	19.33%
S10	18967	15.77%	59.23%
S11	17996	6.55%	59.36%
S12	7010	5.05%	51.44%
S13	5119	6.95%	50.52%
S14	4920	83.74%	90.08%
S15	4219	11.14%	100%
S16	4174	4.48%	0.02%
S17	2999	74.66%	7.9%
S18	1430	20.7%	100%
S19	1411	9.92%	100%
S20	382	97.12%	97.12%

Over a period of 3 months, from September 2023 to November 2023, we received 507,080 requests from 20 different bot services. We first report the detection rate of the anti-bot services and then compare fingerprint attributes of bots that evade detection against those that were detected. This analysis helps understand the attributes used by bots for evasion and ways to overcome them.

Table 1 shows the statistics of the traffic obtained from each bot service along with the evasion rate against the two anti-bot services on our honey site (DataDome and BotD). Among the 507,080 requests we received, 55.44% of requests were detected by DataDome, and 47.07% of requests were detected by BotD. These results show that a significant proportion of bots are able to evade anti-bot services.

Takeaway 1: Our measurement shows that evasive bots are not reliably detected by commercial anti-bot services.

5.1 IP addresses for evasion

We observed requests on our honey site that contained IP addresses with Autonomous System Numbers (ASNs) mapping to cloud services such as Amazon Web Services (AWS). Since such ASNs are likely flagged as those used by bots [10, 28], we check the ASNs of the requests we received against public ASN block lists [8, 27]. We report that 82.54% of requests originated from flagged ASNs. Among these, 52.93% of requests evade BotD and 43.17% of requests evade BotD. These results show that evasive bots are able to evade detection even when they send requests from flagged ASNs.

We suspect that anti-bot services may not rely on ASN block lists since real users and bots can share the same ASNs but can send requests from different IP addresses. Accordingly, we ran similar analysis with blocked IP addresses using MaxMind's minFraud API [43]. Consistent with findings in prior research [40], we find that IP block lists offer limited coverage (15.86%). More interestingly, among the IP addresses that were covered, requests from 48.1% were able to evade DataDome and 68.85% were able to evade BotD.

In conclusion, we see that a significant number of bots that sent requests from blocked IP addresses and ASNs were able to evade both DataDome and BotD. This indicates that evasive bots don't merely send requests from IP addresses not captured by block lists to evade detection.

Takeaway 2: Evasive bots do not merely rely on sending requests from IP addresses that are not captured by block lists to evade detection.

5.2 Fingerprint attributes for evasion

Since evasive bots don't merely rely on IP addresses, we systematically analyze the browser fingerprint attributes in their requests to identify those used for evasion. Concretely, we train models to distinguish between the requests that were detected by and evaded DataDome and BotD respectively. We then use techniques from the explainability of machine learning to identify fingerprint attribute values that help with evasion. We then explore the values of these attributes on requests from bot services that were most successful with evasion to verify that they enable evasion.

5.2.1 *Identifying fingerprint attributes.*

We train two random forest classifiers using XGBoost [68] to distinguish between the requests that were detected and evaded DataDome and BotD respectively. Each classifier takes as input fingerprint attributes from each request (discussed in Section 4.4) and provides a binary decision on whether that request would detected by the respective anti-bot service.

We performed a 90-10 split on the requests to train the classifiers. The classifier for BotD attained an accuracy 97.8% on the training set and 97.71% on the test set while the classifier for DataDome attained an accuracy of 82.09% on the training set and 81.66% on the test set. These high accuracy values indicate that the fingerprint attributes of requests that evade the two anti-bot services are considerably different from those of requests detected by them.

Table 2: Top 5 most important fingerprint attributes that help evade DataDome and BotD.

DataDome	BotD
Vendor Flavors	Vendor Flavors
Plugins	Plugins
Screen Frame	Touch Support
Hardware Concurrency	Vendor
Forced Colors	Contrast

We use SHapley Additive exPlanations or SHAP [53] to analyze these classifiers to identify fingerprint attributes that result in evasion. Table Table 2 lists the top 5 attributes that help evade DataDome and BotD respectively.

5.3 Fingerprint attributes among evasive bots

We now inspect the attribute values of requests from bot services with high evasion rates to see if they exploit the attributes identified in Table 2 for evasion. Concretely, we compare attribute values across bot services that have high evasion rates against those that have low evasion rates.

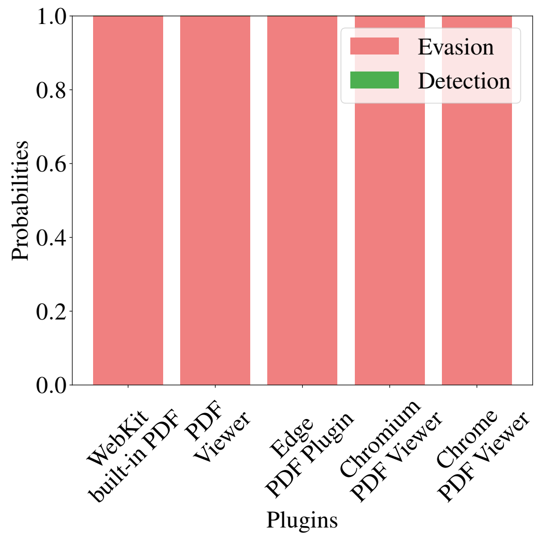


Figure 4: Bar plot showing the probability of PDF plugins that have the highest probability of evasion against BotD. This plot shows that the presence of any plugin helps evade BotD.

5.3.1 Bots evading BotD. We inspected requests from the top 3 bot services with the highest evasion rates against BotD (S15, S18, and S19 in Table 1) and the top 3 bot services with the lowest evasion rates against BotD (S6, S16, and S17 in 1). We record 7,132 requests from the top 3 bot services evading BotD and report 100% evasion among them. We record 39,620 requests from the top 3 bot services that are detected by BotD and report an evasion rate of 5.11% among them.

We did not observe significant differences between the values of `Vendor Flavors`, `Vendor`, and `Touch Support` attributes among requests from these bot services. 99.91% of requests from services evading BotD supported the Chrome PDF Viewer plugin, while 100% of requests detected by BotD did not support any plugins. Motivated by these stark differences, we further investigate the impact of plugins on evading BotD. Concretely, from all requests received on our honey site, we compute the probability of evading BotD when supporting any one of 5 commonly used PDF plugins. Figure 4 shows that the presence of any PDF plugin nearly guarantees evasion against BotD.

5.3.2 Bots evading DataDome. We similarly inspect requests from the top 3 bot services with the highest and lowest evasion rates against DataDome. We record 52,746 requests from the top 3 bot services evading DataDome (S8, S9, and S17 in Table 1) having 79.15% evasion among them. We record 51,110 requests from the top 3 bot services detected by DataDome (S7, S11, and S16 in Table 1) with an evasion rate of 4.12%.

100% of requests from the top 3 bot services having the highest evasion rate against DataDome did not support any plugins. However, 56.45% of requests from the 3 bot services with the lowest evasion rate against DataDome did not support any plugins either. Analyzing the `Screen Frame` and `Forced Colors` attributes revealed certain values that always result in detection. However, we did not observe values for these attributes that help with evasion.

Figure 5 compares cumulative probability distribution functions (CDFs) of the number of CPU cores (captured by `hardwareConcurrency`) on requests from bot services with high evasion rates over DataDome against the values on requests from bot services with low evasion rates over DataDome. These results indicate that low values for `hardwareConcurrency` help evade DataDome. Concretely, 84.7% of requests from bot services with a high evasion rate against DataDome had fewer than 8 cores. In contrast, only 38.16% of requests from bot services detected by DataDome had fewer than 8 cores. To further assess the impact of `hardwareConcurrency`, we disregard requests that contain values for `Screen Frame` and `Forced Colors` that always lead to evasion. Now, 84.7% of requests from bot services

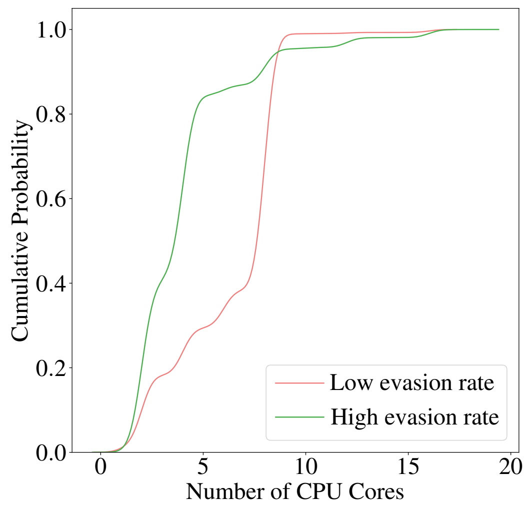


Figure 5: Cumulative probability distribution function (CDF) plots of the number of CPU cores recorded on requests from bot services that had the highest evasion rate over DataDome against those that had the the lowest evasion rate over DataDome.

with a high evasion rate against DataDome have fewer than 8 cores while only 19.05% of requests from bot services with a low evasion rate against DataDome have fewer than 8 cores.

Evasive bots evading DataDome ensure certain values for combinations of attributes. This is different from evasive bots evading BotD that ensured certain values for one set of attributes (plugins). We investigate more combinations of attributes that help evade DataDome in Appendix C.

5.3.3 Bots evading DataDome and BotD. Requests from two different bot services have over 80% evasion rate against both DataDome and BotD (S14 and S20 in Table 1). We received 5,302 requests from these services which have an 84.7% evasion rate against DataDome and 90.59% evasion against BotD.

We observe that 83.77% of these requests have fewer than 8 CPU cores indicating that they exploit hardware concurrency to evade DataDome. Interestingly, 93.02% of these requests do not have any plugins, indicating that they do not exploit plugins to evade BotD. They exploit `touchSupport`, a different blind spot of BotD for evasion. Concretely, 78.36% of requests from the bot services evading both DataDome and BotD support touch events, while only 3.95% of requests from the top 3 bot services having the lowest evasion rate against BotD support touch events. In contrast, only 0.07% of requests from the top 3 bot services that only evaded BotD (Section 5.3.1) showed support for touch events and 8.61% of requests from the top 3 bot services that only evaded DataDome (Section 5.3.2) showed support for touch events.

Takeaway 3: Evasive bots exploit either `touchSupport` and `plugins` to evade BotD. They exploit `hardwareConcurrency` to evade DataDome.

6 INCONSISTENCY ANALYSIS

From our analysis in the previous section, we see ensuring certain values for certain fingerprint attributes helps bots evade detection. One way in which evasive bots could accomplish this would be to send requests from devices that would contain the desired values for attributes. For example, evasive bots could send requests from devices containing 4 CPU cores to ensure a value of 4 for `hardwareConcurrency`. Alternatively, evasive bots could alter browser APIs and device properties to present their desired values for fingerprint attributes [41]. In this case, an evasive bot could alter the `hardwareConcurrency` attribute of the `navigator` object to return 4 on a device that may not have 4 CPU cores.

In this section, we describe various inconsistencies in fingerprint attributes among the requests received on our honey site. These inconsistencies provide evidence of bots altering browser APIs since such inconsistencies are extremely unlikely to occur when using real devices. We use insights from these inconsistencies to develop FP-Inconsistent, our semi-automated technique to generate inconsistency rules to detect evasive bots (Section 7).

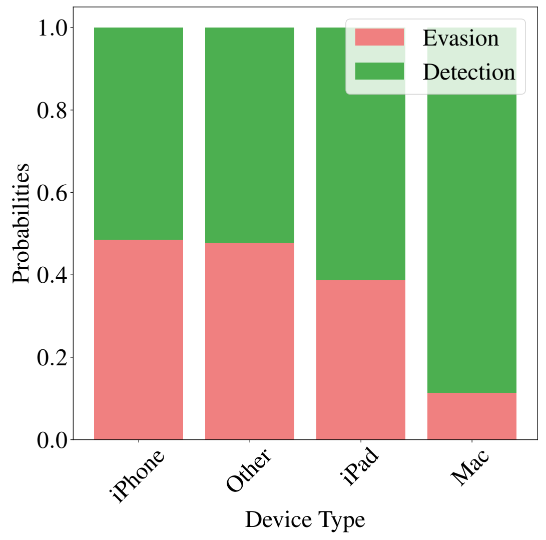


Figure 6: Bar plot showing the top 4 device types (inferred from the User-Agent) that have the highest probability of evading DataDome.

6.1 Inconsistencies across fingerprint attributes

Figure 6 shows the top 4 device types (inferred using the `User-Agent` property of the browser's `navigator` object) that have the highest probability of evading DataDome among the requests recorded on our honey site. From the figure, we see that iPhones have the highest probability of evasion (around 50%). We now look at other fingerprint attributes to determine if evasive bots sent requests from real iPhones or if they altered the `navigator` object on their browser to have their devices appear as iPhones. Since iPhones have a fixed set of screen resolutions (12 resolutions [16]), we inspect the spread of screen resolutions captured on requests from iPhones. Upon inspection, we found 83 unique screen resolutions from iPhones, out of which 42 were present among those requests from iPhones that evaded DataDome. We also find that 9 out of the top 10 screen resolutions that have the highest probability of evading DataDome among requests claiming to use iPhones do not exist in the real world. We visualize these probabilities in Figure 7. This provides strong evidence that bots alter browser APIs to show that they use iPhones rather than using actual iPhones.

From this evidence, we see that while bots alter browser APIs, it is difficult for them to ensure that all fingerprint attributes remain consistent with their alterations. Thus, inconsistencies across fingerprint attributes can be leveraged for bot detection since real users are unlikely to have such inconsistencies. In Section 7 we discuss our systematic, data-driven, semi-automatic approach to discover such inconsistencies to improve bot detection.

Takeaway 4: While bots alter fingerprint attributes for evasion, they do not ensure that all attributes are consistent with their alteration. A particular value for a given attribute mapping to a large number of values for another attribute provides an avenue to discover inconsistencies.

6.2 Inconsistencies across fingerprint attributes and IP addresses

Some bot services advertised sending traffic from specific geographic regions (USA, Mexico, France, etc). Having this ability to send requests from specific regions suggests that the bot services are likely altering attributes that capture the geographical location of their devices. This alteration introduces potential inconsistencies if the bot services did not ensure that all attributes point to the same region.

We analyzed requests from 4 different bot services who claimed to send requests from the United States, Canada, Europe, and France respectively. We first used MaxMind's GeoLite2 database [42] to extract the geolocation from the IP address of the requests from these services. We took a conservative approach when determining if the inferred geolocation matched the region advertised by the bot service. Concretely, we considered locations at the same UTC offset to be a match. For example, when analyzing requests from the bot service who advertised sending requests from France, we considered all requests whose geolocations mapped to any valid UTC offset that could overlap with France (such as

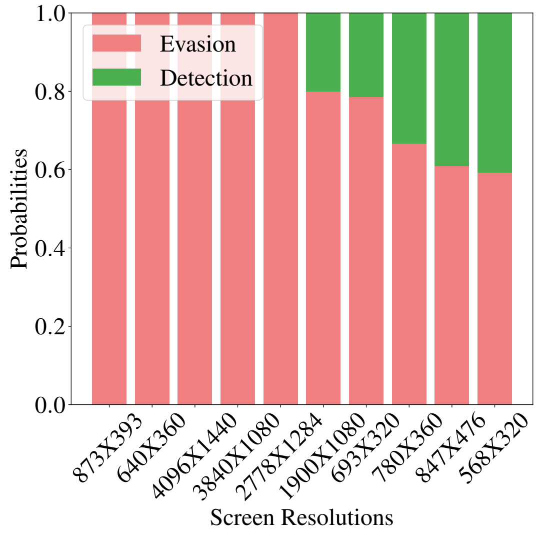


Figure 7: Bar plot showing the top 10 screen resolutions among requests received from iPhones (inferred using the User-Agent) that have the highest probability of evasion against DataDome. 9 out of these 10 resolutions do not exist in the real world indicating an inconsistency that can be leveraged to detect bots.

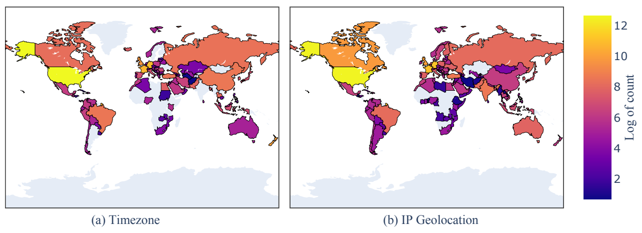


Figure 8: Plots showing a heatmap of the geographical location of requests inferred using the timezone attribute of the navigator object and the IP address. Different regions lighting up in the two heatmaps indicate that while bots alter the navigator object or IP address or both to change their geographical location, they do not ensure that the location inferred using both is consistent.

Europe/Berlin) to also originate from France. With this approach, over 90% of requests from each of the 4 bot services matched the advertised geographical location.

However, we observed significant differences when repeating the same analysis using the browser's timezone API [44] to infer location. We still used the same conservative approach and merely replaced the geolocation inferred from the IP address with the timezone. Only 76.52% of requests mapped to UTC offsets in Canada among the requests from the bot service that advertised traffic from Canada. More alarmingly, we observed that only 56% of requests mapped to UTC offsets in Europe among the requests from the bot service that advertised traffic from Europe. In contrast, we observed 92.44% of requests to originate from Canada and 99.83% of requests to originate from Europe from the corresponding bot services when inferring the geolocation from the IP address. Motivated by these results, we visualize the geographical spread of requests based on both approaches in Figure 8. The figure reveals a number of inconsistencies in geographical locations which also constitute inconsistencies for bot detection.

Takeaway 5: Bots alter their IP addresses, fingerprint attributes or both to fulfill promises of sending requests from specific locations.

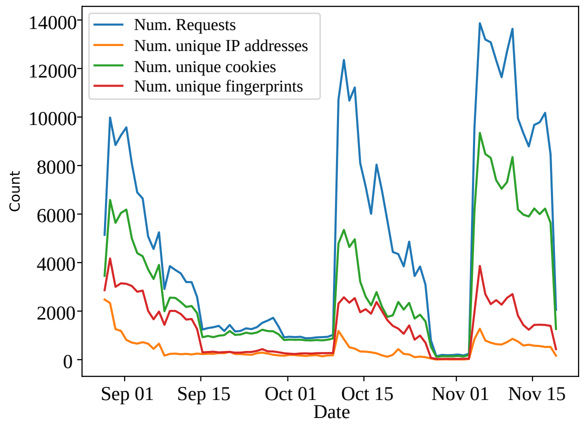


Figure 9: Temporal distribution of traffic on our honey site.

6.3 Inconsistencies across time

Figure 9 shows the temporal spread of requests received on our honey site over time. The plot shows the number of requests, the number of unique IP addresses, the number of unique values for Cookies set by our honey site, and the number of unique FingerprintJS fingerprints seen per day.

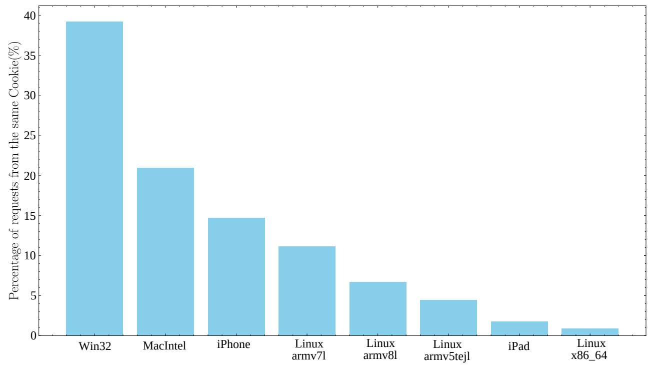


Figure 10: Percentage of requests seen across different values of the `platform` attribute of the `navigator` object for the same Cookie (same device). The diverse spread of values provides strong evidence of bots altering the `platform` attribute since it cannot change otherwise for the same device.

From the figure, we see that even after 2 months, we receive requests with previously unseen fingerprints and IP addresses. More interestingly, the spikes in the plot correspond to the days when we renewed our purchases. These spikes indicate that the bot services could have access to a large number of devices with different device configurations that result in different browser attributes, and thus, different fingerprints. However, we suspect that they have a fixed set of devices but alter fingerprint attributes to create the illusion of sending requests from a large number of devices.

To provide evidence that bots alter their fingerprint attributes, we inspect the `navigator` object's `platform` attribute on all requests that share the most commonly seen Cookie. Whenever a device sends a request to our honey site, we store a large random number in a first-party Cookie if it had not been set previously. Thus, requests bearing the same value for this Cookie should originate from the same device. Since the `platform` property of the `navigator` object captures information about the type of processor on a given device, it can never change for that device unless the entity controlling the device has intentionally altered the attribute. In Figure 10, we see a wide distribution for the navigator's platform property for the device identified as sending us the largest number of requests with the same Cookie. Differing values for fingerprint attributes that cannot change for a given device constitutes a temporal inconsistency that can be used for bot detection.

Takeaway 6: Bots alter fingerprint attributes to create an illusion of sending requests from a large number of devices. Recording differing values for fingerprint attributes that cannot change for a given device also constitute inconsistencies to detect bots.

7 FP-INCONSISTENT

Our measurements in Section 6 reveal that there exist inconsistencies in different fingerprint attributes for a given request as well as multiple requests from the same device at different points in time. In this section, we present our approach to use these inconsistencies to enhance bot detection. We categorize inconsistencies into two types: spatial and temporal.

Spatial inconsistencies refer to attribute values within a request that conflict or are incompatible with other attribute values in that same request. Examples include differing locations inferred from an IP address and time zone, or implausible combinations, such as an iPhone without touch input support. Our takeaways in Section 6.1 and Section 6.2 show that evasive bots incur significant spatial inconsistencies across information captured in their fingerprint attributes as well as IP addresses.

Temporal inconsistencies are attribute values that are incompatible across different requests from the same user or users. Examples include significantly different time zones for requests from the same IP address and inconsistent device memory values for the same Cookie value. Our takeaway from Section 6.3 shows that evasive bots give rise to significant temporal inconsistencies by changing their attributes.

7.1 Identifying spatial inconsistencies

Our methodology for detecting spatial inconsistencies relies on the understanding that real devices can only possess a limited number of hardware and software configurations. In contrast, bots, in their attempts to mimic real devices and evade detection, as described in Section 6.3, often modify these configurations. However, these alterations typically do not account for every possible source of device information (such as JavaScript APIs, User-Agent, etc.), leading to a proliferation of device configurations. This is especially noticeable in devices such as iPhones or iPads that are commonly owned by real users and have the highest success rate in evading detection (as shown in Section 6.1). Consequently, the increased number of bots pretending to be popular devices results in a greater variety of configurations in the dataset of requests obtained on our honey site.

However, identifying such inconsistencies is challenging because analyzing all possible attribute combinations is infeasible. To facilitate the analysis, we first categorize attributes into different groups based on the type of information each attribute provides. For instance, attributes like `Color Depth`, `Screen Resolution`, and `Touch Support` are grouped because they all convey information about the device's screen. Table 7 in Appendix F shows the various groups used in our analysis, demonstrating how we categorize attributes to streamline the detection of inconsistencies.

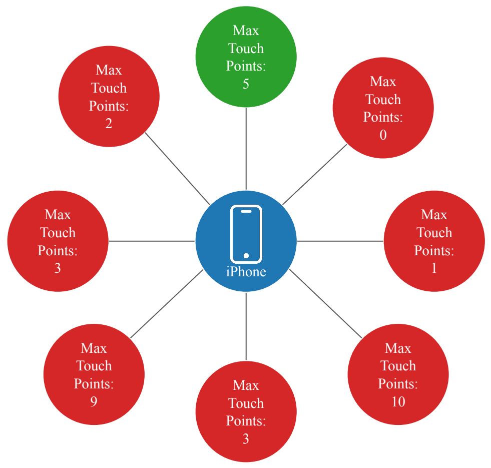


Figure 11: An example of excessive configurations of a device (iPhone) with the fingerprint attribute representing maximum touch points.

Next, we analyze pairs of attributes within each category to identify spatial inconsistencies. For each pair, we rank the attributes based on the number of unique instances recorded in our dataset. For example, in the pair `UA Device` and `Maximum Touch Points`, we sort `UA Device` in descending order by the number of unique `Max Touch Points` values associated with it. A genuine iPhone can only have five simultaneous touch points. However, when bots imitate iPhones but report a different number of touch points, our dataset reveals an implausible number of unique combinations between `UA Device` and `Max Touch Points`. We start with the UA Device instance that has the highest number of unique combinations and identify cases where the combination of these two attributes is impossible. After identifying the inconsistent pair of attribute values, we repeat the process with lower-ranked unique combinations and other attribute pairs. Appendix D defines our algorithm to identify spatial inconsistencies. This algorithm helps us identify the most frequently altered attributes and the spatial inconsistencies they produce. Table 6 in Appendix E provides examples of such inconsistencies in our dataset.

7.2 Identifying temporal inconsistencies

Building upon our findings in Section 6.3, we utilize both the large random number identifier set by our honey sites in each visiting device's browser storage (Cookie) and IP address to identify temporal inconsistencies. First, we use the Cookie identifier to measure variance in immutable device attributes (e.g., number of CPU cores, device memory) across requests containing the same identifier. If an incoming request increases the number of unique attribute values associated with previous identifiers, we consider that request to be temporally inconsistent. For instance, if all previous requests from a device have a `Hardware Concurrency` value of 4 and a new request contains a value of 6, we label that request as temporally inconsistent.

We also use a user's IP address to identify temporal inconsistencies related to time zones and location. If an incoming request increases the number of unique time zones (measured as an offset from UTC) associated with that IP, we classify that request as temporally inconsistent. Similarly, we also identify temporal inconsistencies in location information provided through the IP address and `navigator.geolocation`.

7.3 Improved bot detection

In this section, we describe our methodology to use temporal and spatial inconsistencies to detect bots that evade DataDome and BotD. To measure the improvement in accuracy from spatial inconsistencies, we translate the inconsistencies identified in Table 6 into filter rules. These filter rules are then matched with each request that evaded detection by DataDome or BotD. For temporal inconsistencies, we use the timestamp of each request to determine the order in which requests were made, applying filter rules to identify inconsistencies created by requests arriving later.

The results in Table 4 show that using rules generated through spatial and temporal inconsistency analysis can decrease the evasion of bots against BotD by 44.95% and against DataDome by 48.11%. Table 3 shows the improvement in detection on requests obtained from each individual bot service. We evaluated the generalizability of our methodology by computing filter rules on 80% of the requests obtained on our honey site and evaluating them on the remaining 20%. This evaluation led to a meagre drop in detection accuracy of 0.42% for BotD and 0.23% for DataDome, thereby showing that FP-Inconsistent generalizes to unseen requests.

Our results on the requests received on our honey site show that using a filter list to counter commonly found inconsistencies is an effective method to detect and block evasive bots. Filter lists are commonplace in the anti-tracking community, where they provide a good trade-off between performance and accuracy in detecting advertising and tracking services. Currently, no such alternative exists to detect bots that show inconsistent fingerprints. Our methodology is a first step towards creating such filter lists to enhance online bot detection.

7.4 Real user traffic

We also evaluate FP-Inconsistent's filter rules against traffic from real users to ensure that our improvements in bot detection do not incorrectly detect real users as bots. Concretely, we shared a version of our honey site that contained a unique URL with students at our university. Since we only shared this URL with bonafide students, we have high confidence that requests from real users were recorded at this URL. We did not collect any Personally Identifiable Information (PII) from these users and discuss the ethics of collecting this data in Appendix A.

We report a true negative rate of 96.84% on the 2,206 requests received at this URL. The small number of false positives were likely due to students experimenting with User-Agent spoofers, as these cases triggered spatial inconsistencies involving User-Agents. We could not conduct large-scale evaluation on real user traffic in the wild since it would be challenging to ensure ground-truth. Regardless, our evaluation shows low false positive rates, which can be further mitigated using CAPTCHAs if needed (Section 8.1).

7.5 Privacy-enhancing browsers

Privacy-enhancing browsers such as Brave [7], Tor [60], and Fingerprint Spoofer [21] alter fingerprint attributes to protect user privacy against tracking [29, 33]. In this section, we examine the attributes altered by such technologies and their impact on FP-Inconsistent.

We conducted an experiment where we sent requests to different versions of our honey site (each with a distinct URL) while employing five different privacy-enhancing browsers: Safari, Brave, Tor browsers as well as uBlock Origin and AdBlockPlus browser extensions on Google Chrome. We 300 requests from devices running macOS (M1 MacBook Pro), Linux (Intel Coffee Lake Desktop), iOS (iPad Pro), and Android (Google Pixel 7).

Brave Brave browser currently alters 6 different fingerprint attributes: `audio`, `canvas`, `plugins`, `deviceMemory`, `hardwareConcurrency`, and `screenResolution`. Our inconsistency rules do not incorporate the former three attributes and Brave's alterations to the others were consistent with other attributes. For instance, Brave alters `deviceMemory` on desktops to plausible values (0.5, 1, 2, 4, and 8), which align with the amount of memory in typical desktops and remain consistent with other fingerprint attributes.

However, since Brave browser retains Cookies across requests, the requests triggered several temporal inconsistencies where multiple requests shared the same Cookie but had differing values for both `hardwareConcurrency` and `deviceMemory`. Such inconsistencies are rare in real-world scenarios, as they require users to enable Brave's fingerprint protection while retaining Cookies. These rare false positives

Table 3: Improvement in DataDome and BotD's detection rate on traffic from each bot service when incorporating FP-Inconsistent.

Bot Service	Num. Requests	DataDome Detection Rate	DataDome + FP-Inconsistent Detection Rate	BotD Detection Rate	BotD + FP-Inconsistent Detection Rate
S1	121500	55.99%	83.41%	28.42%	60.26%
S2	63708	57.01%	82.61%	27.71%	55.83%
S3	54746	25.09%	46.31%	89.74%	94.17%
S4	47278	61.35%	82.35%	26.15%	52.09%
S5	40087	76.14%	88.19%	27.35%	50.46%
S6	32447	28.19%	43.7%	94.55%	97.05%
S7	28940	97.44%	99.35%	360.01%	83.91%
S8	26335	19.57%	47.84%	71.1%	86.06%
S9	23412	27.71%	65.69%	80.67%	94.07%
S10	18967	84.23%	94.7%	40.64%	70.43%
S11	17996	93.45%	98.63%	59.36%	80.16%
S12	7010	94.95%	98.36%	48.56%	78.21%
S13	5119	93.04%	99.1%	49.48%	87.04%
S14	4920	16.26%	66.27%	9.92%	67.29%
S15	4219	88.86%	99.6%	0%	77.87%
S16	4174	95.52%	99.69%	99.98%	100%
S17	2999	25.34%	43.88%	92.1%	95.1%
S18	1430	79.3%	99.86%	0%	83.57%
S19	1411	90.08%	99.5%	0%	59.76%
S20	382	2.88%	7.59%	2.88%	7.07%

Table 4: Comparison of the improvement in DataDome and BotD's detection accuracies resulting from different forms of inconsistency analysis.

	DataDome	BotD
None	55.44%	47.07%
Spatial	76.04%	70.33%
Temporal	56.53%	48.09%
Combined	76.88%	70.86%

can be mitigated using CAPTCHAs, with the verification result stored in Cookies (Section 8.1).

Although FP-Inconsistent does not detect requests from Brave browser as bots, we argue that bot services cannot exploit Brave for evasion since they seek to alter attributes that are not supported by Brave. Concretely, we see that Brave only alters 2 attributes that are most commonly altered by evasive bots (Section 5.2) and does not alter other attributes that are of interest to bots such as those pertaining to their device type or geolocation (Section 6). If Brave were to alter more fingerprint attributes in the future, FP-Inconsistent could become prone to more false positives. However, even in such a hypothetical scenario, these false positives can be mitigated using CAPTCHAs (Section 8.1. Moreover, only a small set of users would encounter these CAPTCHAs if Brave's market share continues to remain at 1% [58].

Tor FP-Inconsistent detected all requests from Tor browser as bots since they triggered spatial inconsistencies between the geolocation inferred from their IP address and the `timezone` attribute of their navigator object. While Tor results in false positives, we expect a small set of users to be affected since Tor likely has less than 1% market share [57]. Furthermore, most websites currently block requests from Tor [59], due to the difficulty in distinguishing Tor traffic from bots. To mitigate false positives, we can present users with CAPTCHAs rather than blocking their requests (Section 8.1). We report the detection accuracy of DataDome and BotD on Brave and Tor traffic in Appendix G.

Safari, uBlock Origin, and AdBlockPlus None of these requests were detected as bots. This is because these tools protect privacy by blocking tracking requests rather than altering fingerprint attributes. The two extensions cater to over 80 million users combined on the most widely used browser, Google Chrome [57, 65, 66]. Safari has the second largest market share among web browsers [57]. This shows that FP-Inconsistent can detect bots while having zero impact of all these users.

8 DISCUSSION

8.1 Overcoming false positives

Our evaluation on requests from real users show that FP-Inconsistent incurs low, but non-zero false positive rates (Section 7.4). Our experiments with privacy-enhancing technologies also reveal certain scenarios that could lead to false positives (Section 7.5). In the context of this paper, false positives refer to requests from real users that were incorrectly detected as bots. Challenging users to solve CAPTCHAs rather than blocking them offers a promising solution to mitigate false positives [3, 14]. While effective, CAPTCHAs could potentially frustrate certain users [46, 51]. This frustration can be mitigated by storing the result of a CAPTCHA verification in a Cookie, thereby reducing the frequency at which users are asked to solve CAPTCHAs.

8.2 Improving FP-Inconsistent

Inconsistencies provide a promising avenue for detection as long as there exist at least one pair of attributes that cannot exist in the real world. Accordingly, increasing the number of captured attributes introduces more opportunities for inconsistencies which can be leveraged for detection. In this paper, we confined FP-Inconsistent to only look for inconsistencies among HTTP headers and the attributes captured by FingerprintJS. Incorporating other attributes such as those from CreepJS [1] can further improve FP-Inconsistent.

Researchers have proposed side-channels based on physical device characteristics to uniquely identify devices even among those with identical hardware and software configurations [39, 49, 50, 64]. Such techniques can significantly empower temporal inconsistencies to detect bots. With FP-Inconsistent, we used Cookies to identify requests that originated from the same device. Bots will be able to overcome our temporal inconsistencies by merely deleting their cookies. Bots would not be able to drop unique identifiers that originate from the physical properties of hardware that cannot be modified. However, capturing more attributes as well as capturing persistent identifiers pose threats to privacy.

8.3 Deployment of filter list rules

FP-Inconsistent generates filter lists of inconsistencies to improve bot detection (Section 7.3). The anti-tracking community [2, 25, 32, 34, 45, 54] typically incorporates filter lists on the client side using browser extensions to block the execution of tracking requests and other resources. Similarly, we envision anti-bot services such as DataDome and BotD to include FP-Inconsistent's filter lists as part of their client-side scripts improved for bot detection.

8.4 Limitations

Our results show that FP-Inconsistent's rules improve the detection of evasive bots. Evasive bots will be able to overcome FP-Inconsistent if they evolve to ensure that they can alter fingerprint attributes without introducing any inconsistencies. Incorporating unmodifiable attributes provides a robust solution to enhance FP-Inconsistent, but such attributes also pose threats to privacy.

8.5 Coexistence of Bot Detection and Privacy-Enhancing Technologies

Our evaluation (Section 7.5) shows that FP-Inconsistent can detect bots without impeding most commonly used privacy-enhancing technologies (except Tor). This observation is interesting because many assume that the goals of bot detection and online tracking are identical. They believe that enhancements to bot detection would also bolster online tracking and weaken privacy-enhancing tools.

While bot detection and tracking overlap, tracking is more complex as it seeks to uniquely identify each user. In contrast, bot detection solely seeks to determine if a particular request was generated by a bot. Accordingly, altering any fingerprint attribute enhances privacy by making it harder for trackers to link requests from the same user, even when other attributes remain unchanged. On the other hand, altering any fingerprint attribute does not necessarily help bots with evasion since other attributes can still reveal their presence. This distinction between bot detection and online tracking allows bot detection systems like FP-Inconsistent to coexist with privacy-enhancing technologies.

However, given the overlap, certain enhancements to bot detection such as incorporating more attributes or incorporating unmodifiable attributes can threaten user privacy. Future research focusing on privacy-preserving bot detection such as identifying the intent behind trackers to not block those indulging in bot detection or an in-browser detection mechanism can bridge the gap to potentially address concerns of privacy protection as well as bot detection.

9 CONCLUSION

We find evidence that bots alter fingerprint attributes to evade detection. However, we find evidence that such evasive bots end up introducing inconsistencies among the fingerprint attributes that can be used for more reliable bot detection. We propose FP-Inconsistent, a data-driven, semi-automatic approach to discover inconsistencies in fingerprint attributes for detecting evasive bots in the wild that are able to evade detection by anti-bot services. As the arms race between evasive bots and anti-bot services evolves, it remains to be seen whether bots can alter their fingerprint attributes while avoiding inconsistency. We believe that it would be challenging for bots to do so because a browser fingerprint is a high dimensional feature set with numerous – often subtle – correlations between attributes that are difficult to anticipate and account for when altering fingerprints. Put simply, it is challenging to tell a complex lie while keeping the story always straight. While FP-Inconsistent rule generation approach may need to be evolved to generate rules for other types of consistencies for future generation of bots, we believe the basic principle will stand over time.

REFERENCES

[1] abrahamjuliot. [n. d.]. CreepJS. https://github.com/abrahamjuliot/creepjs.

[2] AdguardTeam. [n. d.]. Adguard Filters. https://github.com/AdguardTeam/AdguardFilters.

[3] Babak Amin Azad, Oleksii Starov, Pierre Laperdrix, and Nick Nikiforakis. 2020. Web Runner 2049: Evaluating Third-Party Anti-bot Services. In *DIMVA 2020 - 17th Conference on Detection of Intrusions and Malware & Vulnerability Assessment*. Lisboa / Virtual, Portugal. https://hal.science/hal-02612454

[4] Hadi Askari, Anshuman Chhabra, Bernhard Clemm von Hohenberg, Michael Heseltine, and Magdalena Wojcieszak. 2024. Incentivizing News Consumption on Social Media Platforms Using Large Language Models and Realistic Bot Accounts. arXiv:2403.13362 [cs.SI]

[5] Dylan Cutler Asuman Senol, Alisha Ukani and Igor Bilogrevic. 2024. The Double Edged Sword: Identifying Authentication Pages and their Fingerprinting Behavior.

[6] Babylon Traffic. [n. d.]. Boost your business visibility with the best Traffic Bot. https://www.babylontraffic.com/.

[7] Brave. [n. d.]. Secure, Fast, & Private Web Browser with Adblocker | Brave. https://brave.com/.

[8] brianhama. [n. d.]. bad-asn-list. https://github.com/brianhama/bad-asn-list/tree/master.

[9] Alberto Cabri, Grażyna Suchacka, Stefano Rovetta, and Francesco Masulli. 2018. Online Web Bot Detection Using a Sequential Classification Approach. In *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*.

[10] Chia-Mei Chen, Sheng-Tzong Cheng, and Ju-Hsien Chou. 2013. Detection of fast-flux domains. *Journal of Advances in Computer Networks* 1, 2 (2013), 148–152.

[11] Elisa Chiapponi, Marc Dacier, Olivier Thonnard, Mohamed Fangar, Mattias Mattsson, and Vincent Rigal. 2022. An industrial perspective on web scraping characteristics and open issues. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks - Supplemental Volume (DSN-S)*. 5–8. https://doi.org/10.1109/DSN-S54099.2022.00012

[12] DataDome. [n. d.]. Bot And Online Fraud Protection Solution. https://datadome.co/.

[13] Vacha Dave, Saikat Guha, and Yin Zhang. 2013. ViceROI: Catching Click-Spam in Search Ad Networks. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security* (Berlin, Germany) *(CCS '13)*. Association for Computing Machinery, New York, NY, USA, 765–776. https://doi.org/10.1145/2508859.2516688

[14] Zainul Abi Din, Hari Venugopalan, Jaime Park, Andy Li, Weisu Yin, HaoHui Mai, Yong Jae Lee, Steven Liu, and Samuel T. King. 2020. Boxer: Preventing fraud by scanning credit cards. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 1571–1588. https://www.usenix.org/conference/usenixsecurity20/presentation/din

[15] Erez Hasson. [n. d.]. Evasive Bots Drive Online Fraud. https://www.imperva.com/blog/evasive-bots-drive-online-fraud-2022-imperva-bad-bot-report/.

[16] Eugene Belinski. [n. d.]. iOS Ref. https://github.com/ebelinski/iosref.

[17] F5 Inc. [n. d.]. Bot Defense. https://docs.cloud.f5.com/docs/how-to/advanced-security/bot-defense.

[18] Shehroze Farooqi, Guillaume Jourjon, Muhammad Ikram, Mohamed Ali Kaafar, Emiliano De Cristofaro, Zubair Shafiq, Arik Friedman, and Fareed Zaffar. 2017. Characterizing key stakeholders in an online black-hat marketplace. In *2017 APWG Symposium on Electronic Crime Research (eCrime)*. IEEE, 17–27.

[19] Fingerprint. [n. d.]. FingerprintJS. https://github.com/fingerprintjs/fingerprintjs.

[20] Fingerprint. [n. d.]. Open-source JavaScript Bot Detection Library. https://fingerprint.com/products/bot-detection/.

[21] Fingerprint Spoofer. [n. d.]. Fingerprint Spoofer. https://chromewebstore.google.com/detail/fingerprint-spoofer/facgnnelgcipeopfbjcajpaibhhdjgcp.

[22] S. Gianvecchio, Z. Wu, M. Xie, and H. Wang. 2009. Battle of Botcraft: Fighting Bots in Online Games with Human Observational Proofs. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*.

[23] S. Gianvecchio, M. Xie, Z. Wu, and H. Wang. 2008. Measurement and Classification of Humans and Bots in Internet Chat. In *Proceedings of the 17th USENIX Symposium on Security*.

[24] Google. [n. d.]. Verifying Googlebot and other Google crawlers. https://developers.google.com/search/docs/crawling-indexing/verifying-googlebot.

[25] gorhill. [n. d.]. uBlock Origin. https://github.com/gorhill/uBlock.

[26] Daniel Goßen, Hugo Jonker, Stefan Karsch, Benjamin Krumnow, and David Roefs. 2021. HLISA: towards a more reliable measurement tool. In *Proceedings of the 21st ACM Internet Measurement Conference* (Virtual Event) *(IMC '21)*. Association for Computing Machinery, New York, NY, USA, 380–389. https://doi.org/10.1145/3487552.3487843

[27] growtoups. [n. d.]. Datacenter ASN Blocking. https://github.com/growtoups/ASN_LIST.

[28] Xin Hu, Matthew Knysz, and Kang G Shin. 2009. RB-Seeker: Auto-detection of Redirection Botnets.. In *NDSS*.

[29] Muhammad Ikram, Hassan Jameel Asghar, Mohamed Ali Kâafar, Balachander Krishnamurthy, and Anirban Mahanti. 2016. Towards Seamless Tracking-Free Web: Improved Detection of Trackers via One-class Learning. *CoRR* abs/1603.06289 (2016). arXiv:1603.06289 http://arxiv.org/abs/1603.06289

[30] Christos Iliou, Theodoros Kostoulas, Theodora Tsikrika, Vasilis Katos, Stefanos Vrochidis, and Yiannis Kompatsiaris. 2019. Towards a Framework for Detecting Advanced Web Bots. In *Proceedings of the 14th International Conference on Availability, Reliability and Security (ARES 2019)*. Association for Computing Machinery, New York, NY, USA.

[31] imperva.com. [n. d.]. 2023 Imperva Bad Bot Report. https://www.imperva.com/resources/resource-library/reports/2024-bad-bot-report/.

[32] Adblock Inc. [n. d.]. Adblock Plus. https://gitlab.com/adblockinc/ext/adblockplus/adblockplus.

[33] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. 2020. Fingerprinting the Fingerprinters: Learning to Detect Browser Fingerprinting Behaviors. *CoRR* abs/2008.04480 (2020). arXiv:2008.04480 https://arxiv.org/abs/2008.04480

[34] Umar Iqbal, Zubair Shafiq, Peter Snyder, Shitong Zhu, Zhiyun Qian, and Benjamin Livshits. 2018. AdGraph: A Machine Learning Approach to Automatic and Effective Adblocking. *CoRR* abs/1805.09155 (2018). arXiv:1805.09155 http://arxiv.org/abs/1805.09155

[35] Mobin Javed, Cormac Herley, Marcus Peinado, and Vern Paxson. 2015. Measurement and analysis of traffic exchange services. In *Proceedings*

of the 2015 Internet Measurement Conference. 1–12.

[36] Jing Jin, Jeff Offutt, Nan Zheng, Feng Mao, Aaron Koehl, and Haining Wang. 2013. Evasive Bots Masquerading as Human Beings on the Web. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, New York, NY, USA, 1–12. https://doi.org/10.1109/DSN.2013.6575366

[37] Jordan Jueckstock, Shaown Sarker, Peter Snyder, Aidan Beggs, Panagiotis Papadopoulos, Matteo Varvello, Benjamin Livshits, and Alexandros Kapravelos. 2021. Towards Realistic and ReproducibleWeb Crawl Measurements *(WWW '21)*. Association for Computing Machinery, New York, NY, USA, 80–91. https://doi.org/10.1145/3442381.3450050

[38] Faezeh Kalantari, Mehrnoosh Zaeifi, Yeganeh Safaei, Marzieh Bitaab, Adam Oest, Gianluca Stringhini, Yan Shoshitaishvili, and Adam Doupé. 2024. Browser Polygraph: Efficient Deployment of Coarse-Grained Browser Fingerprints for Web-Scale Detection of Fraud Browsers. In *Proceedings of the 2024 ACM on Internet Measurement Conference*. https://doi.org/10.1145/3646547.3688455

[39] Tomer Laor, Naif Mehanna, Antonin Durey, Vitaly Dyadyuk, Pierre Laperdrix, Clé mentine Maurice, Yossi Oren, Romain Rouvoy, Walter Rudametkin, and Yuval Yarom. 2022. DRAWN APART : A Device Identification Technique based on Remote GPU Fingerprinting. In *Proceedings 2022 Network and Distributed System Security Symposium*. Internet Society. https://doi.org/10.14722/ndss.2022.24093

[40] Xigao Li, Babak Amin Azad, Amir Rahmati, and Nick Nikiforakis. 2021. Good Bot, Bad Bot: Characterizing Automated Browsing Activity. In *2021 IEEE Symposium on Security and Privacy (SP)*. 1589–1605. https://doi.org/10.1109/SP40001.2021.00079

[41] Zengrui Liu, Prakash Shrestha, and Nitesh Saxena. 2022. Gummy browsers: targeted browser spoofing against state-of-the-art fingerprinting techniques. In *International Conference on Applied Cryptography and Network Security*. Springer, 147–169.

[42] MaxMind. 2024. MaxMind GeoIP Databases. https://www.maxmind.com/en/geoip-databases.

[43] MaxMind. 2024. MaxMind minFraud Services. https://www.maxmind.com/en/solutions/fraud-prevention/overview.

[44] Mozilla. [n. d.]. Date.prototype.getTimezoneOffset(). https://develper.mozilla.org/enUS/docs/Web/JavaScript/Reference/Global_Objects/Date/getTimezoneOffset.

[45] Shaoor Munir, Sandra Siby, Umar Iqbal, Steven Englehardt, Zubair Shafiq, and Carmela Troncoso. 2023. COOKIEGRAPH: Understanding and Detecting First-Party Tracking Cookies. arXiv:2208.12370 [cs.CR]

[46] Yoshimichi Nakatsuka, Ercan Ozturk, Andrew Paverd, and Gene Tsudik. 2021. CACTI: Captcha Avoidance via Client-side TEE Integration. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 2561–2578. https://www.usenix.org/conference/usenixsecurity21/presentation/nakatsuka

[47] Minh Hieu Nguyen Ba, Jacob Bennett, Michael Gallagher, and Suman Bhunia. 2021. A Case Study of Credential Stuffing Attack: Canva Data Breach. In *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*. 735–740. https://doi.org/10.1109/CSCI54926.2021.00187

[48] OpenWPM. [n. d.]. A web privacy measurement framework. https://github.com/openwpm/OpenWPM.

[49] Iskander Sanchez-Rola, Igor Santos, and Davide Balzarotti. 2018. Clock Around the Clock: Time-Based Device Fingerprinting. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 1502–1514. https://doi.org/10.1145/3243734.3243796

[50] Andre Schaller, Wenjie Xiong, Nikolaos Athanasios Anagnostopoulos, Muhammad Umair Saleem, Sebastian Gabmeyer, Stefan Katzenbeisser, and Jakub Szefer. 2017. Intrinsic Rowhammer PUFs: Leveraging the Rowhammer effect for improved security. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE. https://doi.org/10.1109/hst.2017.7951729

[51] Andrew Searles, Yoshimichi Nakatsuka, Ercan Ozturk, Andrew Paverd, Gene Tsudik, and Ai Enkoji. 2023. An Empirical Study & Evaluation of Modern CAPTCHAs. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 3081–3097. https://www.usenix.org/conference/usenixsecurity23/presentation/searles

[52] seoclerks. [n. d.]. SEO Marketplace for backlinks, web design, website traffic, and online marketing. https://www.seoclerks.com/.

[53] SHapley Additive exPlanations. [n. d.]. Welcome to the SHAP documentation. https://shap.readthedocs.io/en/latest/.

[54] Sandra Siby, Umar Iqbal, Steven Englehardt, Zubair Shafiq, and Carmela Troncoso. 2022. WebGraph: Capturing Advertising and Tracking Information Flows for Robust Blocking. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 2875–2892. https://www.usenix.org/conference/usenixsecurity22/presentation/siby

[55] Spark Traffic. [n. d.]. Comprehensive Marketing Suite for better SEO ranking. https://www.sparktraffic.com/.

[56] Kevin Springborn and Paul Barford. 2013. Impression Fraud in Online Advertising via Pay-Per-View Networks. In *22nd USENIX Security Symposium (USENIX Security 13)*. USENIX Association, Washington, D.C., 211–226. https://www.usenix.org/conference/usenixsecurity13/technical-sessions/paper/springborn

[57] StatCounter. 2024. Browser Market Share Worldwide. https://gs.statcounter.com/browser-market-share.

[58] TechReport. [n. d.]. Most Important Brave Market Share Statistics in 2024. https://techreport.com/statistics/software-web/brave-market-share-statistics/.

[59] Tor. [n. d.]. A website I am trying to reach is blocking access over Tor. https://support.torproject.org/tbb/website-blocking-tor/.

[60] Tor. [n. d.]. You have a right to BROWSE without being watched. https://www.torproject.org/download/languages/.

[61] U.S. Department of Health and Human Services. 2018. Decision Charts: 2018 Requirements (Common Rule). https://www.hhs.gov/ohrp/regulations-and-policy/decision-charts-2018/index.html#c1

[62] Antoine Vastel, Pierre Laperdrix, Walter Rudametkin, and Romain Rouvoy. 2018. FP-STALKER: Tracking Browser Fingerprint Evolutions. In *2018 IEEE Symposium on Security and Privacy (SP)*. 728–741. https://doi.org/10.1109/SP.2018.00008

[63] Antoine Vastel, Walter Rudametkin, Romain Rouvoy, and Xavier Blanc. 2020. FP-Crawlers: Studying the Resilience of Browser Fingerprinting to Block Crawlers. In *MADWeb'20 - NDSS Workshop on Measurements, Attacks, and Defenses for the Web*, Oleksii Starov, Alexandros Kapravelos, and Nick Nikiforakis (Eds.). San Diego, United States. https://doi.org/10.14722/ndss.2020.23xxx

[64] Hari Venugopalan, Kaustav Goswami, Zainul Abi Din, Jason Lowe-Power, Samuel T. King, and Zubair Shafiq. 2023. Centauri: Practical Rowhammer Fingerprinting. arXiv:2307.00143 [cs.CR]

[65] Chrome Webstore. 2024. Adblock Plus. https://chromewebstore.google.com/detail/adblock-plus-free-ad-bloc/cfhdojbkjhnklbpkdaibdccddilifddb.

[66] Chrome Webstore. 2024. uBlock Origin. https://chromewebstore.google.com/detail/ublock-origin/cjpalhdlnbpafiamejdnhcphjbkeiagm.

[67] Shujiang Wu, Pengfei Sun, Yao Zhao, and Yinzhi Cao. 2023. Him of Many Faces: Characterizing Billion-scale Adversarial and Benign Browser Fingerprints on Commercial Websites. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society.

[68] XGBoost. [n. d.]. XGBoost Documentation. https://xgboost.readthedocs.io/en/stable/.

[69] Kazuki Yasuhara, Naoki Kodama, and Takamichi Saito. 2024. Challenges in Web Bot Detection and Detection Evasion Technologies. In *Advances in Network-Based Information Systems*, Leonard Barolli (Ed.). Springer Nature Switzerland, Cham, 162–173.

A ETHICS

This study complies with ethical guidelines for research involving data collection and usage. To conduct the research, we paid a small amount to bot operators to generate requests directed solely to our honey site. To ensure data quality and realism, we prioritized bot services with high ratings and traffic advertised as realistic and organic. These requests were analyzed exclusively for research purposes, with the goal of improving bot detection.

The research process was reviewed and approved by our university, ensuring alignment with ethical principles outlined in both the Belmont Report and the Menlo Report. To determine whether Institutional Review Board (IRB) approval was necessary, we consulted official guidelines from the Human Subject Regulations Decision Charts [61], specifically the section addressing activities covered by 45 CFR Part 46. Based on this evaluation, we determined that our research does not involve human subjects as defined by 45 CFR Part 46 and, as a result, qualifies for exemption from IRB oversight.

Furthermore, our study did not collect or store any Personally Identifiable Information (PII), nor did it involve the identification or tracking of individual users across different websites/contexts. Traffic data was analyzed in aggregate, and identifiable information, such as IP addresses, was hashed before storage.

All purchased traffic was directed exclusively towards our honey site, ensuring that no other sites or users were impacted. The primary purpose of this research was to advance the science of bot detection, and we refrained from monetizing the honey site or deriving any profit from the generated traffic.

B COMPARISON OF APIS USED BY BOTD AND DATADOME

Table 5 shows the different APIs accessed by BotD and DataDome scripts on our honey site.

Table 5: Comparison of browser APIs read by DataDome and BotD

Browser API	DataDome	BotD
Display		
window.screen.colorDepth	✓	✗
HTMLCanvasElement.getContext	✓	✓
Navigator		
window.navigator.webdriver	✓	✓
window.navigator.vendor	✓	✓
window.navigator.userAgent	✓	✓
window.navigator.serviceWorker	✓	✗
window.navigator.productSub	✗	✓
window.navigator.plugins	✓	✓
window.navigator.platform	✓	✗
window.navigator.permissions	✓	✓
window.navigator.oscpu	✓	✗
window.navigator.mimeTypes	✓	✓
window.navigator.mediaDevices	✓	✗
window.navigator.maxTouchPoints	✓	✗
window.navigator.languages	✓	✓
window.navigator.language	✓	✓
window.navigator.hardwareConcurrency	✓	✗
window.navigator.buildID	✓	✗
window.navigator.appVersion	✗	✓
window.navigator.__proto__	✓	✗
Storage		
window.sessionStorage	✓	✗
window.localStorage	✓	✗
window.document.cookie	✓	✗
Mouse Movements		
MouseEvent.type	✓	✗
MouseEvent.timeStamp	✓	✗
MouseEvent.clientY	✓	✗
MouseEvent.clientX	✓	✗
addEventListner: mouseup	✓	✗
addEventListner: mousemove	✓	✗
addEventListner: mousedown	✓	✗
Miscellaneous		
addEventListner: asyncChallengeFinished	✓	✗
addEventListner: pagehide	✓	✗
Performance.now	✓	✗

C COMBINATION OF FINGERPRINT ATTRIBUTES TO EVADE DATADOME

We visualized the XGBoost decision tree for DataDome described in Section 5.2. The tree with a depth of 5 indicated that all 44,168 requests having a Screen Frame value less than 20 that do not support the Chrome PDF Viewer plugin, having memory over 256 MB with less than 14 CPU cores having the width of Monospace font used in FingerprintJS larger than 131.5 were able to evade detection.

D ALGORITHM TO IDENTIFY SPATIAL INCONSISTENCIES

Algorithm 1 describes our algorithm to identify spatial inconsistencies.

Table 6: Inconsistencies Identified

Attribute Group	Attributes	Examples
Screen	(UA Device, Screen Resolution)	(iPhone, 1920x1080) (iPhone, 847x476) (iPad, 900x1600) (Samsung SM-S906N, 1920x1080) (M2006C3MG, 800x360) (Mac, 656x1364)
	(UA Device, Touch Support)	(iPhone, None) (Mac, touchEvent/touchStart) (Samsung SM-A127F, None) (M2004J19C, None) (Infinix X652B, None)
	(UA Device, Max Touch Points)	(iPhone, 1) (iPhone, 0) (iPad, 1) (iPad, 7) (Mac, 10) (Samsung SM-A515F, 0) (Pixel 7 Pro, 0)
	(UA Device, Color Depth)	(iPhone, 16) (iPad, 16)
	(UA Device, Color Gamut)	(Samsung Galaxy Tab S7, (p3, rec2020)) (SAM Galaxy S10 Smartphone, (p3, rec2020))
Device	(UA Device, Device Memory)	(XiaoMi Mi Pad4 LTE, 8) (Samsung SM-T387W, 4) (MiPad 3, 8) (Samsung SM-A515F, 1) (XiaoMi Redmi Go, 8)
	(UA Device, Hardware Concurrency)	(iPhone, 3) (iPhone, 32) (Mac, 48) (iPad, 32) (XiaoMi Mi Pad5 Wi-Fi, 1) (Pixel 2, 32)
Browser	(UA Browser, UA OS)	(Safari, Linux) (Samsung Internet, Linux) (MiuiBrowser, Linux) (Safari, Windows)
	(UA Browser, Vendor)	(Mobile Safari, Google Inc.) (Chrome Mobile, Apple Computer, Inc.)
	(UA Browser, Platform)	(Mobile Safari, Linux x86_64) (Chrome Mobile, Win32) (Chrome Mobile, Linux x86_64) (Chrome Mobile iOS, Win32)
Location	(IP Location, Time Zone)	(France/Hauts-de-France, America/Los Angeles) (Germany/Sachsen, America/Los Angeles) (Singapore/Singapore, America/Los Angeles) (United States of America/California, Asia/Shanghai) (United States of America/Virginia, Pacific/Auckland)
Browser	(Platform, Vendor)	(Linux armv5tejl, Apple Computer, Inc) (Linux aarch64, Apple Computer, Inc.) (Linux armv6l, Apple Computer, Inc.) (Win32, Apple Computer, Inc.)

Algorithm 1 Algorithm to Detect Spatial Inconsistencies

```
1: Input: Attribute categories $F$, Dataset containing requests $D$, Labels for requests $L$ (true if the request is from a bot, false for human)
2: for all $f \in F$ do
3:     for all attribute pairs $\{f_a, f_b\} \subseteq f$ do
4:         Filter $D$ where $L$ is false, creating $D'$
5:         Create tuples $(v_{f_a}, nv_{f_b})$, where $v_{f_a}$ is the value of $f_a$ and $nv_{f_b}$ is the number of unique values of $f_b$ found in the same row as $v_{f_a}$ in $D'$
6:         Sort the tuples in increasing order of $nv_{f_b}$
7:         for all $(v_{f_a}, nv_{f_b})$ in the sorted order do
8:             if the combination is inconsistent then
9:                 Label all rows in $D$ containing $(v_{f_a}, v_{f_b})$ as true
10:            end if
11:        end for
12:    end for
13: end for
```

E INCONSISTENCIES IDENTIFIED

Table 6 lists some examples of the inconsistencies that we identified for each attribute group in Table 7.

F ATTRIBUTE CATEGORIES FOR INCONSISTENCY ANALYSIS

Table 7 list different categories of attributes used for inconsistency analysis.

G DATADOME AND BOTD ON BRAVE AND TOR TRAFFIC

Roughly after the first 10 requests on each device, DataDome starts detecting all requests from Brave as bots resulting in a false positive rate of 41% on the 300 requests described in Section 7.5. BotD on the other hand does not detect any requests as bots.

Similar to FP-Inconsistent, DataDome detects all requests from Tor browser as bots while BotD does not detect any requests as bots. This further sheds light on the difficulty in distinguishing between Tor and bot traffic.

Table 7: Attribute Categories

Category	Attributes
Screen	UA Device, Color Depth, Screen Resolution, Touch Support, Max Touch Points, HDR, Contrast, Reduced Motion
Device	UA Device, Device Memory, Hardware Concurrency, UA OS
Browser	UA Browser, Plugins, Platform, UA OS, UA Vendor, Vendor, Vendor Flavors
Location	IP Location, Timezone, Languages