How gullible are web measurement tools? A case study analysing and strengthening OpenWPM's reliability

Benjamin Krumnow
TH Köln,
Open University Netherlands
Germany

Hugo Jonker
Open University Netherlands,
Radboud University
Netherlands

Stefan Karsch
TH Köln
Germany

ABSTRACT

Automated browsers are widely used to study the web at scale. Their premise is that they measure what regular browsers would encounter on the web. In practice, deviations due to detection of automation have been found. To what extent automated browsers can be improved to reduce such deviations has so far not been investigated in detail. In this paper, we investigate this for a specific web automation framework: OpenWPM, a popular research framework specifically designed to study web privacy. We analyse (1) detectability of OpenWPM, (2) resilience of OpenWPM's data recording, and (3) prevalence of OpenWPM detection.

Our analysis (1) reveals OpenWPM is easily detectable. Our investigation of OpenWPM's data recording integrity (2) identifies novel evasion techniques and previously unknown attacks against OpenWPM's instrumentation. We investigate and develop mitigations to address the identified issues. Finally, in a scan of 100,000 sites (3), we observe that OpenWPM is commonly detected (~14% of front pages). Moreover, we discover integrated routines in scripts specifically to detect OpenWPM clients. In conclusion, our case study shows that even the most popular web measurement framework, OpenWPM, is more gullible than expected, and this gullibility is rarely accounted for in studies.

CCS CONCEPTS

• **Security and privacy → Browser security**; • **Information systems → World Wide Web**; • **General and reference → Measurement**.

KEYWORDS

Web bots, bot detection, web measurements, reliability, security, privacy

ACM Reference Format:
Benjamin Krumnow, Hugo Jonker, and Stefan Karsch. 2022. How gullible are web measurement tools? A case study analysing and strengthening OpenWPM's reliability. In *The 18th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '22), December 6–9, 2022, Roma, Italy*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3555050.3569131

CoNEXT '22, December 6–9, 2022, Roma, Italy

ACM ISBN 978-1-4503-9508-3/22/12…$15.00
https://doi.org/10.1145/3555050.3569131

1 INTRODUCTION

Web studies use web measurement tools (typically built on top of browser automation frameworks) to accrue data over thousands of sites. The goal of such studies is to analyse what regular visitors would experience on the web. This relies on an (often unstated) assumption that the data as collected is representative of what a regular, human-controlled browser would encounter. Previous works [45, 47, 101] have shown that this is not always the case: websites have been found to omit content (advertisements, video, JavaScript execution, login forms, etc.) or require completion of a CAPTCHA for automated clients. This casts doubts on the validity of such analyses, especially since this effect is often not accounted for (Sec. 2). In some cases, studies implement their own automation tooling [15, 50] to bypass detectors targeting such frameworks. While such an approach can be technically sufficient, the additional steps may also shift or even exacerbate the detection problem [37]. Another typical approach is to use third-party plugins that hide differences of the fingerprint (e.g., [47]). Again, this could be a reasonable approach, but 100% fidelity should not be assumed – though in practice, it often is. This raises the question: how gullible are frameworks as measurement tools? That is, to what extent can web measurement tools be fooled by websites?

To analyse a website, a web measurement tool visits the site and collects whatever data it needs for its analysis. In our view, there are two key issues that affect whether websites can fool the tool. First: detectability. Detectability enables a web site to deliver innocuous contents to analysers instead of what regular visitors would receive (a so-called cloaking attack, see e.g., [42]). Second, resilience of data collection, that is: able to collect the sought-after data even in adverse conditions. Websites may employ obfuscation and other tricks to thwart analysis, e.g., sprinkling random breakpoints throughout the code to hinder analysis [63]. A malicious website would use tricks specifically targeting analysers to hide its wrongdoings.

We investigate the extent to which these two issues affect reliability of web measurement tools by means of a case study. Many web measurement tools are one-off creations, used to perform a specific analysis, but not designed for use cases beyond that (e.g., [9, 38, 45, 52]). Amongst the few more general web measurement frameworks, most have not gained traction in the community and have been used little beyond their initial study so far (e.g., FPDetective [3], Crawlium [60], VisibleV8 [46]). In contrast, OpenWPM, a framework [29] for measuring web privacy, has been used over 70 peer-reviewed publications (Sec. 2). Studies based on OpenWPM keep appearing frequently in top conferences (Tbl. 15). Its maturity, as well as its popularity and impact in the web measurement community, make it an ideal subject for our case study.

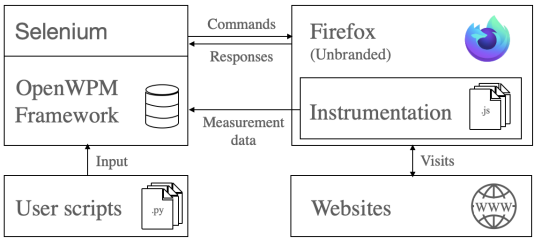

Figure 1: Components of the OpenWPM framework

OpenWPM offers increased stability, fidelity and easy access to measurement functionality on top of Selenium + WebDriver (a browser automation framework). The framework can be run under either Ubuntu or macOS. It consists of four parts (Fig. 1): a web client, automation components, instrumentation for measurements, and a framework. As a web client, OpenWPM uses an unbranded Firefox browser. In contrast to a regular Firefox browser, this allows running unsigned browser extensions. There are various measurement instruments implemented via one browser extension. Each facilitates recording a specific aspect, such as JavaScript calls, cookies, or HTTP traffic. Last is the framework, which acts as a maestro orchestrating work. Its purpose is to control browsers and data collection. It also adds various functionalities, such as monitoring for browser crashes and liveliness, restoring after failures, loading input data, etc.

Main contributions.

- **(Sec. 3)** We provide the first analysis of OpenWPM's detectability based on both conventional fingerprinting [45] and template attacks [75] techniques. We find previously not reported, identifiable properties for every mode of running OpenWPM (headless, Xvfb, etc.), even allowing to distinguish between these modes.
- **(Sec. 5)** We explore how sites can attack OpenWPM's data collection. We find various attack vectors targeting OpenWPM's most commonly used instruments and implement proof-of-concept attacks for these.
- **(Sec. 6)** We harden OpenWPM against poisoning attacks and detection. This hides all identifiable properties when run in regular mode and addresses the identified attacks against OpenWPM's instrumentation. We evaluate its performance against vanilla OpenWPM. The number of cookies received is severely impacted. Conversely, ads/tracker traffic is hardly impacted.
- **(Sec. 4)** We look for bot detectors in the Tranco Top 100K sites via both static and dynamic analysis. We find a drastic increase of Selenium-based bot detection compared to earlier studies. In addition, we find detectors in the wild specifically targeting OpenWPM.

Ethics & responsible disclosure. Our work aims to make OpenWPM a more reliable measurement framework. We responsibly disclosed our findings and shared fixes of the identified issues. This helps make OpenWPM less detectable, and therefore its results more reliable. Of course, a less detectable web bot may itself be abused. For attacking specific sites, our improvements do not greatly impact the attack surface: a less detectable OpenWPM is a fine tool for studying the web, but not for a targeted attack on a specific site. For attacks that span thousands of sites (e.g., clickfarming), our improvements do not help: disguising as a regular browser is insufficient to overcome contemporary defenses. For that, site-specific fingerprints are needed [86]. Thus, existing re-identification-based countermeasures (e.g., rate limiting) are not impacted.

Availability. Our stealth extension and our collected data set (see Sec. 6.3) are available from https://bkrumnow.github.io/openwpm-reliability.

Table 1: Measurement characteristics in 72 peer-reviewed studies that are built upon OpenWPM

Category	Studies
Measures	
– HTTP traffic	56
– cookies	35
– JavaScript	22
– other	6
Run mode	
– unspecified	59
– virtualisation	16
– headless	7
– regular mode	3
– Docker	2
– Xvfb	2

Category	Studies
Interaction	
– no interaction	55
– clicking	11
– scrolling	8
– typing	5
Subpages	
– not visited	53
– visited	19
Bot detection	
– ignored	55
– discussed	17
∘ uses mitigation	8

2 USE OF OPENWPM IN PREVIOUS STUDIES

To understand how OpenWPM is being used, we review the different studies performed to date with OpenWPM. In June 2022, 76 works, of which 57 peer-reviewed, were listed[1] as using OpenWPM. We further add 15 recent studies that had not yet been listed. For each study, we check the following: what is measured, whether subpages are visited, whether interaction is used, and what run mode is used. Tbl. 1 summarises our findings.

The *measures* category tallies how many studies used OpenWPM's various measurement instruments: HTTP traffic, cookies, and JavaScript. Each of these measures may be impacted individually due to bot detection. Interestingly, while most studies use OpenWPM to record HTTP traffic, a few (e.g. [19, 27, 55, 81]) have used it as automation instead as a measurement tool. These are tallied under 'other' in Tbl. 1. The other categories pertain to aspects that may impact detectability. In each case, it is currently not known whether these play a role in bot detection. With respect to the *interaction* category, we note that no study mentioned implementing interaction mechanisms. Therefore, we assume all studies used OpenWPM's default interaction functionality.

With respect to the *run mode* category, note that not all studies provide information about this. Nevertheless, the used run mode may impact detectability (e.g. [39]) and thus should be considered. We therefore consider all currently supported modes:

a. *unspecified*: does not specify mode,
b. *regular*: uses a full Firefox browsers,

[1] https://webtap.princeton.edu/software/

c. *headless*: uses Firefox without a GUI,
d. *Xvfb*: as regular, with visual output redirected to a buffer,
e. *Docker*: runs OpenWPM within a Docker container,
f. *Virtualisation*: uses virtual machines, possibly in cloud infrastructure.

Lastly, we track whether the studies considered *bot detection* at all and, if so, whether they used OpenWPM's built-in anti-detection features. Aside from studies investigating bot detection directly, only very few consider fingerprinting [87] or cloaking [14, 53] as a potential risk for valid results.

3 FINGERPRINT SURFACE OF OPENWPM

We begin by addressing the research question *how can OpenWPM be distinguished from human-controlled web clients?* In general, a website operator looking to identify OpenWPM clients can either probe for identifiable properties (i.e., fingerprinting), or attempt to recognise OpenWPM's interaction. The latter is due to Selenium, whose interaction was studied in detail by Goßen et al. [37]. Those results fully carry over to OpenWPM. This leaves uncertainty about how OpenWPM's fingerprint distinguishes it from other clients and other bots. In line with previous works, we call that part of a browser fingerprint that distinguish a certain type of client from other types the *fingerprint surface* [86]. Determining the fingerprint surface of an OpenWPM client requires a way to find its properties that deviate from properties and values in other clients. Jonker et al. [45] showed that it suffices to consider differences within the client's 'browser family', that is, fingerprint differences with those clients who use the same rendering engine and JavaScript engine. By comparing the results for multiple clients of the same browser family, differences unique to each client are brought to light. In previous works, two approaches for browser fingerprinting were used: probing a specific list of properties [45], or using an automated approach for DOM traversion [75]. While there is overlap between the results of these methods, neither offers a complete superset of the other. We combine the results of both approaches to determine the fingerprint surface.

Limitations. Fingerprint-based bot detection requires identifiable properties of bots, such as deviations from regular user clients or inconsistencies. While we use state-of-the-art tooling to identify outstanding characteristics in OpenWPM at the HTTP layer and above, we cannot guarantee that all properties or methods are covered. Furthermore, our method compares the fingerprint of OpenWPM to the fingerprint of a regular (equivalent version) Firefox. Any differences cannot be due to the browser then. However, this by itself does not guarantee that these differences are unique when compared to other browsers. To reduce the likelihood of this, we validate the found fingerprint surface against a number of other web browsers (see Sec. 3.3).

3.1 RQ1: How recognisable is OpenWPM?

We determine OpenWPM's fingerprint surface by comparing its client to a standalone version of the same Firefox browser. Any differences must originate in the hosting environment, the framework itself, the base implementation, the added automation, or measurement components. Note that it is already well-known that

Table 2: Summary of deviating properties of each OpenWPM setup (v.017.0) contrasted with OpenWPM's Firefox (v.90)

	macOS 10.15		**Ubuntu 18.04**			**Docker 19.03.6**
	RM	HM	RM	HM	Xvfb	RM
navigator.webdriver is true	✓	✓	✓	✓	✓	✓
screen dimension prop.	✓	✓	✓	✓	✓	✓
screen position prop.	✓	✓	✓	✓	✓	✓
font enumeration	–	–	–	–	–	✓
timezone is 0	–	–	–	–	–	✓
navigator.languages prop.	–	43	–	43	–	–
deviating WebGL prop.	–	2037	–	2061	18	27
With instrumentation:						
- through tampering	+253	+253	+252	+252	+252	+252
- added custom functions	+1	+1	+1	+1	+1	+1

RM: Regular mode; HM: Headless mode; Xvfb: X virtual frame buffer mode.

Table 3: Screen properties for various configurations

OS	**Mode**	**Resolution**	**Window**	**X**	**Y**	**Offset (x,y)**
macOS	Regular	2560 x 1440	1366 x 683	23	4	0, 0
	Headless	1366 x 768	1366 x 683	4	4	0, 0
Ubuntu	Regular	2560 x 1440	1366 x 683	80	35	8, 8
	Headless	1366 x 768	1366 x 683	0	0	0, 0
	Xvfb	1366 x 768	1366 x 683	0	0	0, 0
	Docker	2560 x 1440	1366 x 683	0	0	0, 0

OpenWPM's underlying automation component, Selenium, is trivially recognisable by the `navigator.webdriver` property,[2] which is not addressed by OpenWPM. We are looking for further distinguishing aspects. To account for possible effects of the various run modes of OpenWPM on the fingerprint surface, we determine variations for each setup on Ubuntu and macOS. Tbl. 2 summarises the identifying properties found. In addition to ways to recognise OpenWPM's instrumentation, we also identify ways to recognise display-less scraping (headless or Xvfb mode), and virtualised run mode. Thus, every mode of running OpenWPM is identifiable as a web bot.

3.1.1 Recognisable via screen resolution, window position. We found two new identification measures that work against all modes of OpenWPM: screen resolution, set by OpenWPM, and window position, set by the browser automation framework. OpenWPM screen properties use standard values and cannot be changed (see Tbl. 3). On macOS, all browser instances will use the same absolute coordinates; on Ubuntu, each window shifts by the same offset when using regular mode.

3.1.2 Suppressing output → more identifiable. Suppressing output to display (by using Xvfb, headless, or Docker) adds a significant number of differences. In headless mode, the lack of a WebGL implementation leads to thousands of missing properties. We also observe that this mode adds 43 new properties to the `navigator.language` object. Xvfb mode uses a regular Firefox browser, which contains WebGL functionality. Nevertheless, Xvfb mode causes 5 changed and 13 missing properties. Interestingly, both headless and Xvfb mode allow the detection of missing user elements by accessing

[2] https://www.w3.org/TR/webdriver/#x4-interface

Table 4: Selected deviations, Ubuntu no-display modes

Mode	WebGL vendors	avail{Top|Left}
RM	*AMD AMD TAHITI*	27, 72
HM	*Null*	0, 0
Xvfb	*Mesa/X.org llvmpipe (LLVM 12.0.0,…)*	0, 0
Docker	*VMware, Inc. llvmpipe (LLVM 10.0.0,…)*	27, 72

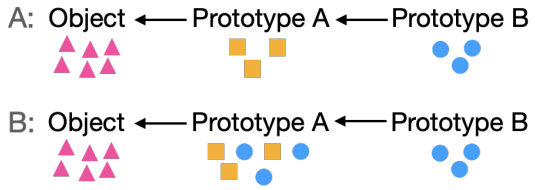

Figure 2: Properties in a (A) original object or (B) by the instrumentation polluted object

the property *screen.availTop*. This describes the first y-coordinate that does not belong to the user interface.[3] In display-less modes, this is always zero, while regular browsers have larger values.

3.1.3 Virtualisation leaves identifiable traces. Using OpenWPM's docker container causes the WebGL vendor property to contain the term `VMware, Inc.` (Tbl. 4) – clear evidence for the use of virtualisation [94]. In addition, the Docker environment reduces the number of available JavaScript fonts to one (Bitstream Vera Sans Mono), nor does it provide information about the time zone.

3.1.4 Using JS instrumentation has large effect on detectability. We checked if using any of OpenWPM's various instruments has any effect on its fingerprint surface. The only differences occur when using the JavaScript instrument. First, this instrument overwrites certain of the browser's standard JavaScript objects, which can be detected by using the `toString` function of a function or object (see Listing 1). Another identifying aspect of this instrument is the presence of a function in the window object (`window.getInstrumentJS`), which is not a part of the ECMA specification,[4] nor present in any common desktop browser (Firefox, Safari, Chrome, Edge, Opera). Third, OpenWPM's wrapper functions can be found in stack traces. For that, a script need to provoke an error in any overwritten function and catch the stack trace to successfully identify a modification by OpenWPM. Lastly, the JavaScript instrument 'pollutes' prototypes along the prototype chain of an object. Instrumenting is done by changing the prototype of an object, as well as all its ancestor prototypes. However, the properties of later ancestor prototypes are all added to the first ancestor prototype (see Fig. 2). This distinguishes a visitor with instrumentation from one without.

3.2 RQ2: How stable is the fingerprint surface?

We explored how stable our determined fingerprint surface is, as new Firefox and OpenWPM versions may appear frequently. To that end, we repeated our experiments for older version of OpenWPM (0.11.0 and 0.10.0). In general, we found that the fingerprint surfaces largely overlap. For example, on MacOS, the number of WebGL deviations in headless mode increases to 2037 in OpenWPM 0.17.0, from OpenWPM 0.11.0's 2022. In the oldest OpenWPM version (0.10.0), we find that the JavaScript instrument adds two properties instead of one to the window object (`jsInstruments` and `instrumentFingerprintingApis`). In addition, we also investigated whether using an unbranded browser (as OpenWPM does) impacts OpenWPM's fingerprint. We did not find differences between branded and unbranded versions.

Using outdated browsers, however, does impact the fingerprint. For example, Google's reCAPTCHA service assigns a higher risk to older browser variants [78]. In the past, OpenWPM's integrated Firefox version has been lagging behind the official release of Firefox several times (Tbl. 14 in Appx. C). We found that OpenWPM used an outdated version of Firefox 69% of the time.

3.3 Validation of the fingerprint surface

Our measurement of OpenWPM's fingerprint surface should be validated to ensure our methodology did not introduce measurement artefacts. Moreover, our method only contrasts OpenWPM to Firefox; other browser fingerprints could contain elements that are present in our measured fingerprint surface. To validate the fingerprint surface, we test its distinctiveness from other consumer browsers and platforms. We implemented an OpenWPM detector that uses four test strategies for the entire measured fingerprint surface to identify OpenWPM amongst web clients:

(1) test for presence of a DOM property
(2) test for absence of a DOM property
(3) test if a native function was overwritten
(4) compare a DOM property with an expected value

We tested the detector by setting up four machines, two Macintoshes and two PCs with Ubuntu. On each machine, we used OpenWPM and common browsers (Chrome, Safari, Opera and Firefox). We tested all distinguishing property from Tbl. 2. Our detector site was able to correctly identify OpenWPM every single time. Except for a few WebGL- and screen-related properties, all properties uniquely identify OpenWPM. As reported in Sec. 3.1, screen properties differ per operating system. For regular modes, the screen resolution depends on the system setup (display size and selected resolution). For WebGL properties, we found that these also occur on some non-OpenWPM clients (roughly 200 of 4K properties). After removing these, the fingerprint surface still contained a sizeable number of identifying WebGL and screen properties.

4 INCIDENCE OF OPENWPM DETECTION

To assess the extent of OpenWPM detection in the wild, we conduct a large-scale measurement for client-side bot detection. In detail, we focus on scripts with capabilities to detect OpenWPM, i.e. scripts with routines to access properties unique for Selenium-based bots and/or OpenWPM. We find both general Selenium detectors and OpenWPM-specific detectors.

4.1 Data acquisition and classification

4.1.1 Methodology. Previous automated approaches [45, 46] to identify bot detectors have either relied on static or dynamic analysis. The idea behind static analysis is to identify code patterns in

[3] https://developer.mozilla.org/en-US/docs/Web/API/Screen/availTop
[4] https://262.ecma-international.org/5.1/

```
window.canvas.getContext.toString();
// output of .toString when not instrumented
"function getContext() {
  [native code]
}"

// output of .toString when instrumented
"function () {
  const callContext = \
  getOriginatingScriptContext(!!logSettings.logCallStack);
  logCall(objectName + "." + \
  methodName, arguments, callContext, logSettings);
  return func.apply(this, arguments);
}"
```

Listing 1: Detectability of OpenWPM's JavaScript instrumentation

source code that link to known bot detectors or that use specific bot-related properties. A limitation is that scripts may create code dynamically, which will be missed out by static analysis. Moreover, minification and obfuscation further increase the false negative rate of static analysis. The alternative approach, dynamic analysis, is to monitor JavaScript calls that identify a script as bot detector based on access to bot-related properties. Dynamic analysis does cover dynamically-generated scripts. Moreover, it does not monitor the code itself, but only executed calls. An upside of this is that neither minification nor obfuscation affects the analysis. On the other hand, code that happens not to be executed during the run, is not analysed. Both static and dynamic analysis have been able to identify some bot detectors in the wild. It is not clear whether and to what extent the results of the methods differ in practice for finding web bot detectors. We combine both methods to increase coverage.

4.1.2 Setup. In order to assess the extent of client-side bot detection, we scan the top 100K websites of the Tranco list [51][5]. We set up an instance of OpenWPM running Firefox in regular mode. During a site visit, our OpenWPM client stores a copy of any transmitted JavaScript file and records JavaScript calls. We wait an additional 60 seconds after every completed page load[6] to give websites time to perform JavaScript operations. In addition, our client measures the presence of bot detection on subpages by opening a maximum of three URLs extracted from a site's landing page. For selecting subpages, we consider only URLs linking to the same domain. We use the `eTLD+1` scheme to identify domains. To account for websites that use same-origin requests to redirect clients to foreign domains, our client checks if a foreign domain was entered after following all redirects.

Accessing OpenWPM's fingerprint surface is cause to consider a script as a bot detector. However, certain scripts may access fingerprint surface attributes for other purposes, such as checking supported WebGL functionality. To reduce such false positives, we only classify a script as bot-detecting when it accesses properties pertaining to browser automation or are unique to OpenWPM (see Sec. 3.1). This leaves only the following: `navigator.webdriver`, which is specific to WebDriver-controlled bots; and the new identifying properties introduced by OpenWPM's JavaScript instrumentation: `getInstrumentJS`, `instrumentFingerprintingApis`, and `jsInstruments`. Tbl. 5 shows the results of the data collection and classification.

4.1.3 Limitations. Inherent in the above approach are several assumptions that can impact the results. First, our approach relies on the fingerprint surface we established. Detectors based on other methods (e.g., mouse tracking [17]) will be missed. Second, we do not account for cross-site tracking. A third-party tracker could classify our client as a bot on one site and would need only to re-identify the client on another site, e.g., using IP filtering or regular browser fingerprinting. This amounts to a form of *website cloaking* – serving different content to specific clients. To what extent third-party tracking in general employs cloaking is a different study and left to future work. Both these limitations may cause underestimation of the number of detectors (false negatives). As such, our approach approximates a lower bound on the number of detectors in the wild.

Preprocessing for static analysis. Within the static analysis, we pre-process scripts to undo straightforward obfuscation. We derive the respective encoding, transform hex literals to ASCII characters, and remove code comments. We apply our static analysis to scripts that we collected during our scan of the Tranco Top 100K, which resulted in 1,535,306 unique scripts. To identify Selenium-detector scripts, we then use patterns to look for access to `navigator.webdriver` (more details can be found in Appx. B).

Using honey properties to catch iterators. For the dynamic analysis, every recorded access to the fingerprint surface identifies a script with the potential to detect OpenWPM as a bot. This will also be triggered by scripts that iterate over all properties, e.g., for regular browser fingerprinting. Determining the purpose of such iteration requires per-script manual inspection and goes beyond dynamic analysis.

To determine whether property iteration takes place, we extend our client's navigator and window object with 'honey'[7] properties. These honey properties are added on the fly and use random strings as name. Hence, only a script using property iteration would access all honey properties. We assign scripts that use property iteration into two categories, based on access to the `navigator.webdriver` property: definitely detecting bots, and inconclusive. Iterator scripts are classified as inconclusive if they do not access `navigator.webdriver`, as all accesses to the fingerprint surface could be due to property iteration. Scripts that iterate the navigator object will naturally access the `webdriver` property. To check whether this access is only by iteration or intentional, we distinguish between scripts that trigger our static analysis and those that do not. Only scripts that do not surface in the static analysis are classified as inconclusive.

4.2 RQ3: How often is OpenWPM detected?

OpenWPM can be detected directly, via OpenWPM-specific properties, or indirectly, via properties of its underlying components (Selenium, WebDriver, etc.). Our results show that, when checking

[5] https://tranco-list.eu/list/WV79
[6] As determined by the *document.readyState* property.
[7] We are not aware of any previous works using such an approach.

Table 5: Number of websites with Selenium detectors

# sites	static	dynamic	union
identified	32,694	19,139	38,264
without false positives / 'inconclusive'	15,838	16,762	18,714

Table 6: Sites with scripts probing OpenWPM-specific properties

	cz	gs	google.com	ad1t
total	**331**	**14**	**9**	**2**
jsInstruments	331	5	2	2
instrumentFingerprintingApis	0	6	4	0
getInstrumentJS	0	3	3	0

cz: cheqzone.com, gs: googlesyndication.com, ad1t: adzouk1tag.com

both front- and subpages, at least 16.7% of websites in the Tranco Top 100K execute scripts that accessed properties specific to Selenium. Moreover, we also find 4 actors serving scripts that access OpenWPM-specific properties.

4.2.1 356 sites detect OpenWPM-specific properties. Most scripts we found recognise OpenWPM by targeting Selenium. A small number of detectors also include specific routines to detect OpenWPM itself. Overall, 356 sites executed scripts that accessed OpenWPM-specific properties. These scripts were all included via third-party domains, belonging to four distinct providers. Tbl. 6 summarises these detectors and their detection method. Detectors on cheqzone.com were found by both static and dynamic analysis; detectors on the other three domains used some form of minification, obfuscation, and/or dynamic loading, and were only found by dynamic analysis. We investigated the four hosting domains by consulting `whois` records, EasyList,[8] and the WhoTracksMe database [18]. All domains are related to the advertising industry. The domain `cheqzone.com` belongs to CHEQ, a company fighting ad fraud. The scripts hosted by Google domains are included through Google's reCAPTCHA service. While we could not clarify the origin of `adzouk1tag.com`, we found this domain listed in the EasyList for ad domains.

4.2.2 14% of sites detect bots on the front page. Fig. 4 depicts the distribution for detectors active on the front page of websites for static and dynamic analysis. Dynamic analysis without considering property iteration identifies 12,208 sites with detectors on the front page. Static analysis measures the number of sites where bot detection could be triggered (11,897), including those where detection is present but not (yet) executed, e.g., where detection is only triggered after hovering over certain elements. While both static and dynamic analysis identify a similar number of detectors for each bucket, they do not fully overlap. Combining both provides a slight increase in the presence of detectors (~1.7K sites).

4.2.3 Deep scanning increases rate of detection by 5%-points. As discussed in Sec. 2, 26% of studies conducted with OpenWPM (also) investigated subpages. This raises the question whether such studies are more often subject to bot detection, that is: does bot detection occur more frequently on subpages? Fig. 3 depicts the occurrence of bot detectors on front pages and subpages. In general, studies examining subpages are at greater risk to be detected: the number of sites with active detectors increases for by at least 37%. Hence, the average detection rate within the Top 100K sites will increase. That is: the study will be exposed to more detectors. Combining the results of both measurements, we see an increase of 5 per cent points (from 14% to 19%).

[8] https://easylist.to/easylist/easylist.txt

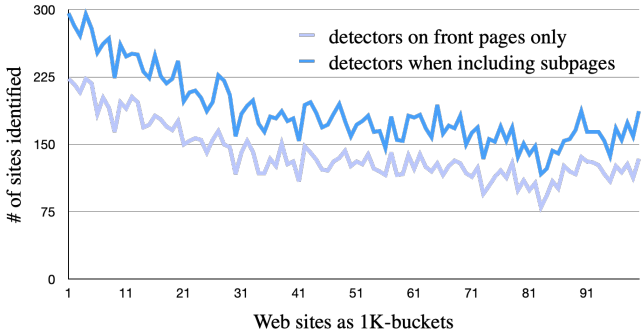

Figure 3: Number of sites with bot detectors on front- and subpages (depicted per 1K sites)

4.3 RQ4: Who employs bot detection?

To explore this question, we separated detectors into first and third parties. We find that the majority of sites includes detectors from third-party domains. We count how often scripts on these third-party domains are included on scanned sites, tallying each third-party domain once per including site. Some sites include more than one detector, hence the total number of inclusions exceeds the number of sites with detectors. Overall, we count 3,867 first-party detector scripts and 21,325 third-party detector scripts.

We explore what sites include detectors. For the identified 16K websites with detectors, we collect categories using Symantec's site review service (https://sitereview.norton.com/). Sites may be assigned multiple categories; for such sites, we tally each listed category. Fig. 5 depicts the 16 most often tallied categories for both first-party detectors (4,198 times) and third-party detectors (16,323 times). News sites are responsible for 18.4% of all third-party inclusions, followed by Technology (9%) and Business (7%). Interestingly, the ranks for Shopping (16.4%) and News (5%) switch for first-party detector inclusions. Moreover, sites in the categories Finance (8% vs 3%) and Travel (7% vs 2%) make up for a larger portion in the set of first-party inclusions than for third parties.

4.3.1 Third-party bot detection typically serves the advertisement industry. Following up on the previous point, we investigated the origins of third party detectors. Tbl. 7 breaks down the most common included domains. The top 10 domains account for two third of inclusions. The site WhoTracks.me [18] categorises trackers according to purpose. Using this, we find that the bot-detecting scripts on the most commonly included domains can serve a variety of purposes. For example, yandex.ru offers scripts used for advertising, content delivery network, site analytics, social media, and others. Other uses include web analytics (crazyegg.com), CDN (jsdelivr.net)

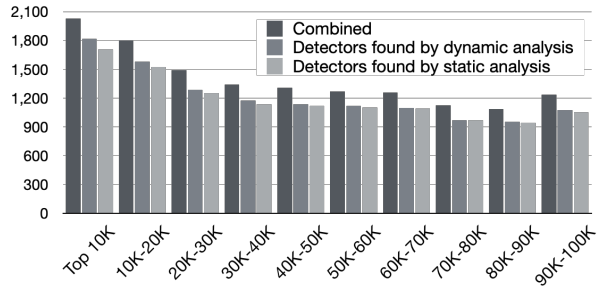

Figure 4: Detectors found on front pages

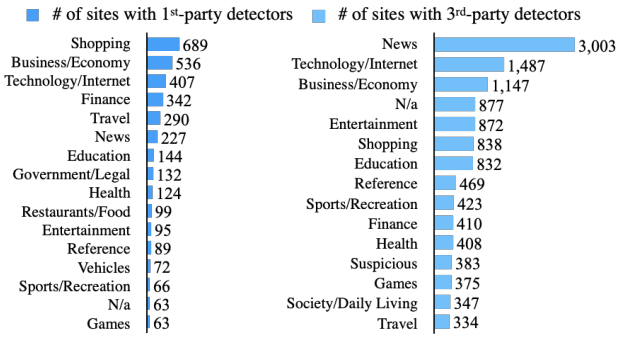

Figure 5: Common categories of sites with detectors

Table 7: Domains hosting 3^{rd}-party detector scripts

	hosting domain	# inclusions (1/site)	%
	all	*21,325*	*100%*
1	yandex.ru	3,848	18.04%
2	adsafeprotected.com	2,309	10.83%
3	moatads.com	2,165	10.15%
4	webgains.io	2,091	9.81%
5	crazyegg.com	1,552	7.28%
6	intercomcdn.com	1,061	4.98%
7	teads.tv	854	4.00%
8	jsdelivr.net	423	1.98%
9	mxcdn.net	416	1.95%
10	mgid.com	402	1.89%
11+	*remaining 704 domains*	*6,204*	*29.1%*

and live chat (intercomcdn.com). However, bot detection is most commonly deployed by advertisers (e.g., domains 2,3,4,7,9, and 10 in Tbl. 7).

4.3.2 The vast majority of first-party detectors are embedded third parties. To determine the origins of first-party bot detection scripts, we look for similarities between their inclusions of detectors. To do so, we hash the scripts and check for structural similarities in script URLs (for more details see Appx. A). We found various similarities amongst unrelated sites. Scripts originating from Akamai occur the most frequent (1,004 sites). Second is Incapsula (998 sites), third is an unknown bot detector (659 sites), and fourth is Cloudflare (486 sites). Together, these top four originators account for 3,147 out of 3,867 sites (88%) where we found first-party detectors. In contrast to the purpose of third-party detectors, first-party detectors are not supplied by advertisement companies. Moreover, Akamai, Incapsula and Cloudflare all offer commercial bot detection services. With that in mind, one should expect sites with first-party detectors to likely tailor their responses for detected bots (e.g., throttling, blocking, withholding resources, and serving CAPTCHAs).

5 ATTACKING JAVASCRIPT RECORDING

We have found detectors specifically targeting OpenWPM. This raises the question to what extent a malicious site could harm an OpenWPM study. We investigate whether a malicious website or third party could corrupt OpenWPM's data collection process. In particular, we consider an attacker that can deliver arbitrary content (HTML, cookies, JavaScript), but cannot break the browser's security model. To do so, our focus resides on attacks against the integrity or completeness of measurements. More specifically, we aim to attack the resilience of OpenWPM's most commonly used instruments: HTTP traffic, cookie recording, and JavaScript call recording. Both HTTP and cookie instruments are simple wrappers around browser functionality. Breaking them thus requires breaking the browser, which is outside the attacker model. The JavaScript instrument, on the other hand, needs to supply all its monitoring functionality itself. It is therefore clearly in scope of our attacker model.

Since the instruments focus on data recording, we investigate attacks on data recording. More specifically, we consider:

1. whether data recording may be prevented;
2. whether fake data can be injected into the data recorder;
3. whether already recorded data can be deleted or altered;
4. finally, whether the data recording is complete.

Instruments in OpenWPM are implemented as a browser extension. Extensions are isolated to protect higher privilege APIs from access by untrusted code. Website scripts thus cannot directly interact with extensions. However, both extensions and website scripts can read and change the DOM, opening the door for injection attacks against extensions that read the DOM. We conducted source code analysis for each instrument under investigation to identify vulnerabilities to such attacks. Below we discuss the found vulnerabilities.

Limitations. As we focus on data recording, the scope of our evaluation is limited to OpenWPM's instruments. Vulnerabilities could also be introduced by other OpenWPM components (see Sec. 1). Furthermore, we used manual code analysis; automated code analysis, such as code scanners or fuzzers, may give more results. To detect false positives, we validate the findings of the code analysis by implementing proof-of-concept attacks.

5.1 RQ5: How to prevent recording?

We found two ways to prevent OpenWPM from recording JavaScript: first, disrupting communication to the data recorder; secondly, CSP stopping JavaScript injections.

5.1.1 By disrupting communication. We found a vulnerability that allows a website to turn off recording of JavaScript calls in the JavaScript instrument. In more detail, the JavaScript instrument overwrites several API functions which use the event dispatcher to send messages when called. The event dispatcher then notifies

the JavaScript instrument's back end to record the corresponding API call. To prevent an attacker from silently undoing these hooks, OpenWPM also hooks into (and thus: records access to) setters and getters to these API functions themselves. However, the event dispatcher itself is not protected. Thus, we can alter the event dispatcher to inject our own messages and manipulate messages sent to OpenWPM (Listing. 2). To carry out this attack, the attacker overrides the event dispatcher to block all messages (all events from instrumented objects). This would already block OpenWPM recording, by breaking any JavaScript API calls. However, this also would break a website's own JavaScript. To block only OpenWPM messages, the block needs to be tailored. Conveniently, tags messages with an ID to identify any monitored objects. Though this ID is randomly generated, it can easily be determined: simply trigger an API call to a monitored object, acquire the random ID from the observed message, and update the event dispatcher to only block messages containing this ID.

```
//Step I: Retrieve OpenWPM's random ID
function grabID() { return new Promise((resolve, reject) => {
  let id;
  document.dispatchEvent = function (event) {
    id = event.type; document.dispatchEvent = dispatch_fn;
    if (id !== undefined) { resolve(id);
    } else { reject(new Error(msg));}
  }
  // Perform an action to grab the ID
  navigator.userAgent;});}
// Step II: Overwrite event dispatcher to block events
async function attackExtension() {
  let id = await grabID();
  document.dispatchEvent = (event) => {
    if (event.type != id) { dispatch_fn(event); // Dispatch event
    } else {console.log("Event swallowed: " + event);}}}
```

Listing 2: Turn off the script recorder

5.1.2 By CSP stopping script injections. As stated above, OpenWPM overrides builtin browser functions. To do so, OpenWPM injects a script to the DOM to enter page context, then calls that script to override the APIs, and finally, it removes the script from the DOM to hide what happened. This happens before the website's JavaScript code is loaded. However, websites may use Content Security Policy (CSP) to shield their users from XSS and data injection attacks. We found that the CSP `script-src` directive can be used to prevent OpenWPM's code injection. This will lead to CSP violation in the JavaScript instrument, leaving the API functions un-instrumented.

5.2 RQ6: Can fake data be injected?

Altering the event dispatcher not only allows an attacker to block data recording, it also allows an attacker to learn the ID OpenWPM uses to record data. This is sufficient to inject almost arbitrary messages to be recorded. The attacker simply creates a custom event following the format used by OpenWPM's JavaScript extension and includes OpenWPM's assigned event ID. This enables an attacker to define most of the content of the resulting entry in OpenWPM's recording, such as the executing script URL or which function was called. Crucially, though, the website that originated the call is set outside of the browser by OpenWPM. The data sent by the event dispatcher is properly sanitized by the back-end, which prevents spoofing this. We can thus only inject fake data for the currently visited website. Note that a third party included on the site can also execute this attack.

5.3 RQ7: Can records be deleted or altered?

Whereas the previous attacks exploited a vulnerability in the DOM-parsing front-end of the respective instruments, deleting already recorded data requires manipulating the instrument's back-end: SQLite. Attacking a database back-end requires an SQL injection vulnerability. We found that the current OpenWPM's data recording back-end (OpenWPM v0.20.0) properly sanitizes its inputs; we deem this sufficient and did not investigate further.

5.4 RQ8: Is data recording complete?

We evaluated whether data recording is complete or whether there are unobserved channels. We found two different attacks against completeness: a bypass of the JavaScript instrument's recording, and silent delivery of JavaScript code.

5.4.1 JS instrument's recording can be bypassed. We found a way to bypass OpenWPM's recording of JavaScript function calls. This attack again exploits OpenWPM's hooks to record function calls. In particular, the hooks must be attached to every object that is to be observed. For every new window or iframe, this must be done afresh. However, there is a long-standing bug in Chrome and Firefox [80], where both browsers under some circumstances fail to inject scripts into iframes. We tested if OpenWPM's implementation is affect by this and we found that this is indeed the case.

Our evaluation of this attack involves two different ways to access an iframe's DOM[9] to create/execute iframes and their code: static vs. dynamic creation and immediate vs. delayed execution. Of these, immediate code execution (at creation time) is required to successfully exploit this bug. None of the other parameters we tested influenced the result. Listing 3 shows a proof-of-concept of this type of attack.

```
setTimeout(() => {
  let element = document.querySelector("#unobserved");
  let iframe = document.createElement('iframe');
  // HTML code for instantiating an iFrame
  iframe.src = "unobserved-iframe.html";
  element.appendChild(iframe);
  iframe.contentWindow.navigator.userAgent;
}, 500);
```

Listing 3: Example of an unobserved channel

5.4.2 Silent delivery of JavaScript code. The HTTP instrument either stores all response bodies (full coverage), or it can be set to store JavaScript files only. The latter option significantly reduces stored content. The HTTP instrument should thus ensure recording of the aforementioned JavaScript attacks, unless this instrument's recording can also be bypassed. We managed to achieve this with an HTTP instrument only recording JavaScript files. For this mode, an attacker can silently deliver JavaScript code by sending it as text and, on the client-side, convert it to code and execute it (Appx. D).

[9] `window.frames[0]`, and `frame.contentWindow`

To successfully bypass OpenWPM's traffic recording of JS files, three aspects must be accounted for:

i. the content-type attribute must be set to something other than text/javascript;
ii. the src attribute must not contain a ".js" extension;
iii. the delivered file is not automatically executed; this must be handled by a different client-side script (e.g., using `eval()`).

6 IMPROVING OPENWPM RELIABILITY

This section focuses on OpenWPM's reliability as an instrument measuring the web as encountered by regular visitors. We explore how and to what extent reliability can be improved. To do so, we design an approach to hardening OpenWPM's instrumentation and to hiding its distinctive fingerprint (from here on referred to as WPM_{hide}). Our proof-of-concept successfully hides the telltale signs of OpenWPM from its fingerprint and makes OpenWPM robust in the face of the discussed attacks in a lab setting. To evaluate its effectiveness in an open world setting, we run WPM_{hide} against detectors in the wild and contrast its measurements with those of a regular OpenWPM client.

6.1 RQ9: How to hide the fingerprint?

OpenWPM's characteristic fingerprint varies with the various modes of running OpenWPM. For example, in headless Firefox mode, the fingerprint surface is difficult to hide due to headless mode's lack of functionality when compared to regular browsers. Hence, we focus on run modes where OpenWPM runs the browsers natively (Regular Mode). For such modes, we achieve stealth by overriding properties without leaving traces. These techniques can also be applied in other run modes (e.g., virtualisation).

The identifying properties for Regular Mode (see Tbl. 2) relate to the `webdriver` property, window position, and dimension. Of OpenWPM's various instruments, only the JavaScript instrument causes further identifiable properties. Hiding these properties can be achieved by a customized browser, or by including additional code inside a page's scope. Implementing the former requires significant work, but it can hide the fingerprint near-perfectly. The latter approach is far simpler to implement but risks leaving residual traces. For our proof-of-concept, we chose the second option, as it can be seamlessly integrated within the current OpenWPM framework without significant effort.

To prevent detection, our proof-of-concept addresses all five identifiability issues (Sec. 3.1): (1) the `toString` operation of overwritten functions must return the regular (unchanged) output string; (2) no additional property may appear in the DOM; (3) stack traces must not show any signs of the instrumentation; (4) prototype pollution must be avoided; (5) prevent detection of automation components.

6.1.1 Preserve `toString` *output.* For the first issue, we found that CanvasBlocker[10] addresses this well. Its implementation successfully fools all our fingerprinting tests (Sec. 3). CanvasBlocker creates a getter function with an identical signature to the function that must be overwritten and attaches it to the DOM based on a specific Firefox feature called `exportFunction`. The newly exported function is then used to redefine the getter of a object's prototype for a specific property. As a result, the overwritten function returns the native code string like a default browser property (Listing 1). Normally, accessing the getter of an object's prototype leads to an error. If this getter is replaced with a custom getter, that error is never thrown. This makes tampering with properties via an object's prototype detectable [37]. Calling the original getter from the customised getter results in the original error being thrown, addressing this aspect of the fingerprint surface.

6.1.2 Preserve clean DOM. The second issue arises during page load, prior to the page's JavaScript activation. The instrumentation injects its code as script from the content context into the page context, overwrites the needed properties, and removes its code from the page context again. However, in practice, not all injected functions are deleted. We update the instrument to overwrite all functionality directly from the content context, thus keeping the page context clean.

6.1.3 Fake stack traces. The third issue requires the stack trace to show no signs of instrumented functions. A web page can only access stack traces if errors occur. Normally, if an error occurs, the stack trace would show that the called function is called from inside the instrumentation. We address this by catching each error and throwing a new error with properly adjusted values for file name, column, message, and line number.

6.1.4 Avoid prototype pollution. The fourth issue relates to the pollution of an object's prototype. OpenWPM's instrument modifies only the first prototype in the prototype chain, not others further in the chain. We mitigate this by overwriting properties per prototype. Nevertheless, this mitigation has a limitation: it is not possible to determine the caller of a function when a prototype has multiple descendants. This means our mitigation may inadvertently instrument more objects than intended. For prototypes located higher up the chain, the number of children increases; exacerbating this problem. Thankfully, most of the APIs that OpenWPM instruments by default are provided by prototypes close to the bottom of their prototype chain.

6.1.5 Prevent detection of automation components. The automation components are detectable by window size, window position and the webdriver attribute. For the latter, our hidden version must set the `navigator.webdriver` property to false like a regular Firefox browser. Since Firefox version 88, this flag is not user-settable.[11] We override the getter function of the `navigator.webdriver` property to return the default value (false) in the same fashion as described in the previous section. To change OpenWPM default window settings, we introduce a settings file that makes the window size and position settable in OpenWPM.

6.2 RQ10: How to mitigate recording attacks?

6.2.1 Secure messaging from page context to background context (see Sec. 5.1 and 5.2). A key benefit from migrating to Firefox's `exportFunction`, as described in the previous section, is the ability to export higher privileged browser functions into the page. Hence, we can port functionality to the page context that is otherwise only available for content or background scripts of a browser extension.

[10] https://github.com/kkapsner/CanvasBlocker

[11] https://bugzilla.mozilla.org/show_bug.cgi?id=1632821

We use this to secure our instrumented functions, as we now can use the `browser.runtime` API to pass messages from the page to the background context. It is crucial that such functionality is exported to a private scope of an overwritten function to prevent access by other scripts in the page context. In addition, injecting functionality this way does not violate the CSP `script-src` directive. Thus, using the export function prevents the 'restriction of script injections', 'turn recording off' and 'inject fake data' attacks.

6.2.2 Intercepting DOM-modifying APIs (see Sec. 5.4). To address the tested variants of incomplete recordings, we use CanvasBlocker's frame protection. The basic idea is to intercept APIs used by page scripts to modify the DOM or create a new, non-instrumented copy of the DOM. This ensures that each modification or newly constructed DOM contains the instrumentation. Our implementation covers five cases: window constructors, DOM modification API, window mutations, DOM creation via the `document.write` API, and finally the `window.open` API.

6.2.3 Filtered HTTP recording not robust (see Sec. 5.4). To the best of our knowledge, there is no known way to distinguish JavaScript code from text that is robust against a dedicated obfuscator. Therefore, an active adversary should be assumed to be capable of hiding JavaScript in a way that would accidentally be filtered out. Since this issue only arises in the presence of active adversaries, we recommend in such a case not to use any filtering.

6.3 RQ11: Does hardening impact measurements?

We developed a proof-of-concept implementation to hide the telltale signs of automation and to mitigate the found attacks. We evaluate the impact of our proof-of-concept implementation (from here on, WPM_{hide}) on web measurements when encountering bot detection in the wild. To that end, we contrast its results with vanilla OpenWPM (from here on: *WPM*) in HTTP traffic, cookies, JavaScript execution, and delivered JavaScript files. We test on all sites with bot detectors as found by the analysis in Sec. 4. This list contains 1,487 sites with detectors. On these sites, we run *WPM* and WPM_{hide} in parallel (OpenWPM v.0.18.0, Firefox v.100, regular mode, HTTP, JavaScript, and cookie instrument activated) and configure each browser to idle 60 seconds on a page after loading completed. We use the latest version of Firefox on both machines; detection based on outdated browser versions thus does not apply to our evaluation (see Sec. 3.2). We take steps to mitigate noise in measurements. In particular, we avoid cross-client interference by separating both crawlers via two individual machines and IP addresses. These residential IP addresses are both located in the same country, which avoids differences caused by geo-location and cloud-based IP blocking [42]. We synchronise visits between both machines to further reduce differences. Lastly, there is a risk that one-off events or single actors alter the measurements. To prevent the former, we repeat our measurement three times (r_1, r_2, and r_3). This allows us to check whether an effect persists or is only temporary. To address the latter, we test for significance. As the data sets are not normally distributed, we use the Wilcoxon signed-rank test with a confidence interval of 95%. In general, our findings show that *WPM* encounters less privacy-invasive behaviour than WPM_{hide}. We executed all three runs of our experiment one after another between the 20^{th} and 21^{st} of June 2022.

Table 8: Comparison of HTTP request resource types

	r_1			r_2	r_3
Resource type	***WPM***	***WPM$_{hide}$***	**Diff.**	**Diff.**	**Diff.**
csp_report	784	188	-76.02%	-74.19%	-70.79%
media	530	610	+15.09%	-15.75%	-14.24%
beacon	5,951	6,622	+11.28%	+8.09%	+11.98%
websocket	321	302	-5.92%	-6.29%	-3.63%
xmlhttprequest	58,867	6,1702	+4.82%	+3.21%	+7.52%
imageset	5,730	5,982	+4.40%	+3.89%	+12.04%
font	9,608	9,356	-2.62%	-0.87%	-1.23%
object	50	49	-2.00%	0.00%	+6.38%
main_frame	3,955	3,883	-1.82%	-1.45%	-0.84%
image	116,296	118,068	+1.52%	+5.86%	+5.65%
script	83,239	84,385	+1.38%	+1.85%	+2.11%
sub_frame	15,393	15,592	+1.29%	+2.81%	+4.86%
other	95	96	+1.05%	-6.32%	+6.67%
stylesheet	9,943	10,028	+0.85%	+1.39%	+2.11%
Total	**310,737**	**316,673**	**+1.91%**	**+3.37%**	**+5.32%**

Table 9: HTTP requests to ad/tracker resources

	EasyList		EasyPrivacy	
	WPM	*WPM$_{hide}$*	*WPM*	*WPM$_{hide}$*
r_1	43,238	+1.64%	39,063	-1.64
r_2	41,659	+5.64%	37,710	+5.37
r_3	41,418	+5.81%	34,402	+7.85

6.3.1 OpenWPM-induced CSP violations eliminated. We find that using WPM_{hide} results in a higher traffic volume, which increases with each run (see 'total' in Tbl. 8). In order to find where this difference originates, we group requests by their resource type.[12] Tbl. 8 breaks the differences down per resource type, showing results for data set r_1; proportions are similar for the other data sets. Most interesting are the changes in CSP reports. We see much less CSP reports for WPM_{hide}. This is intended: this version does not inject nodes into the DOM. This is also highly relevant, as CSP adoption and the directive for content restriction is on the rise [70]. We checked whether any of the remaining CSP reports was due to WPM_{hide}, none were. Note that the *WPM* column offers insights into how often *WPM* fails to install its hooks. In the worst case out of our three data sets, *WPM* failed to do so on 113 of 1,487 sites.

6.3.2 More ad/tracker HTTP-traffic. To assess the amount of trackers and advertisers in traffic, we use the same approach as previous works [5, 15, 47]: use the EasyList and EasyPrivacy blocklists[13] to identify trackers. We find that around a quarter of all HTTP traffic falls into this category. We further see that both machines encounter a significant difference in traffic by advertisers and trackers (p-value < 0.0001). In most cases, this is a significant increase (~5%), though r_1 is an outlier in this regard (see Tbl. 9).

6.3.3 Significantly more tracking cookies. For cookies, we contrasted the number of cookies between both variants per site. We

[12] https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/webRequest/ResourceType
[13] https://easylist.to/

Table 10: Served cookies and differences with WPM_{hide}

	# first-party cookies		# third-party cookies		# tracking cookies	
	WPM	WPM_{hide}	*WPM*	WPM_{hide}	*WPM*	WPM_{hide}
r_1	28,826	+3.33%	31,335	+5.05%	3,031	+41.70%
r_2	28,841	+3.06%	30,977	+7.12%	2,929	+52.13%
r_3	28,744	+4.23%	29,692	+8.11%	2,719	+59.65%

find that the number of served cookies differs significantly for many sites, both for first parties as well as third parties (p-value < 0.0001). As shown in Tbl. 10, *WPM* receives less cookies, with the effect increasing each repetition (possibly due to *WPM* being re-identified). We see a similar effect in the number of sites that serve an unequal number of cookies to both machines: in r_1, 353 sites serve more cookies to WPM_{hide} vs. 156 sites serving more cookies to *WPM*; this difference increases in r_3 to 394 sites for WPM_{hide} vs. 134 sites for *WPM*.

We investigated whether the difference in cookies was due to tracking cookies. To determine whether a cookie can be used for web tracking, we use the approach of Englehardt et al. [30], as refined by Chen et al. [16]. According to this method, a cookie may be used for tracking when: (1) it cannot be a session cookie, (2) the length of the cookie is 8 or more characters (excluding surrounding quotes), (3) the cookie is always set, (4) the cookie is "long-living" (at least 3 months), and (5) the values differ significantly based on the Ratcliff-Obershelp algorithm [11] among all runs. In data set r_1, 3,031 cookies satisfy these criteria for *WPM*, while 4,295 cookies for WPM_{hide} match; a strong increase of 41.70%. This effect is again amplified in the other two runs.

6.3.4 Up to 37%-points more JS calls caught. As discussed in Sec. 5.4, some access methods are not covered by *WPM*'s instrumentation. For WPM_{hide}, we track all accessing API calls, to track the total volume of API calls. Fig. 6 depicts this for JavaScript APIs call in r_1 for WPM_{hide}. Blue depicts the portion of calls covered by *WPM*, the rest is not. Some properties were almost fully covered (`Screen.top`, 99% coverage), others not. Most prominently not covered is the `Screen.availLeft` API, where *WPM* only 63% of calls that WPM_{hide} catches.

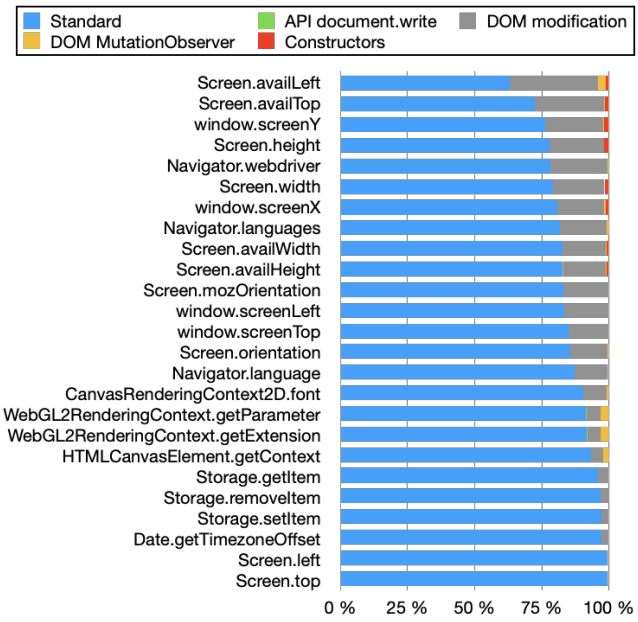

Figure 6: API calls in the context of DOM creation

7 RELATED WORK

Determining the fingerprint surface of web bots. The idea of using fingerprinting to identify certain client components (such as automation frameworks) has gained more attention recently. Vastel [93] and Shekyan [76] conducted manual investigations of headless browsers to pin down identifiable properties in these frameworks. Jonker et al. [45] automated the search for identifiable properties by using a browser fingerprinting library. They compared properties of regular browsers against properties of bots that belong to the same engine class. In contrast, Schwarz et al. [75] applied a new form of fingerprinting (JavaScript template attacks) to perform client-side vulnerability scanning. For a template creation, they traverse object hierarchy and store characteristics of each object. Later on, templates can be compared to determine the difference. Our work comprises the approaches by Jonker et al. and Schwarz et al. to explore the fingerprint surface of OpenWPM. We apply these systematically to the various run modes of OpenWPM clients, uncovering distinguishers for each mode.

Reliability of scraping results. Multiple studies have explored differences between various automated clients, and also between automated clients and human-driven clients in website responses. Ahmad et al. [5] contrasted response differences between HTTP engine tools, headless browsers, and automated browsers. They found that while HTTP engine tools miss many resources, they more often pass bot detection than the other two classes. Jueckstock et al. [47] studied differences between headless and regular Chrome. For regular Chrome, they used a puppeteer plugin which hides distinguishable properties to focus on bot detection. Their results reinforce previous findings [29, 93] to not use headless browsers. Zeber et al. [101] contrast data from human users with OpenWPM clients. In their study, OpenWPM clients encountered three times more tracking domains and had more interaction with third-party domains than human-controlled browsers. Cassel et al. [15] investigate the reliability of emulated browsers. To avoid bot detection, they created their own browser remote control. Interestingly, their observations show the opposite of Zeber et al.'s findings. They observed 84% less third-party traffic for a Selenium-driven vs. a non-Selenium-driven Firefox browser. This contradiction shows that there is yet no consistent picture for the influence of bot detection on measurements. Further investigation to resolve this conundrum is needed.

Attacking bots. Investigations into attacking bots predominantly focus on cloaking [42, 96–98, 102]. To the best of our knowledge, other attack vectors have not been extensively studied, though Jueckstock et al. [46] argue limited reliability of JavaScript instruments. They discuss several limitations based on JavaScript code

found in the wild. Our work overcomes these limitations for OpenWPM's instruments.

Measuring bot detection in the wild. Two studies exist that carried out a large-scale investigation of the existence of unknown fingerprint-based bot detectors. Jonker et al. [45] scanned 1M websites gathering statically included scripts and analysing these using static code analysis. Shortly after, Jueckstock and Kapravelos [46] presented a similar experiment using dynamic script collection and dynamic analysis. Their presented tool relies on a modified V8 engine to instrument browser functions.

8 CONCLUSIONS

Our work calls into question the fidelity of web measurement tools. This fidelity has, so far, been mostly overlooked in measurement studies (see Tbl. 1). We show that the most widely used web measurement tool, OpenWPM, is easily detectable by websites. We even found OpenWPM-specific detectors in practice. Moreover, this detectability may be leveraged by websites to hide actions from OpenWPM or even attack OpenWPM's functionality, undermining the fidelity of its results.

This illustrates that web measurements should account for operating in a hostile environment. We show that OpenWPM can be hardened for such an environment, mitigating these adverse effects. However, similar caution should be taken with other web measurement tools. We have shown that the browser automation frameworks underpinning most measurement studies are themselves detectable. Our work should thus not be seen as an indictment of OpenWPM in favour of self-written one-off tooling. On the contrary, we need hardened tooling, which requires significant development effort. Our work is a call to action for web measurement studies, to ensure any potential bot-induced bias in the measurement is eliminated. This means taking a hostile environment into account while developing tooling, and validating results specifically with a view to hostile actors.

Towards robust instrumentation. Our findings show that deployment of instruments via page context is fraught with difficulties. Ideally, instrumentation is handled outside page scope. For example, by leveraging the debugger API. Unfortunately, Selenium v4 (the version used by OpenWPM) does not support this API currently. Alternatively, instrumentation could be integrated in the browser's source code. This supports great flexibility in hiding distinctive aspects of the browser fingerprint, at the cost of significant additional maintenance overhead. This would slow adoption of new browser versions; however, OpenWPM's rate of adoption is already slow – the tradeoff may thus be worth it.

Bot detection on the rise. In comparison with previous studies, we see the number of sites looking for the `webdriver` property has significantly increased in the span of less than one year (see Tbl. 11). This rapid change clearly suggests that websites are swiftly transitioning to responding differently to automated clients than to regular clients. Web studies should therefore no longer ignore the potential impact of bot detection on their study.

Advice for web measurement studies. In general, do not use virtualisation or headless or display-less modes. Studies that focus on measuring the amount of HTTP traffic seem to not be affected by detection and can, for now, get away with ignoring bot detection. In contrast, studies that focus on web tracking or cookies are affected (see Tbl. 10) and must take bot detection into account. Finally, studies that automatically crawl beyond the front page will also significantly more often be exposed to bot detectors.

Table 11: Studies measuring `webdriver` property access on front pages

	when	analysis	corpus	# sites	%
[46]	2019–10	dynamic	Alexa 50K	2,756	5.51%
This paper	2020–07	combined	Tranco 100K	13,989	13.99%
		– static		*11,957*	*11,96%*
		– dynamic		*12,194*	*12.19%*

Advice for automated web measurement tooling. This paper identified two main challenges for measurement tooling: tooling resilience and reliability of its measurements. Tooling resilience requires assuming that the measured site is actively trying to break the measurement tool. This thus necessitates programming the measurement tooling defensively. Secondly, reliability of measurements is under pressure if the tooling's interactions with the measured objects deviate significantly from regular interactions. To minimise this effect, web measurement tools must take effort to blend in with human-originating traffic. Concretely, that includes avoiding DOM pollution as well as avoiding or reducing other tell-tale traces. Finally, these aspects should both be checked. That is: the detectability of the measurement tool should be checked using standard techniques (e.g., fingerprinting [45], template attacks [75], behaviour [37]). There currently are no standardised ways to check measurement platforms for susceptibility to malicious data; however, the approach we took in Sec. 6.2 provides a starting point.

ACKNOWLEDGEMENTS

We wish to thank Daniel Goßen for his tests for injection JavaScript into iFrames. Additional thanks are due to Alan Scott Davies and Patrick Ruhe, whom both supported us in gaining access to residential IP addresses. We are grateful to Steven Englehardt and Stefan Zabka from OpenWPM, for their constructive and positive interactions. Christopher Wilger and Svenja Schulze were kind enough to help with rechecking Tbl. 15 to ensure its correctness. Finally, we would like to thank reviewers of earlier drafts of this paper for their constructive comments.

REFERENCES

[1] Gunes Acar, Steven Englehardt, and Arvind Narayanan. No boundaries: data exfiltration by third parties embedded on web pages. *Proc. Priv. Enhancing Technol.*, 2020(4):220–238, 2020.

[2] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juárez, Arvind Narayanan, and Claudia Díaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proc. 21st ACM SIGSAC Conference on Computer and Communications Security (CCS'14)*, pages 674–689. ACM, 2014.

[3] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. FPDetective: dusting the web for fingerprinters. In *Proc. 20th ACM SIGSAC Conference on Computer and Communications Security (CCS'13)*, pages 1129–1140. ACM, 2013.

[4] Pushkal Agarwal, Sagar Joglekar, Panagiotis Papadopoulos, Nishanth Sastry, and Nicolas Kourtellis. Stop tracking me bro! Differential tracking of user demographics on hyper-partisan websites. In *Proc. 29th The Web Conference 2020 (WWW'20)*, pages 1479–1490. ACM, 2020.

[5] Syed Suleman Ahmad, Muhammad Daniyal Dar, Muhammad Fareed Zaffar, Narseo Vallina-Rodriguez, and Rishab Nithyanand. Apophanies or epiphanies? How crawlers impact our understanding of the web. In *Proc. 29th The Web Conference 2020 (WWW'20)*, page 271–280. ACM, 2020.

[6] Suzan Ali, Tousif Osman, Mohammad Mannan, and Amr M. Youssef. On privacy risks of public wifi captive portals. In *DPM/CBT@ESORICS*, volume 11737 of *LNCS*, pages 80–98. Springer, 2019.

[7] Ibrahim Altaweel, Nathan Good, and Chris Jay Hoofnagle. Web privacy census. *Technology Science*, 2015121502, 2015.

[8] Amelia Andersdotter and Anders Jensen-Urstad. Evaluating websites and their adherence to data protection principles: Tools and experiences - contributions to IFIP summer school proceedings. In *Privacy and Identity Management*, volume 498 of *IFIP Advances in Information and Communication Technology*, pages 39–51, 2016.

[9] Muhammad Ahmad Bashir, Sajjad Arshad, William K. Robertson, and Christo Wilson. Tracing information flows between ad exchanges using retargeted ads. In *25th USENIX Security Symposium (USENIX Security'16)*, pages 481–496. USENIX Association, 2016.

[10] Reuben Binns, Jun Zhao, Max Van Kleek, and Nigel Shadbolt. Measuring third-party tracker power across web and mobile. *ACM Trans. Internet Techn.*, 18(4):52:1–52:22, 2018.

[11] Paul E. Black. Ratcliff/obershelp pattern recognition. https://www.nist.gov/dads/HTML/ratcliffObershelp.html, 2021. last access: November 1, 2022.

[12] Dino Bollinger, Karel Kubicek, Carlos Cotrini, and David Basin. Automating cookie consent and GDPR violation detection. In *31st USENIX Security Symposium (USENIX Security'22)*, pages 2893–2910, Boston, MA, 2022. USENIX Association.

[13] Justin Brookman, Phoebe Rouge, Aaron Alva, and Christina Yeung. Cross-device tracking: Measurement and disclosures. *Proc. Priv. Enhancing Technol.*, 2017(2):133–148, 2017.

[14] Stefano Calzavara, Tobias Urban, Dennis Tatang, Marius Steffens, and Ben Stock. Reining in the web's inconsistencies with site policy. In *Proc. 28th Network and Distributed System Security Symposium (NDSS'21)*. The Internet Society, 2021.

[15] Darion Cassel, Su-Chin Lin, Alessio Buraggina, William Wang, Andrew Zhang, Lujo Bauer, Hsu-Chun Hsiao, Limin Jia, and Timothy Libert. Omnicrawl: Comprehensive measurement of web tracking with real desktop and mobile browsers. *Proc. Privacy Enhancing Technologies Symposium (PETS'22)*, 2022(1):227–252, 2022.

[16] Quan Chen, Panagiotis Ilia, Michalis Polychronakis, and Alexandros Kapravelos. Cookie swap party: Abusing first-party cookies for web tracking. In *Proc. 31st The Web Conference 2022 (WWW'22)*, pages 2117–2129. ACM, 2021.

[17] Zi Chu, Steven Gianvecchio, Aaron Koehl, Haining Wang, and Sushil Jajodia. Blog or block: Detecting blog bots through behavioral biometrics. *Computer Networks*, 57(3):634–646, 2013.

[18] Cliqz GmbH. Whotracks.me - learn about tracking technologies, market structure and data-sharing on the web. https://whotracks.me/, 2021. last access: November 1, 2022.

[19] John Cook, Rishab Nithyanand, and Zubair Shafiq. Inferring tracker-advertiser relationships in the online advertising ecosystem using header bidding. *Proc. Priv. Enhancing Technol.*, 2020(1):65–82, 2020.

[20] Vittoria Cozza, Van Tien Hoang, Marinella Petrocchi, and Rocco De Nicola. Transparency in keyword faceted search: An investigation on google shopping. In *IRCDL*, volume 988 of *Communications in Computer and Information Science*, pages 29–43. Springer, 2019.

[21] Ha Dao and Kensuke Fukuda. Characterizing CNAME cloaking-based tracking on the web. In *TMA*. IFIP, 2020.

[22] Ha Dao and Kensuke Fukuda. A machine learning approach for detecting CNAME cloaking-based tracking on the web. In *GLOBECOM*, pages 1–6. IEEE, 2020.

[23] Ha Dao, Johan Mazel, and Kensuke Fukuda. Understanding abusive web resources: characteristics and counter-measures of malicious web resources and cryptocurrency mining. In *AINTEC*, pages 54–61. ACM, 2018.

[24] Ha Dao, Johan Mazel, and Kensuke Fukuda. CNAME cloaking-based tracking on the web: Characterization, detection, and protection. *IEEE Trans. Netw. Serv. Manag.*, 18(3):3873–3888, 2021.

[25] Anupam Das, Gunes Acar, Nikita Borisov, and Amogh Pradeep. The web's sixth sense: A study of scripts accessing smartphone sensors. In *Proc. 25th ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*, pages 1515–1532. ACM, 2018.

[26] Nurullah Demir, Matteo Große-Kampmann, Tobias Urban, Christian Wressnegger, Thorsten Holz, and Norbert Pohlmann. Reproducibility and replicability of web measurement studies. In *Proc. 31st The Web Conference 2022 (WWW'22)*, page 533–544. ACM, 2022.

[27] Rob van Eijk, Hadi Asghari, Philipp Winter, and Arvind Narayanan. The impact of user location on cookie notices (inside and outside of the european union). In *Workshop on Technology and Consumer Protection (ConPro'19)*, 2019.

[28] Steven Englehardt, Jeffrey Han, and Arvind Narayanan. I never signed up for this! privacy implications of email tracking. *Proc. Priv. Enhancing Technol.*, 2018(1):109–126, 2018.

[29] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proc. 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*, pages 1388–1401. ACM, 2016.

[30] Steven Englehardt, Dillon Reisman, Christian Eubank, Peter Zimmerman, Jonathan R. Mayer, Arvind Narayanan, and Edward W. Felten. Cookies that give you away: The surveillance implications of web tracking. In *Proc. 24th International Conference on World Wide Web (WWW'15)*, pages 289–299. ACM, 2015.

[31] Imane Fouad, Nataliia Bielova, Arnaud Legout, and Natasa Sarafijanovic-Djukic. Missed by filter lists: Detecting unknown third-party trackers with invisible pixels. *Proc. Priv. Enhancing Technol.*, 2020(2):499–518, 2020.

[32] Imane Fouad, Cristiana Santos, Feras Al Kassar, Nataliia Bielova, and Stefano Calzavara. On compliance of cookie purposes with the purpose specification principle. In *EuroS&P Workshops*, pages 326–333. IEEE, 2020.

[33] Imane Fouad, Cristiana Santos, Arnaud Legout, and Nataliia Bielova. My cookie is a phoenix: Detection, measurement, and lawfulness of cookie respawning with browser fingerprinting. *Proc. Priv. Enhancing Technol.*, 2022(3):79–98, 2022.

[34] Nathaniel Fruchter, Hsin Miao, Scott Stevenson, and Rebecca Balebako. Variations in tracking in relation to geographic location. *Proc. of the 9th Workshop on Web 2.0 Security and Privacy (W2SP) 2015*, 2015.

[35] Steven Goldfeder, Harry A. Kalodner, Dillon Reisman, and Arvind Narayanan. When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies. *Proc. Priv. Enhancing Technol.*, 2018(4):179–199, 2018.

[36] Hélder Gomes, André Zúquete, Gonçalo Paiva Dias, and Fábio Marques. Usage of HTTPS by municipal websites in portugal. In *WorldCIST (2)*, volume 931 of *Advances in Intelligent Systems and Computing*, pages 155–164. Springer, 2019.

[37] Daniel Goßen, Hugo Jonker, Stefan Karsch, Benjamin Krumnow, and David Roefs. HLISA: Towards a more reliable measurement tool. In *Proc. 21st ACM Internet Measurement Conference (IMC'21)*, pages 380–389. ACM, 2021.

[38] Aniko Hannak, Gary Soeller, David Lazer, Alan Mislove, and Christo Wilson. Measuring price discrimination and steering on e-commerce web sites. In *Proc. 14th ACM Internet Measurement Conference (IMC'14)*, pages 305–318. ACM, 2014.

[39] Grant Ho, Dan Boneh, Lucas Ballard, and Niels Provos. Tick tock: Building browser red pills from timing side channels. In *WOOT*, pages 1–11. USENIX Association, 2014.

[40] Henry Hosseini, Martin Degeling, Christine Utz, and Thomas Hupperich. Unifying privacy policy detection. *Proc. Priv. Enhancing Technol.*, 2021(4):480–499, 2021.

[41] Xuehui Hu, Guillermo Suarez de Tangil, and Nishanth Sastry. Multi-country study of third party trackers from real browser histories. In *Proc. 6th IEEE European Symposium on Security and Privacy (EuroS&P'20)*, pages 70–86. IEEE, 2020.

[42] Luca Invernizzi, Kurt Thomas, Alexandros Kapravelos, Oxana Comanescu, Jean Michel Picod, and Elie Bursztein. Cloak of visibility: Detecting when machines browse a different web. In *Proc. 37th IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 743–758, 2016.

[43] Umar Iqbal, Steven Englehardt, and Zubair Shafiq. Fingerprinting the fingerprinters: Learning to detect browser fingerprinting behaviors. In *Proc. 42nd IEEE Symposium on Security and Privacy (S&P'21)*, pages 1143–1161, 2021.

[44] Umar Iqbal, Charlie Wolfe, Charles Nguyen, Steven Englehardt, and Zubair Shafiq. Khaleesi: Breaker of advertising and tracking request chains. In *31st USENIX Security Symposium (USENIX Security'22)*, pages 2911–2928, Boston, MA, 2022. USENIX Association.

[45] Hugo Jonker, Benjamin Krumnow, and Gabry Vlot. Fingerprint surface-based detection of web bot detectors. In *Proc. 24th European Symposium on Research in Computer Security (ESORICS'19)*, volume 11736 of *LNCS*, pages 586–605. Springer, 2019.

[46] Jordan Jueckstock and Alexandros Kapravelos. Visiblev8: In-browser monitoring of javascript in the wild. In *Proc. 19th ACM Internet Measurement Conference (IMC'19)*, pages 393–405. ACM, 2019.

[47] Jordan Jueckstock, Shaown Sarker, Peter Snyder, Aidan Beggs, Panagiotis Papadopoulos, Matteo Varvello, Ben Livshits, and Alexandros Kapravelos. Towards realistic and reproducible web crawl measurements. In *Proc. 30th The Web Conference 2021 (WWW'21)*. ACM, 2021.

[48] Martin Koop, Erik Tews, and Stefan Katzenbeisser. In-depth evaluation of redirect tracking and link usage. *Proc. Priv. Enhancing Technol.*, 2020(4):394–413, 2020.

[49] Michael Kranch and Joseph Bonneau. HTTPS in mid-air: An empirical study of strict transport security and key pinning. In *Proc. 22nd Network and Distributed System Security Symposium (NDSS'15)*. The Internet Society, 2015.

[50] Dhruv Kuchhal and Frank Li. Knock and talk: investigating local network communications on websites. In *Proc. 21st ACM Internet Measurement Conference (IMC'21)*, pages 550–568. ACM, 2021.

[51] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. In *Proc. 26th Network and Distributed System Security Symposium (NDSS'19)*, pages 1–15. The Internet Society, 2019.

[52] Adam Lerner, Anna Kornfeld Simpson, Tadayoshi Kohno, and Franziska Roesner. Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016. In *25th USENIX Security Symposium (USENIX Security'16)*. USENIX Association, 2016.

[53] Baojun Liu, Zhou Li, Peiyuan Zong, Chaoyi Lu, Hai-Xin Duan, Ying Liu, Sumayah A. Alrwais, XiaoFeng Wang, Shuang Hao, Yaoqi Jia, Yiming Zhang, Kai Chen, and Zaifeng Zhang. Traffickstop: Detecting and measuring illicit traffic monetization through large-scale DNS analysis. In *Proc. 5th IEEE European Symposium on Security and Privacy (EuroS&P'19)*, pages 560–575. IEEE, 2019.

[54] Baojun Liu, Zhou Li, Peiyuan Zong, Chaoyi Lu, Hai-Xin Duan, Ying Liu, Sumayah A. Alrwais, XiaoFeng Wang, Shuang Hao, Yaoqi Jia, Yiming Zhang, Kai Chen, and Zaifeng Zhang. Traffickstop: Detecting and measuring illicit traffic monetization through large-scale DNS analysis. In *Proc. 5th IEEE European Symposium on Security and Privacy (EuroS&P'19)*, pages 560–575. IEEE, 2019.

[55] Fang Liu, Chun Wang, Andres Pico, Danfeng Yao, and Gang Wang. Measuring the insecurity of mobile deep links of android. In *26th USENIX Security Symposium (USENIX Security'17)*, pages 953–969. USENIX Association, 2017.

[56] Max Maass, Stephan Schwär, and Matthias Hollick. Towards transparency in email tracking. In *APF*, volume 11498 of *LNCS*, pages 18–27. Springer, 2019.

[57] Max Maaß, Pascal Wichmann, Henning Pridöhl, and Dominik Herrmann. Privacyscore: Improving privacy and security via crowd-sourced benchmarks of websites. In *APF*, volume 10518 of *LNCS*, pages 178–191. Springer, 2017.

[58] Arunesh Mathur, Gunes Acar, Michael Friedman, Elena Lucherini, Jonathan R. Mayer, Marshini Chetty, and Arvind Narayanan. Dark patterns at scale: Findings from a crawl of 11k shopping websites. *Proc. ACM Hum. Comput. Interact.*, 3(CSCW):81:1–81:32, 2019.

[59] Johan Mazel, Richard Garnier, and Kensuke Fukuda. A comparison of web privacy protection techniques. *Comput. Commun.*, 144:162–174, 2019.

[60] Georg Merzdovnik, Markus Huber, Damjan Buhov, Nick Nikiforakis, Sebastian Neuner, Martin Schmiedecker, and Edgar R. Weippl. Block me if you can: A large-scale study of tracker-blocking tools. In *Proc. 3rd IEEE European Symposium on Security and Privacy (EuroS&P'17)*, pages 319–333. IEEE, 2017.

[61] Najmeh Miramirkhani, Oleksii Starov, and Nick Nikiforakis. Dial one for scam: A large-scale analysis of technical support scams. In *Proc. 24th Network and Distributed System Security Symposium (NDSS'17)*. The Internet Society, 2017.

[62] Maaz Bin Musa and Rishab Nithyanand. ATOM: ad-network tomography. *Proc. Priv. Enhancing Technol.*, 2022(4):295–313, 2022.

[63] Marius Musch and Martin Johns. U can't debug this: Detecting javascript anti-debugging techniques in the wild. In Michael Bailey and Rachel Greenstadt, editors, *30th USENIX Security Symposium (USENIX Security'21)*, pages 2935–2950. USENIX Association, 2021.

[64] Lukasz Olejnik, Steven Englehardt, and Arvind Narayanan. Battery status not included: Assessing privacy in web standards. In *IWPE@SP*, volume 1873 of *CEUR Workshop Proceedings*, pages 17–24. CEUR-WS.org, 2017.

[65] Shahrooz Pouryousef, Muhammad Daniyal Dar, Suleman Ahmad, Phillipa Gill, and Rishab Nithyanand. Extortion or expansion? an investigation into the costs and consequences of ICANN's gTLD experiments. In *Passive and Active Measurement (PAM'20)*, volume 12048 of *LNCS*, pages 141–157. Springer, 2020.

[66] Andrew Reed and Michael J. Kranch. Identifying https-protected netflix videos in real-time. In *CODASPY*, pages 361–368. ACM, 2017.

[67] Nathan Reitinger and Michelle L. Mazurek. ML-CB: machine learning canvas block. *Proc. Priv. Enhancing Technol.*, 2021(3):453–473, 2021.

[68] Valentino Rizzo, Stefano Traverso, and Marco Mellia. Unveiling web fingerprinting in the wild via code mining and machine learning. *Proc. Priv. Enhancing Technol.*, 2021(1):43–63, 2021.

[69] Nicky Robinson and Joseph Bonneau. Cognitive disconnect: understanding facebook connect login permissions. In *Proc. 2nd ACM Conference on Online social networks (COSN'14)*, pages 247–258. ACM, 2014.

[70] Sebastian Roth, Timothy Barron, Stefano Calzavara, Nick Nikiforakis, and Ben Stock. Complex security policy? A longitudinal analysis of deployed content security policies. In *NDSS*. The Internet Society, 2020.

[71] Takahito Sakamoto and Masahiro Matsunaga. After GDPR, still tracking or not? understanding opt-out states for online behavioral advertising. In *IEEE Symposium on Security and Privacy Workshops*, pages 92–99. IEEE, 2019.

[72] Nayanamana Samarasinghe, Aashish Adhikari, Mohammad Mannan, and Amr M. Youssef. Et tu, brute? privacy analysis of government websites and mobile apps. In *Proc. 31st The Web Conference 2022 (WWW'22)*, pages 564–575. ACM, 2022.

[73] Nayanamana Samarasinghe and Mohammad Mannan. Towards a global perspective on web tracking. *Comput. Secur.*, 87, 2019.

[74] Steven Schmeiser. Online advertising networks and consumer perceptions of privacy. *Applied Economics Letters*, 25(11):776–780, 2017.

[75] Michael Schwarz, Florian Lackner, and Daniel Gruss. Javascript template attacks: Automatically inferring host information for targeted exploits. In *Proc. 26th Annual Network and Distributed System Security Symposium (NDSS'19)*. The Internet Society, 2019.

[76] Sergey Shekyan. Detecting PhantomJS based visitors. https://blog.shapesecurity.com/2015/01/22/detecting-phantomjs-based-visitors/, 2015. last access: November 1, 2022.

[77] Sandra Siby, Umar Iqbal, Steven Englehardt, Zubair Shafiq, and Carmela Troncoso. {WebGraph}: Capturing advertising and tracking information flows for robust blocking. In *31st USENIX Security Symposium (USENIX Security'22)*, pages 2875–2892, Boston, MA, 2022. USENIX Association.

[78] Suphannee Sivakorn, Jason Polakis, and Angelos D Keromytis. I'm not a human: Breaking the google recaptcha. *Black Hat*, pages 1–12, 2016.

[79] Ido Sivan-Sevilla, Wenyi Chu, Xiaoyu Liang, and Helen Nissenbaum. Unaccounted privacy violation: A comparative analysis of persistent identification of users across social contexts. 2021.

[80] Peter Snyder, Cynthia Bagier Taylor, and Chris Kanich. Most websites don't need to vibrate: A cost-benefit approach to improving browser security. In *Proc. 24th ACM SIGSAC Conference on Computer and Communications Security (CCS'17)*, pages 179–194. ACM, 2017.

[81] Konstantinos Solomos, Panagiotis Ilia, Sotiris Ioannidis, and Nicolas Kourtellis. TALON: an automated framework for cross-device tracking detection. In *RAID*, pages 227–241. USENIX Association, 2019.

[82] Konstantinos Solomos, Panagiotis Ilia, and Nicolas Kourtellis. Clash of the trackers: Measuring the evolution of the online tracking ecosystem. In *TMA*. IFIP, 2020.

[83] Jannick Kirk Sørensen and Sokol Kosta. Before and after GDPR: the changes in third party presence at public and private european websites. In *Proc. 28th The Web Conference 2019 (WWW'19)*, pages 1590–1600. ACM, 2019.

[84] Oleksii Starov, Johannes Dahse, Syed Sharique Ahmad, Thorsten Holz, and Nick Nikiforakis. No honor among thieves: A large-scale analysis of malicious web shells. In *Proc. 25th International Conference on World Wide Web (WWW'16)*, pages 1021–1032. ACM, 2016.

[85] Giorgio Di Tizio and Fabio Massacci. A calculus of tracking: Theory and practice. *Proc. Priv. Enhancing Technol.*, 2021(2):259–281, 2021.

[86] Christof Ferreira Torres, Hugo L. Jonker, and Sjouke Mauw. Fp-block: Usable web privacy by controlling browser fingerprinting. In *Proc. 20th European Symposium on Research in Computer Security (ESORICS'15)*, volume 9327 of *LNCS*, pages 3–19. Springer, 2015.

[87] Tobias Urban, Martin Degeling, Thorsten Holz, and Norbert Pohlmann. Beyond the front page:measuring third party dynamics in the field. In *Proc. 29th The Web Conference 2020 (WWW'20)*, page 1275–1286. ACM, 2020.

[88] Tobias Urban, Dennis Tatang, Martin Degeling, Thorsten Holz, and Norbert Pohlmann. A study on subject data access in online advertising after the GDPR. In *DPM/CBT@ESORICS*, volume 11737 of *LNCS*, pages 61–79. Springer, 2019.

[89] Tobias Urban, Dennis Tatang, Martin Degeling, Thorsten Holz, and Norbert Pohlmann. Measuring the impact of the GDPR on data sharing in ad networks. In *Proc. 15th ACM Asia Conference on Computer and Communications Security (AsiaCCS'20)*, pages 222–235. ACM, 2020.

[90] Pelayo Vallina, Álvaro Feal, Julien Gamba, Narseo Vallina-Rodriguez, and Antonio Fernández Anta. Tales from the porn: A comprehensive privacy analysis of the web porn ecosystem. In *Proc. 19th ACM Internet Measurement Conference (IMC'19)*, pages 245–258. ACM, 2019.

[91] Steven Van Acker, Daniel Hausknecht, and Andrei Sabelfeld. Raising the bar: Evaluating origin-wide security manifests. In *Proc. 34th Annual Computer Security Applications Conference (ACSAC'18)*, pages 342–354. ACM, 2018.

[92] Rob Van Eijk, Hadi Asghari, Philipp Winter, and Arvind Narayanan. The impact of user location on cookie notices (inside and outside of the european union). In *Workshop on Technology and Consumer Protection (ConPro'19). IEEE*. IEEE, 2019.

[93] Antoine Vastel. Detecting Chrome headless, new techniques. "https://antoinevastel.com/bot%20detection/2018/01/17/detect-chrome-headless-v2.html", 2018. last access: November 1, 2022.

[94] Antoine Vastel, Walter Rudametkin, Romain Rouvoy, and Xavier Blanc. FP-Crawlers: Studying the Resilience of Browser Fingerprinting to Block Crawlers. In *Proc. 2nd NDSS Workshop on Measurements, Attacks, and Defenses for the Web (MADWEB'20)*, pages 2–14, 2020.

[95] Yash Vekaria, Vibhor Agarwal, Pushkal Agarwal, Sangeeta Mahapatra, Sakthi Balan Muthiah, Nishanth Sastry, and Nicolas Kourtellis. Differential tracking across topical webpages of indian news media. In *Proc. 13th ACM Web Science Conference (WebSci'21)*, pages 299–308. ACM, 2021.

[96] David Y. Wang, Stefan Savage, and Geoffrey M. Voelker. Cloak and dagger: dynamics of web search cloaking. In *Proc. 18th ACM Conference on Computer and Communications Security (CCS'11)*, pages 477–490, 2011.

[97] Baoning Wu and Brian D. Davison. Cloaking and redirection: A preliminary study. In *1st International Workshop on Adversarial Information Retrieval on the Web (AIRWeb'05)*, pages 7–16, 2005.

[98] Baoning Wu and Brian D. Davison. Detecting semantic cloaking on the web. In *Proc. 15th international conference on World Wide Web (WWW'06)*, pages 819–828, 2006.

[99] Zhiju Yang and Chuan Yue. A comparative measurement study of web tracking on mobile and desktop environments. *Proc. Priv. Enhancing Technol.*, 2020(2):24–44, 2020.

[100] Xiufen Yu, Nayanamana Samarasinghe, Mohammad Mannan, and Amr M. Youssef. Got sick and tracked: Privacy analysis of hospital websites. In *EuroS&P Workshops*, pages 278–286. IEEE, 2022.

[101] David Zeber, Sarah Bird, Camila Oliveira, Walter Rudametkin, Ilana Segall, Fredrik Wollsén, and Martin Lopatka. The representativeness of automated web crawls as a surrogate for human browsing. In *Proc. 29th The Web Conference 2020 (WWW'20)*, page 167–178. ACM, 2020.

[102] Penghui Zhang, Adam Oest, Haehyun Cho, RC Johnson, Brad Wardman, Shaown Sarker, Alexandros Kpravelos, Tiffany Bao, Ruoyu Wang, Yan Shoshitaishvili, Adam Doupé, and Gail-Joon Ahn. CrawlPhish: Large-scale Analysis of Client-side Cloaking Techniques in Phishing. In *Proc. 42nd IEEE Symposium on Security and Privacy (S&P'21)*, pages 1109–1124, 2021.

A FIRST-PARTY DETECTOR PATTERNS

Tbl. 12 shows patterns from our first-party script analysis in Sec. 4.3. Scripts provided by Akamai, Incapsula, Cloudflare, and PerimeterX follow the same script pattern, these can be easily recognised. For the unknown script, we found that the for common path patterns between larger clusters of script hashes. A manual validation showed that scripts found under the listed path are most similar.

Table 12: Similarities in first-party detectors

Origin	URL path similarities	# sites
Akamai	domain/akam/11/…	1,004
Incapsula	domain/_Incapsula_Resource?…	998
Unknown	domain/asssets/{hash of 31-32 bits length} domain/resources/{hash of 32-33 bits length} domain/public/{hash of 32-33 bits length} domain/static/{hash of 34 bits length}	659
Cloudflare	domain/…/cdn-cgi/bm/cv/2172558837/api.js	486
PerimeterX	domain/…/{8 character string}/init.js	134

B PATTERNS USED IN STATIC ANALYSIS

We iterate on the pattern design to reduce false positives. Our very first run used patterns matching strings literally. However, in the specific case of matching the term *webdriver*, we found that this selects scripts that use this word in another context than checking Selenium-driven Firefox browsers (see e.g. [45, 46] for conflicting bot detection properties with this term). In the next iteration we used patterns that take the context of the access to a property into account. For example, the pattern `navigator\[["']webdriver["']\]` only matches if the `webdriver` property is checked via the navigator object. Tbl. 13 lists our explored patterns. Finally, we manually checked a random subset to check pattern performance. Only one pattern still introduced false positives; all its matches were manually validated and false positives eliminated.

Table 13: Patterns evaluated in static analysis

Pattern	false positives found
`webdriver`	✓
`instrumentFingerprintingApis`	-
`getInstrumentJS`	-
`jsInstruments`	-
`(?<!_|-)webdriver(?!_|-)`	✓
`navigator.webdriver`	-
`navigator\[["']webdriver["']\]`	-

C INTEGRATION OF FIREFOX VERSIONS

Releases of OpenWPM do not appear synchronously with Firefox. As a result, certain time frames exist where the OpenWPM client uses an older Firefox versions than regular users. Tbl. 14 summarises migration of Firefox versions in the OpenWPM Framework since version 0.10.0. Between the release of Firefox 77 (March 2020) and the release of Firefox 104 (current at the time of writing) were 780 days. Within this period, OpenWPM was shipped with an outdated version for 540 days (69%).

Table 14: Migration to newer Firefox releases in OpenWPM

Firefox	release date	OpenWPM	integration date	Outdated
104.0	07/23/22			53 days
101.0	05/31/22			
100.0	05/03/22	0.20.0	05/05/22	30 days
99.0	04/05/22			
98.0	03/08/22	0.19.0	03/10/22	58 days
96.0	01/11/22			
95.0	12/07/21	0.18.0	12/16/21	69 days
91.0	08/10/21			
90.0	07/13/21	0.17.0	07/24/21	11 days
89.0	06/01/21	0.16.0	06/10/21	9 days
88.0	04/19/21	0.15.0	05/10/21	48 days
87.0	03/23/21			
86.0.1	03/11/21	0.14.0	03/12/21	87 days
84.0	12/15/20			
83.0	11/18/20	0.13.0	11/19/20	58 days
81.0	09/22/20			
80.0	08/25/20	0.12.0	08/26/20	29 days
79.0	07/28/20			
78.0.1	07/01/20	0.11.0	07/09/20	9 days
78.0.	06/30/20			
77.0	06/03/20	0.10.0	06/23/20	20 days

D SILENT DELIVERY OF JS FILES

Listing 4 shows an attack to bypass OpenWPM's mode of recording only JavaScript files. Note that the loaded resource does not include a file extension. Therefore, it will be loaded as regular text and its content does not occur in OpenWPM's logging of loaded JavaScript files. After loading, the content is executed via *eval*.

```
const stealth_code = "https://{attacker_domain}/cheat";
fetch(stealth_code)              // load code from server
    .then(res => res.text())     // convert code to JS-string
      .then(res => eval(res));   // code execution
```

Listing 4: Example to silently load a JS file

E OPENWPM IN LITERATURE

Tbl. 15 provides a detailed view on our analysis of peer-reviewed studies based on OpenWPM. Each category that applies to a study is marked with a "✓". For those studies that measure certain aspects, but rely on out of bound mechanisms (e.g., by deploying a proxy) and do not rely on OpenWPM's instrumentation are marked with a "○". Running modes are shortened in the table as follow: unspecified (u), native (n), headless (h), xvfb (x), docker (d)Papers that are not included in the seed list, but where added by us, are highlighted with a "★". Studies marked with a "†" use an OpenWPM data set, but do not perform their own data acquisition.

Table 15: Overview of previous studies using OpenWPM for web studies

				deployed as		measures/analyses			performs			visits	uses	mentions
Year	**Ref.**	**Venue**	**1st Author**	**Mode**	**VM**	**Cookies**	**HTTP**	**JS**	**Scrolling**	**Clicking**	**Typing**	**Sub-pages**	**Anti-BD**	**BD**
2014	[2]	CCS	Acar	u	✓	○	○	✓						
	[69]	CoSN	Robinson	u						✓	✓			
2015	[30]	WWW	Englehardt	u	✓	✓	✓							
	[49]	NDSS	Kranch	u		✓	○							
	[7]	Tech Science	Altaweel	h		✓	✓			✓		✓		
	[34]	W2SP	Fruchter	u	✓	✓	✓							
2016	[8]	IFIP AICT	Andersdotter	u		✓	✓					✓		
	[29]	CCS	Englehardt	x	✓	✓	✓	✓				✓		
	[84]	WWW	Starov	u	✓		✓							
2017	[61]	NDSS	Miramirkhani	u	✓		○	✓		✓				
	[13]	PETS	Brookman	u	✓	✓	✓							
	[66]	CODASPY	Reed	u			✓							
	[64]	CEUR	Olejnik	u				✓						
	[57]	APF	Maass	u		✓	✓							
	[55]	USENIX	Liu	h										
	[74]	Appl. Econ. Letters	Schmeiser	u			✓							
2018	[35]	PETS	Goldfeder	u			✓							
	[28]	PETS	Englehardt	u			✓			✓	✓	✓		✓
	[10]	ACM ToIT	Binns	h		✓	✓							
	[25]	CCS	Das	u			✓	✓						✓
	[91]	ACSAC	van Acker	u			✓							
	[23]	AINTEC	Dao	u			✓							
2019	[20]	IRCDL	Cozza	u					✓	✓	✓	✓		
	[36]	WorldCIST	Gomes	u			✓							
	[92]	ConPro	van Eijk	d										
	[83]	WWW	Sørensen	u	✓		✓					✓		
	[54]	EuroS&P	Liu	u			✓							✓
	[58]	CSCW	Mathur	u			✓			✓		✓		
	[59]	Comput. Comm.	Mazel	u			✓							
	[6]	LPM	Ali	u		✓								
	[73]	Comp. Secur.	Samarasinghe	u		✓	✓							✓
	[56]	APF	Maass	u			✓							
	[81]	RAID	Solomos	u					✓	✓				
	[90]	IMC	Vallina	u		✓	✓	✓						
	[45]	ESORCIS	Jonker	h		✓		○						✓
	[88]	DPM	Urban	u		✓	✓					✓		
	[71]	SPW	Sakamoto	u		✓								
2020	[31]	PETS	Fouad	u		✓	✓					✓		
	[19]	PETS	Cook	u					✓				✓	✓
	[99]	PETS	Yang	u		✓	✓	✓	✓					
	[1]	PETS	Acar	u	✓		✓	✓				✓	✓	✓
	[48]	PETS	Koop	d		✓	✓	✓		✓			✓	
	[101]	WWW	Zeber	n/x	✓	✓	✓	✓						✓
	[5]	WWW	Ahmad	u			✓						✓	✓
	[4]	WWW	Agarwal	h	✓	✓	✓	✓						
	[87]	WWW	Urban	u		✓	✓	✓	✓			✓	✓	✓
	[89]	AsiaCCS	Urban	u	✓	✓	✓		✓			✓	✓	✓
	[65]	PAM	Pouryousef	u		✓								
	[32]	EuroS&P	Fouad	u		✓	✓							
	[79]	PrivacyCon	Sivan-Sevilla	u	✓	✓	✓	✓					✓	✓
	[41]	EuroS&P	Hu	u		✓	✓							
	[21]	TMA	Dao	u			✓							
	[82]	TMA	Solomos	n		✓	✓							
	[22]	GLOBECOM	Dao	u			✓							
2021	[14]	NDSS	Calzavara	u		✓	✓					✓		✓
	[68]	PETS	Rizzo	u	✓		✓	✓				✓		
	[43]	S&P	Iqbal	u			✓	✓						
	★[37]	IMC	Goßen	n			✓		✓	✓	✓			✓
	★†[85]	PETS	Di Tizio	u			✓							
	★[67]	PETS	Reitinger	u			✓							
	★[40]	PETS	Hosseini	u										
	★[95]	WebSci	Vekaria	u		✓	✓	✓				✓		
	★[24]	IEEE TNSM.	Dao	u			✓			✓				
2022	★[15]	PETS	Cassel	u			○	○						✓
	★[77]	USENIX	Siby	u		✓	✓	✓						
	★[44]	USENIX	Iqbal	u		✓	✓	✓		✓		✓		✓
	★[33]	PETS	Fouad	u		✓	✓	✓						
	★[26]	WWW	Demir	n/h	✓		✓				✓	✓		✓
	★[100]	EuroS&PW	Yu	h		✓	✓	✓						
	★[62]	PETS	Musa	u			✓						✓	✓
	★[72]	WWW	Samarasinghe	u	✓	✓	✓	✓						
	★[12]	USENIX	Bollinger	u	✓	✓			✓			✓		