Apophanies or Epiphanies? How Crawlers Impact Our Understanding of the Web

Syed Suleman Ahmad
University of Wisconsin - Madison

Muhammad Daniyal Dar
University of Iowa

Muhammad Fareed Zaffar
LUMS

Narseo Vallina-Rodriguez
IMDEA Networks / ICSI

Rishab Nithyanand
University of Iowa

ABSTRACT

Data generated by web crawlers has formed the basis for much of our current understanding of the Internet. However, not all crawlers are created equal and crawlers generally find themselves trading off between computational overhead, developer effort, data accuracy, and completeness. Therefore, the choice of crawler has a critical impact on the data generated and knowledge inferred from it. In this paper, we conduct a systematic study of the trade-offs presented by different crawlers and the impact that these can have on various types of measurement studies. We make the following contributions: First, we conduct a survey of all research published since 2015 in the premier security and Internet measurement venues to identify and verify the repeatability of crawling methodologies deployed for different problem domains and publication venues. Next, we conduct a qualitative evaluation of a subset of all crawling tools identified in our survey. This evaluation allows us to draw conclusions about the suitability of each tool for specific types of data gathering. Finally, we present a methodology and a measurement framework to empirically highlight the differences between crawlers and how the choice of crawler can impact our understanding of the web.

ACM Reference Format:
Syed Suleman Ahmad, Muhammad Daniyal Dar, Muhammad Fareed Zaffar, Narseo Vallina-Rodriguez, and Rishab Nithyanand. 2020. Apophanies or Epiphanies? How Crawlers Impact Our Understanding of the Web. In *Proceedings of The Web Conference 2020 (WWW '20), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3366423.3380113

1 INTRODUCTION

Web crawlers and scrapers[1] have been a core component of the toolkits used by the Internet measurement, security & privacy, information retrieval, and networking communities. In fact, nearly 16% of all publications from the 2015-18 editions of ACM Internet Measurement Conference (IMC) [23], Network and Distributed System Security Symposium (NDSS) [27], ACM Conference on Computer and Communication Security (CCS) [13], USENIX Security Symposium (SEC) [34], IEEE Symposium on Security & Privacy (S&P) [22], International World Wide Web Conference (WWW) [25], International AAAI Conference on Web and Social Media (ICWSM) [24], and Privacy Enhancing Technology Symposium (PETS) [31] relied on Web data obtained through crawlers. However, not all crawlers are created equal or offer the same features, and therefore the choice of crawler is critical. Crawlers generally find themselves *trading-off between computational overhead, flexibility, data accuracy & completeness, and developer effort.* On one extreme, command-line tools such as `Wget` [35] and `cURL` [18] are lightweight and easy to use while providing data that is hardly accurate or complete – e.g., these tools have the inability to handle dynamically generated content or execute JavaScript which restricts the completeness of gathered data [21]. As JavaScript is currently used in over 95% of the Alexa Top-100K web rank (as of August 2019), their use as crawlers in specific types of research is questionable [37]. On the other extreme, tools like Selenium [10] provide the ability to get extremely accurate and complete data at the cost of high developer effort (needed to develop automated web interaction models that can mimic real users) and computational overhead (needed to run a fully-fledged Selenium-driven browser such as Firefox or Chrome). Further complicating matters, in response to a growing variety of attacks, webservers are becoming increasingly sensitive to and willing to block traffic appearing to be anomalous – including traffic appearing to be generated by programmatic crawlers [15, 33].

As a consequence of the above mentioned trade-offs, it is important to understand how the choice of crawler can make an impact on data gathered and inferences drawn by a measurement study. What is lacking, however, is a systematic study of the trade-offs presented by different crawlers and the impact that these can have on various types of measurement studies. In this paper, we fill this gap. Our overall objective is to provide researchers with an understanding of the trade-offs proposed by different crawlers and also to provide data-driven insight into their impact on Internet measurements and the inferences drawn from them. We accomplish this objective by making the following contributions.

Surveying crawlers used in prior work. (§2) We start our study by surveying research published since 2015 at the following venues: IMC [23], NDSS [27], S&P [22], CCS [13], SEC [34], WWW [25], ICWSM [24], and PETS [31]. In our survey of 2,424 publications, we find 350 papers that rely on data gathered by 19 unique crawlers, which we categorize by their intended purpose. We use the findings of this survey to select the most relevant crawlers in these academic communities for further analysis.

A systematic qualitative evaluation of crawling tools. (§2) We focus on performing a qualitative evaluation of different crawling tools used by previous research. For this evaluation, we rely

[1] Although there is a subtle distinction between a web crawler and a web scraper, we refer to both as "crawlers" in the remainder of this paper.

WWW '20, April 20–24, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-7023-3/20/04.
https://doi.org/10.1145/3366423.3380113

on the results of our survey to select crawlers for further analysis. We study the capabilities and flexibility of each of these crawlers with the goal of understanding how they are likely to impact the data collection and their limitations. This allows us to draw conclusions about the suitability of each tool for specific types of research problems.

A framework for performing comparative empirical crawler evaluations. (§3) We present a methodology and accompanying framework for empirically highlighting the differences between crawlers in terms of the network and application data they generate. Further, in an effort to guide future researchers in making a choice between different crawlers in specific scenarios, and to make our work repeatable, we make this evaluation framework available publicly at `https://sparta.cs.uiowa.edu/projects/methodologies.html`.

Illustrating the impact of different crawlers on research results and inferences. (§4) We empirically demonstrate how different crawlers impact measurement results and inferences drawn from them (and consequently: our understanding of the web). Based on large amounts of prior work, we focus on the three major problem domains: privacy, content availability, and security measurements. Through our experiments, we observe the impact on the inferences made on the (1) success of traffic analysis classifiers, (2) third-party ecosystem, and (3) incidence rates of server-side blocking. This illustrates that crawler selection does indeed impact measurement and experiment quality, soundness, and completeness.

2 QUALITATIVE EVALUATION

In this section, we conduct a qualitative comparative evaluation of a selected set of eight crawling tools uncovered by our survey (§2). The chosen crawlers (listed in §2) reflect the most popular 'out-of-the-box' solutions that are used by the research studies analyzed in our survey. Our primary focus is on understanding the capabilities and flexibility afforded by each of these tools with an eye on how these might impact the data generated by them.

2.1 Evaluation methodology

Characterizing crawling approaches. We begin our evaluation by first categorizing crawlers by their layer of operation. We make this categorization based on the observation that at a fundamental level, web crawlers are tools to generate application-layer protocol requests and process corresponding responses. The generation and processing of these requests and responses may occur in different ways. For example, some crawlers may directly interface with the application-layer of the networking stack to create (e.g., HTTP GET) requests with specific parameters, some may rely on controlling a browser to generate application-layer requests, and others may rely on mimicking user interactions with a browser to generate these requests. More concretely, we classify crawling tools into one of the following three categories.

Application-layer protocol crawlers (ALPCs). Crawlers in this category operate by handling web content as static files and perform HTTP `GET`s or `POST`s to "crawl" a website. They do not interface with web browsers. `cURL`, `Wget`, and ZAP spider fall in this category.

Browser-layer crawlers (BLCs). Crawlers in this category operate by driving web browsers. These crawlers typically provide wrappers in many popular languages to interface with browser-provided JavaScript APIs which interact and manipulate the DOM. For example, crawler-provided functions to navigate to specific webpages work by generating JavaScript which manipulates the `window.location.assign` property in the DOM of Blink-based browsers. PhantomJS, Selenium, and Puppeteer fall in this category.

User-layer crawlers (ULCs). Crawlers in this category operate by implementing a wrapper around standard browser-layer crawlers. These wrappers introduce new functionalities, such as mimicking user interaction with loaded pages, by chaining together multiple methods provided by browser-layer crawler bindings, therefore reducing the manual effort required to conduct specific types of crawls. OpenWPM and TorBC fall in this category.

Characterizing features of crawling tools. We characterize the features of different tools by the flexibility they offer to: (1) manipulate application-layer protocols (e.g., HTTP request parameters), (2) modify browser or client characteristics (e.g., enabling cookies, caching, or extensions), (3) automate user-interactions (e.g., enabling rate-limiting, automatic anti-bot detection mechanisms, etc.), (4) facilitate data gathering (e.g., saving screenshots, network traces, etc.), and (5) automate and scale-up crawls (e.g., permitting headless crawls, parallel crawling, etc.). Each of these categories is described in more detail below.

Application-layer protocol-level features. Features in this category capture the ability of a crawler to manipulate the parameters associated with typical web protocols. The ability to modify these protocol characteristics is particularly important for measuring the support of security protocols and characterizing server-side protocol behaviors – e.g., measuring server behaviour in response to different TLS options and HTTP headers. In our evaluation, we focus on the ability of the crawler to add new or modify existing HTTP headers, toggle the HTTP keepalive option, create new or manipulate generated HTTP methods (e.g., `GET` and `POST`), and set SSL/TLS options (e.g., version number).

Browser(client)-level features. Features in this category generally capture the ability of a crawler to manipulate browser characteristics. Browser characteristics have an impact in the types of requests generated by the crawler – e.g., clients configured to disable JavaScript will generate fewer requests for third-party content [49]. Therefore, it is crucial to be able to manipulate them. The features that we consider in this study are related to: (1) determining how the client handles web content – e.g., can the tool be set/configured to ignore locally cached content, not execute JavaScript, disable cookie access; (2) browser-specific settings including which extensions should be loaded and whether proxies can be specified.

User-level features. Traffic generated by crawlers are increasingly seen as unwanted and often characterized as attack traffic by web administrators [15, 33]. Consequently, it is becoming increasingly important for researchers conducting web measurements to take steps to ensure that their crawls do not trigger events that result in blocking or differential treatment. This is especially important for studies seeking to attribute failed page loads to server-side policies [65, 76] or censorship [72, 80]. This requires some crawl characteristic manipulation in order to match crawls to typical human behavior (or at the very least: not look bot-like). Therefore, we

Table 1: Breakdown of the most popular crawlers used by publication venue, year, and research domain.

	Total	Year				Publication venue							
		2015	2016	2017	2018	S&P	IMC	PETS	SEC	NDSS	CCS	WWW	ICWSM
`Wget`	11	4	3	2	2	2	2	0	3	1	2	1	0
`cURL`	11	3	0	3	5	1	3	1	3	2	1	0	0
ZAP Spider	2	0	1	0	1	0	0	0	1	1	0	0	0
PhantomJS	25	6	9	7	3	1	2	2	3	3	8	4	2
Selenium	52	6	12	20	14	3	6	4	7	11	12	9	0
Puppeteer	4	0	0	1	3	0	1	0	1	0	0	2	0
OpenWPM	8	2	1	3	2	0	0	2	1	2	2	1	0
Tor BC	5	0	0	2	3	0	0	2	0	1	2	0	0
Other (12)	24	3	9	8	4	2	6	1	2	5	4	3	1
Custom built	101	20	18	20	43	7	8	1	13	14	13	32	13
Not specified	136	36	33	29	38	11	13	7	17	11	18	47	12
Total	379	80	86	95	118	27	41	20	51	51	62	99	28

(a) Crawling tools broken down by year and publication venue. Note that a publication could use multiple crawling tools.

	Research domain							
	Privacy	Censorship	Performance	Content	Security	Mining	Automation	Survey
`Wget`	1	0	3	4	1	5	1	0
`cURL`	2	1	5	3	1	2	2	0
ZAP Spider	0	0	1	1	1	0	0	0
PhantomJS	5	2	2	9	7	5	7	0
Selenium	20	2	1	14	14	13	7	1
Puppeteer	0	0	0	3	1	1	2	0
OpenWPM	6	0	0	1	1	0	2	0
Tor BC	5	0	0	3	0	2	0	0
Other (12)	2	2	3	6	12	9	2	0
Custom built	16	0	8	39	23	35	16	6
Not specified	5	1	6	17	8	93	33	1
Total	62	8	29	100	69	165	72	8

(b) Crawling tools broken down by their use in different research domains. Note that a single use of a crawler can be counted in multiple research domains – e.g., crawlers used for gathering data about website fingerprints are counted in the "Security" and "Mining" categories.

evaluate each crawler on its ability to evade server-side blocking by emulating human characteristics. Specifically, we ask the following three questions of each crawler: Does the crawler provide a mechanism to (1) restrict the rate of crawling, (2) dynamically interact with a webpage – e.g., through clicking and typing, and (3) trigger mouse movements and rate-limiting keystrokes?

Data gathering features. Features in this category capture the variety of data that a crawler might facilitate. We consider whether a crawling tool provides access to data at the network/transport layer (packet captures), application layer (HTTP requests and responses), and content layer (screenshots and content fetches). Access to each of these is critical for different problem domains – e.g., packet captures are crucial for understanding the fingerprintability of webpages [42, 79], request and response pairs are crucial for understanding how trackers and third-party content are loaded [47, 49], and screenshots of pages can be crucial for automating the process of blockpage and censorship identification [76].

Crawl-level features. Generally speaking, conducting large-scale web crawls can be an expensive task to undertake – computationally and manually. Therefore, we consider whether each crawler provides features to ease the computational and developer efforts required to use them. Specifically, we check to see if they are capable of automatically splitting workload across multiple parallel crawls (multi-instance), able to conduct crawls in headless mode – i.e., a browser with in-memory DOM rendering, and able to conduct crawls using full-fledged browsers such as Chromium and Firefox.

2.2 Results

A complete breakdown of our evaluation is shown in §2. Below, we show how the characteristics of crawling tools and their features result impact their suitability for different tasks.

Rich crawl-level and data gathering features come at the cost of protocol-level flexibility. As one might expect, ULCs generally provide the most number of crawl-level and data gathering features. This is unsurprising as ULCs are explicitly designed with the goal of simplifying data collection and facilitating parallelized BLCs. Unfortunately, this comes at a cost – the ability to manipulate lower-level details such as application-layer protocol parameters. Conversely, while ALPCs provide direct access to these parameters, the data they facilitate is incomplete due to the inability to actually render requested content or load dynamic content (since they treat web content as files and do not present a rendering or execution environment). Some BLCs offer a happy medium between these extremes – providing the flexibility to enable nearly all these features through customized scripts. *This suggests that ALPCs are more suitable for crawls which seek to uncover server behavior in response to different headers (e.g., for security fuzzing, network performance measurements, etc.) while studies which seek to measure user experiences (e.g., for censorship, privacy measurements, page load performance, etc.) need to rely on BLCs or ULCs.*

BLCs provide the richest user-level and browser-level features. BLCs which drive standard browsers offer the capability to manipulate all the user- and client-level features we considered. This is not possible with ALPCs since they do not use browser environments. Rather surprisingly, we found that the implementations of the ULCs in our study often removed access to important user- and browser-level features that are available in their underlying

Table 2: Features available in different crawling tools. '✔' indicates that the corresponding class of features is directly available through an API call or input argument. '●' indicates that the class of features is not easily available, but an equivalent action can be achieved by custom scripts. '◗' indicates that a subset of the class of features can be modified through custom scripts. '✘' indicates the inability to attain the class of features through crawler-provided resources.

	Crawl-level features			Data-gathering features			User-level features			
	Multi-instance	Headless mode	Full browser	Webpage content	Network traffic	Screen shots	Rate limit	Spider	Anti bot	Interact
`Wget` (ALPC) [35]	●	✘	✘	◗	◗	✘	✔	✔	✘	✘
`cURL` (ALPC) [18]	●	✘	✘	◗	◗	✘	✔	✘	✘	✘
ZAP Spider (ALPC) [8]	●	✘	✘	◗	✔	✘	●	●	✘	✘
PhantomJS (BLC) [59]	●	✔	✘	●	✘	✔	●	●	✘	✘
Selenium (BLC) [10]	●	✔	✔	●	✘	●	●	●	●	●
Puppeteer (BLC) [56]	●	✔	✔	●	✘	✔	●	●	●	●
OpenWPM (ULC) [50]	✔	✔	✔	✔	◗	✔	●	●	✔	●
Tor BC (ULC) [12]	✔	✔	✔	✘	✔	✔	✘	✘	✘	✘

(a) Crawler-level, data-gathering, and user-level features.

	Client-level features						(Application layer) Protocol-level features			
	Cookie options	Proxy options	Cache options	JavaScript options	Extensions	Browser config.	HTTP headers	HTTP methods	HTTP keepalive	SSL/TLS options
`Wget` (ALPC) [35]	✔	✔	✔	✘	✘	✘	✔	✔	✔	✔
`cURL` (ALPC) [18]	✔	✔	✘	✘	✘	✘	✔	✔	✔	✔
ZAP Spider (ALPC) [8]	✔	✔	✘	✘	✘	✘	◗	✘	✘	✔
PhantomJS (BLC) [59]	✔	✔	✔	✔	✘	✔	✔	✔	✘	✔
Selenium (BLC) [10]	✔	✔	●	✔	●	✔	✘	✘	✘	◗
Puppeteer (BLC) [56]	✔	✔	✔	✔	✘	✔	●	◗	●	◗
OpenWPM (ULC) [50]	✔	✘	✘	✔	✔	◗	✘	✘	✘	◗
Tor BC (ULC) [12]	✘	✘	✘	✘	✔	✘	✘	✘	✘	✘

(b) Client-level and (application layer) protocol-level features.

BLC (e.g., the proxy and caching options for both and all user-level features for TorBC). *This suggests that studies which need to avoid bot detection (e.g., content testing required in censorship and privacy measurement) and manipulating local state (e.g., network performance measurements) are more suitable for BLCs and ULCs.*

Takeaways. Our analysis hints that specific crawling tools are more suitable for use in different research domains and that the incorrect choice of crawler may significantly impact research results. For example, using an ALPC to gather network traces for input to a website fingerprinting classifier model might over-estimate the attack's success rate due to the lack of dynamic content. Similarly, studies seeking to understand censorship or differential treatment of users will likely have different results when measured using BLCs and ULCs which incorporate bot-detection mitigation. We empirically evaluate several similar hypotheses in §4.

Table 3: Crawler versions used for experimentation.

Crawler	Version	Companion tools
Wget [7]	1.17.1	
cURL [19]	7.64	
OWASP ZAP [29]	2.7	
PhantomJS [6]	2.1.1	
Selenium [11]	3.141	Firefox v66.0
Puppeteer [32]	1.12.1	Chromium 73.0
OpenWPM [28]	0.8	Selenium v3.6, Firefox v52.9
Tor BC [12]	NA	Tor Browser v8.0, Selenium v2.53

3 A COMPARATIVE FRAMEWORK

In this section, we describe our framework for empirically highlighting the differences in the network traffic generated between the crawlers described in §2. §3 shows the version information of crawlers used in this framework. Our evaluation framework focuses on uncovering the impact of crawlers on requests generated and responses received by clients and traffic characteristics observed by the network. Our hope is that this framework will enable more complete, sound, and correct research which considers and reports the impact of different crawling methodologies on their hypotheses.

3.1 Design considerations

Our framework was designed with two specific goals: extensibility and correctness. To achieve high extensibility, we seek to make our framework modular and easily configurable. Achieving correctness is more challenging due to a number of confounding factors when performing comparative crawls, including: (1) websites are frequently updated and crawls starting at different times may observe different content, (2) web servers may provide localized results making it challenging to rely on cloud infrastructure for measurements, and (3) websites might trigger IP address based discrimination against IPs suspected to be associated with crawlers. To achieve extensibility and address the above confounding factors which impact correctness, we take the following steps:

Rely on an easily configurable master-worker architecture. The current implementation of the master only requires three configuration parameters: a list of worker machines, available IP addresses from which the crawls are to be conducted, and target domains to crawl. Each worker machine only needs to implement a wrapper program which takes as input a domain name. It then crawls this domain using any crawling tool. Our first release includes wrappers for the eight different crawlers. Workers only require two configuration parameters: the master's IP address and the command line instruction to launch the local wrapper program.

Perform crawl synchronization. We make an effort to achieve a reasonable level of synchronization between crawls from different tools. Our master script coordinates efforts in such a way that all workers begin crawling a specific webpage at approximately the

same time (granularity of seconds). This allows us to negate the impact of frequent website updates within a small margin of error.

Account for changing IP reputations. To account for web servers discriminating our workers based on blacklisting and IP reputation scores, we integrate Cisco's public Talos Intelligence in our master program. Talos returns one of three classes for each queried IP address – 'Good', 'Neutral', or 'Poor'. If any worker is found to have an IP reputation that is lower than the others, a new IP address with similar reputation to the IPs used by other workers is dynamically reassigned to it. Therefore, each worker is conducting its crawl from an IP address of comparable reputation to the other worker IP addresses. A limitation of our approach is that we rely on intelligence data from a single source – Talos [16]. However, our IP intelligence class can be easily extended to use APIs from other intelligence services.

3.2 Deployment for this study

Master and worker machine setup. We used one virtual machine for each worker and master – each with a unique IP address. In total, we had eight workers – one for each crawler. Each worker was configured to use its crawling tool in its default configuration, with the exception of OpenWPM for which we also enabled its "anti-bot detection" functions.

IP addresses. We obtained 22 unique IPv4 addresses which were used in our crawl. These IPv4 addresses were obtained from a regional ISP for the duration of these experiments and were found to have a "Good" IP reputation at the start of our measurements. Our IP addresses were geolocated to the same region thereby reducing the impact of content localization.

URL list. Our URLs included the union of the Top 500 domains from Alexa [14] and Umbrella's [17] top sites lists. In total, there were 932 unique domains (of which 30 domains were removed as they returned SOA records) which were crawled by each of our workers. The domain lists were fetched on 25^{th} April 2019.

4 QUANTITATIVE EVALUATION

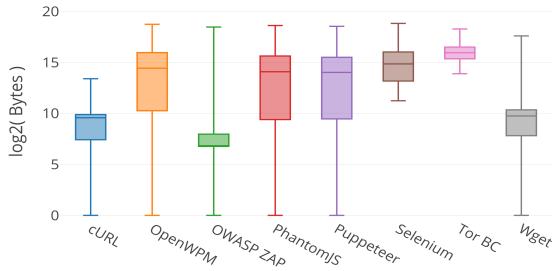
In this section, we empirically demonstrate how different crawling tools impact measurement results and their resulting inferences. The analysis in this section is based on the data generated by our comparative framework (§3) using the parameters described in §3.2.

4.1 Generic data differences

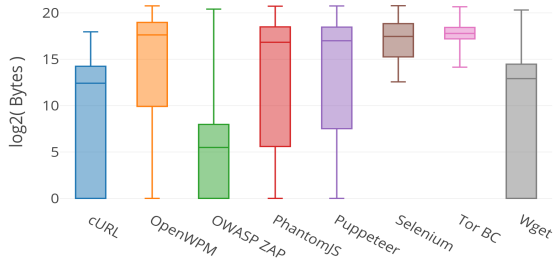
We start by analyzing the differences in data generated by each of our studied crawlers. We analyze these differences in two distinct vantage points: the differences observed by end hosts (e.g., requests generated by clients and responses received from servers) and the differences observed by the network (e.g., flow characteristics).

Host-observed differences. Our analysis shows that different crawler classes (ALPC, BLC, and ULC) cause different characteristics in requests and responses. We highlight two of these below.

ALPC request and response characteristics are drastically different from BLCs and ULCs. As one might expect, the inability of ALPCs to load dynamic content makes the characteristics of their requests and responses very different from those generated by BLCs and ULCs. Our analysis that the three ALPCs (`Wget`, `cURL`, and ZAP) in our study have several orders of magnitude fewer destinations that they send requests to, in comparison to BLCs and ULCs, while loading a webpage. Further, the sizes of these requests (shown in §1a) and the corresponding responses (shown in §1b) are also orders of magnitude smaller. In comparison, we see smaller variations between different ULCs and BLCs. We attribute these variations to the differences in the browsers (and the browser versions) that they drive – suggesting that crawler specifications are important for accurate reproduction of research and measurement findings. These findings further suggest that ALPCs, despite their favorable performance, are not suitable for content analysis or third-party ecosystem measurements.



(a) Sizes of requests generated.



(b) Sizes of responses received.

Figure 1: Host-observed differences between crawlers.

Ciphersuite measurements are crawler dependant. We find that there are not only notable differences in the ciphersuites made available by each crawler during TLS negotiation, but also noticeable differences in the negotiated ones. When simply considering the availability of ciphersuites, we see that ALPCs once again stand out (offering over 50 different ciphersuites in TLS negotiation) in comparison to ULCs and BLCs which typically offer less than 20. It is important to note that while the suites offered by ULCs and BLCs are simply those offered by the browser driven by them, there are still significant differences since not all crawlers drive the same browser versions. For example, at the time of our experiments OpenWPM was unable to drive Firefox versions later than v52.9, while Selenium drove Firefox v66.0. Both browser releases were over a year apart and support quite different ciphersuites. The fraction of overlapping ciphersuites offered by the latest releases of our crawlers is shown in §2. As expected, this difference in ciphersuite support has an impact on the finally negotiated TLS ciphersuite with servers. Measuring the usage of different ciphersuites has been utilized in prior studies to uncover the usage of broken cryptographic schemes [63], understand energy consumption [67], and identify how censorship circumvention traffic may better blend in with

uncensored traffic [53]. Our findings suggest that such research needs to account for the fact that their conclusions may differ based on the choice of crawler (and consequently the browser/version) used in their measurements.

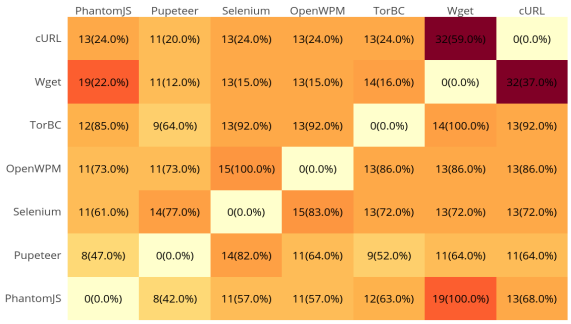


Figure 2: Ciphersuite Overlap amongst crawlers.

Network-observed differences. In addition to differences at the end-hosts, we find that the choice of crawler also has an impact on the traffic flow characteristics observed by the network. Specifically, we see that: *burst characteristics vary by crawler.* At the network-layer a burst is characterized by a sequence of consecutive packets (or bytes) all flowing in the same direction. Our analysis shows that the size of bursts are crawler dependent. Specifically, we find that ALPCs show much lower variations in burst sizes while more complex crawlers show much higher variation. TorBC is an exception since it relies on the Tor Browser which uses padding to prevent traffic analysis. Our finding makes intuitive sense: crawlers operating complete browsers are likely to have more bursts and variations in their sizes due to dynamic content on webpages. Further, we also see small variations in the mean burst sizes across different BLCs. This finding is particularly relevant to traffic analysis and TLS fingerprinting studies seeking to identify different classes of traffic based on network flows [51, 55, 74] since it shows that classifying traffic generated by different crawlers may yield different results.
Crawl performance characteristics vary by crawler. The rate of page requests is an important aspect to consider when selecting a crawler. As one might expect, the crawling rate (pages loaded per hour) is highest for ALPCs and lowest for ULCs. The differences between crawler page-load times is shown in §3. Each point represents a successful page-load with a response status of 200 and its corresponding x- and y-axis values denote the time-to-load and bytes transferred. We see that ALPCs occupy the bottom left of the plot and ULCs occupy the top right corner. This plot, although difficult to inspect, hints at the inherent trade-offs at play when selecting a crawler: ALPCs provide high crawling performance at the cost of completeness while ULCs provide completeness at the cost of performance.

4.2 Impact on research inferences

Our previous results (§4.1) have shown that crawling method can have a significant impact on lower-level measurements pertaining to characteristics of client-generated requests, server-generated

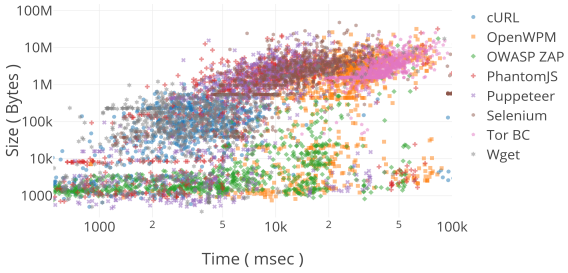


Figure 3: Distribution for crawling time vs content size.

responses, and network flows. In this section, we seek to understand how these differences can impact higher-level measurements which seek to characterize the state of the web. Specifically, we use data generated from our crawling framework to test the following hypotheses: (H_{wf}) Choice of crawling method impacts the fingerprintability of websites, (H_{tp}) choice of crawling method impacts the measurements of prevalence and prominence of online tracking third-parties, and (H_{sb}) choice of crawling method impacts measurements seeking to uncover server-side blocking (e.g., for censorship or discrimination measurements).

Table 4: WF accuracy on traces generated by different crawlers.

	NB	MB	JS
Wget	95%	94%	97%
cURL	96%	92%	97%
OWASP ZAP	90%	85%	95%
PhantomJS	92%	89%	95%
Selenium	86%	72%	91%
Puppeteer	90%	89%	93%
OpenWPM	92%	90%	96%
TOR BC	2%	5%	4%

Hypothesis H_{wf}: Choice of crawling method impacts the fingerprintability of websites. In order to validate this hypothesis, we need to show that the success rates of website fingerprinting classifiers are impacted by the crawling method used to generate the data input to the classifiers.
Methodology. We selected three simple classifiers from previous website fingerprinting research in our study: a Naive Bayes (NB) classifier [64], a Multinomial Bayes (MB) classifier [57], and a Jaccard Similarity (JS) classifier [71]. We note that although these classifiers are not the current state-of-the-art, they are sufficient to prove our hypothesis. Implementations of these classifiers were obtained from [58, 78]. Data was generated by our crawling framework. The URL list (described in §3) was crawled through each crawler 10 times and traces for each domain and each iteration were logged. These traces were split into training (70%) and testing (30%) datasets and used as input to our classifiers.
Results. Our results are highlighted in §4. Here we see that TorBC has the lowest fingerprintability since it has explicit defenses to prevent such attacks. However, amongst other crawlers we see that there is between a 2-22% difference in accuracy for the same fingerprinting classifiers and only changing the crawling tool. While this

is sufficient to prove our hypothesis, we can show even more significant implications of not using uniform crawler configurations when conducting evaluations of classifier effectiveness. Consider a comparison of the JS and NB classifier both evaluated using OpenWPM – we see that the JS classifier has a success rate of 96% and the NB classifier has a success rate of 92%. This leads us to conclude that the JS classifier is more effective at launching website fingerprinting attacks on the specific list of domains. However, by evaluating the JS classifier using Selenium or the NB classifier with any ALPC we might conclude the opposite. It is important to note that while our toy experiment does not go into comprehensive evaluations of the state-of-the-art, the implications of not specifying the crawling methodology or not maintaining consistent methodology in traffic analysis research are still significant and valid.

Hypothesis H_{tp}: Choice of crawling method impacts the measurements of prevalence and prominence of online tracking third-parties. In order to validate this hypothesis, we need to show that the prevalence and prominence rankings of embedded third-party advertising and tracking domains vary by crawler.

Methodology. In order to identify third-party advertising and tracking related domains we rely on EasyList and EasyPrivacy [20]. While these lists may be incomplete, they must only show that results per crawler change. We execute the rules in these lists to identify counts of requests to third-party advertising and tracking domains. We then rank these domains by two metrics for each crawler: *prevalence* and *prominence*. We borrow metrics for prominence and prevalence from Englehardt et al. [49]. The general idea is that rankings of prevalence capture the exposure of different trackers to the web – since it is assumed that all sites are equally likely to be visited by users. On the other hand, rankings of prominence capture the exposure of different trackers to *actual users* – since it is assumed that users are likely to visit popular domains more than less popular domains (i.e., the power law distribution applies to website popularity). Finally, we compare the differences in the prevalence and prominence rankings obtained by data generated from each crawler. We note that our results are impacted by our inability to measure the inclusion chains of third-party resources for all crawlers – a prohibitively expensive proposition. This might introduce variability across crawls due to real-time bidding (RTB) [69] and header bidding [45]. We instead try to mitigate the impact of this noise by relying on data gathered by multiple crawl iterations from each crawler.

Results. Our results are shown in §4. These results showcase how the measured prevalence and prominence of different trackers vary when computed on data generated from different crawlers. For both results, we use the data generated by the OpenWPM crawler as a baseline for comparison since it was explicitly built for the purpose of conducting privacy measurements. Specifically, §4a shows how the prevalence of different trackers may be over- or under-estimated when compared to the baseline (OpenWPM) – e.g., ALPCs generally significantly underestimate the presence of trackers from `sohu.com` since these are often dynamically loaded while statically loaded trackers such as those from `casalmedia.com` are overestimated. According to §4b, BLCs show varying results when compared to the baseline, suggesting that the underlying browser has greater implications on third-party tracking as opposed to the crawler type. For example, we see that Firefox-driven Selenium does fairly similar in comparsion to the, also Firefox-driven, baseline, while the Chrome-driven Puppeteer and the V8/Gecko-less PhantomJS differ significantly.

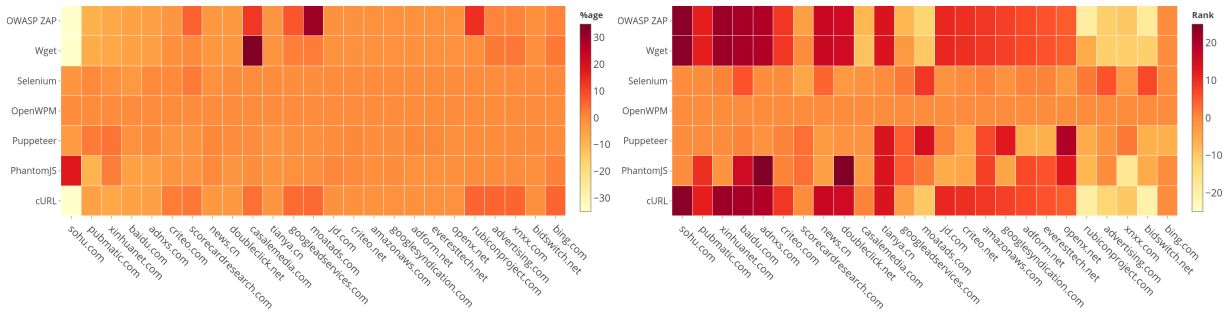
Table 5: Types of block pages observed by different crawlers. **Success** denotes the fraction of pages that were successfully loaded, **4XX** denotes the rate of 4xx status code errors, **GEO** denotes the rate of regional blocking, **BRO** is the rate of browser errors, **CAP** is the rate of CAPTCHAs, and **ABU** denotes the rate of IP-abuse based blockpages. Note that ZAP Spider does not capture HTML files, hence we fail to capture blockpage HTMLs.

	Success	4XX	GEO	BRO	CAP	ABU
Wget	82.94%	5.36%	2.68%	0.43%	9.01%	0.00%
cURL	82.62%	4.61%	2.68%	0.43%	10.09%	0.00%
OWASP ZAP	90.34%	9.66%	-	-	-	-
PhantomJS	81.97%	6.00%	0.54%	0.54%	11.48%	0.00%
Selenium	78.22%	7.40%	2.68%	0.54%	11.59%	0.00%
Puppeteer	77.04%	8.69%	2.79%	0.54%	11.16%	0.00%
OpenWPM	93.13%	7.51%	2.68%	0.54%	10.94%	0.00%

Hypothesis H_{sb}: Choice of crawling method impacts measurements of server-side blocking. In order to validate this hypothesis, we need to show that different crawling methods incur different levels of server-side blocking.

Methodology. In order to identify server-side blocking, we rely on methodologies constructed from prior work [60, 68]. Based on previous repositories of block pages [1, 2, 68] and manually identified block pages observed by our own crawls, we crafted generalized regular expressions to identify occurrences of the same or other block pages. Our regular expressions were crafted to identify blocking and discrimination due to HTTP 4XX status errors (i.e., client-side errors typically returned for unauthorized or unexpected client interactions with servers), presence of CAPTCHAs (typically returned when bot activity is suspected), out-dated browser errors (returned when websites cannot be loaded or rendered correctly due to browser characteristics), geo-blocking (i.e., when websites block traffic originating at different regions) [66], and IP abuse (i.e., when the server suspects that the client is behaving maliciously). It is important to note that it is common for servers to simply return HTTP 4XX errors rather than more specific blockpages.

Results. Our results are shown in §5. Here we see obvious differences in the way servers respond to requests generated by different crawlers – i.e., we have a variation of over 16% in the number of successful page loads. This immediately validates our hypothesis that different crawling tools can significantly impact the inferences drawn from content accessibility measurements. In fact, simply changing our crawling tool, we are able to uncover over 160 sites which actually do implement some form of server-side blocking. Further inspecting the server-provided reasons for blocking, we see that a small fraction of servers consistently provide detailed reasons for blocking – regardless of crawler choice, while a majority simply return an HTTP 4XX status error. Rather surprisingly, we see that ALPCs are not blocked as frequently as previously expected, *with the lowest numbers of both 4XX-Errors and CAPTCHA-based blocking*. We hypothesize that this is because popular bot detection tools require JS execution (e.g., Distil Networks [15]). This suggests that they are suitable for measurements of content reachability, which do not require dynamic loading or user interactions. We also see



(a) Prevalence of third party trackers.

(b) Prominence of third party trackers.

Figure 4: Prevalence and prominence rankings of third-party advertising and tracking domains identified by each crawler. The OpenWPM crawler is used as the baseline ranking and darker and lighter colors show how much or how little results from other crawlers deviated from this baseline.

OpenWPMs bot-mitigation method results in *the least instances of CAPTCHA-based blocking among the BLCs and ULCs*, thus highlighting server-side blocking against bot-like behavior and also the potential use of OpenWPM as a viable circumvention tool. Note that we do not see instances of IP-Abuse based blockpages, indicating that the IP-reputation checks and dynamic reassignment in §3 is successful in removing IP-reputation as a confounding variable.

Takeaways. Our quantitative analysis shows that the data generated by different crawlers clearly impacts lower-level measurements (§4.1) and that crawlers can also impact inferences from higher-level measurement (e.g., H_{wf}, H_{tp} and H_{sb}). These findings highlight that researchers must not only consider crawler categories, but must also consider the underlying browser-engine used by the crawler, as that have a significant influence on the results generated.

5 RELATED WORK

As shown in our literature review (§2), the use of web crawlers in academic research is ubiquitous, and there have been numerous works seeking to improve our understanding of the implications of different crawling approaches. While Yadav and Goyal [81] focus on the comparison of 12 open-source crawlers, following the same direction as our work for determining suitability of each tool, more specifically and, similar to Ricardo and Serrao [73], their work is limited to comparative analysis of crawlers in terms of scalability and performance. Avraam et al. [40] review and compare focused-web-crawling techniques by examining the proceedings of multiple web conferences. Glez-Pena et al. [54] assess existing commercial scraping frameworks by identifying their strengths and limitations in terms of data extraction capabilities, under the umbrella of biomedical applications. Kumar et al. [61] carry out a large literature survey to identify and compare types of crawling strategies, techniques and policies for information retrieval through search engine crawlers. Likewise, Anu et al. [39] survey and compare various crawling techniques specifically for mining web forums. Becchetti et al. [41] studied and compared the bias induced by different methods of web sampling. While all these studies carry out various comparisons of crawling methods at some level, our contributions differ by providing both qualitative and quantitative comparison of crawling methodologies and focusing on how their use can directly impact various research inferences.

More recently, we have seen works that address reproducibility concerns. Flittner et al. [52] conducted an analysis of papers from four different ACM conferences and assessed the state of artifact availability. They highlight that the majority of the analyzed papers do not provide the complete toolset required to reproduce all results and stressed upon the release of artifacts by the authors in the measurement community. Work by Scheitle et al. [75] indicates how internet top lists show considerable churn across multiple snapshots in time, impacting replicability of results. Collberg et al. [44] survey a number of computer systems papers to examine the repeatability of the experiments. They focus on their ability to locate, build, and run the system prototypes, and develop metrics that help assess the degree to which certain works are more repeatable than others. Elmenreich et al. [48] carry out an online survey to explore the need for reproducibility and provide guidelines on making research results reproducible.

6 DISCUSSION

Limitations. Our survey, qualitative, and quantitative studies each come with their own limitations. Some of these arise from our inability to perform large scale manual analysis and others from our inability to make observations without interfering with natural web interactions. First, due to the scale of our survey, we rely on computational filtering heuristics to narrow down the set of publications to study. This introduces the possibility of missing important insights into how crawlers are used by the academic community. Our focus on a few select conferences does not invalidate the insights provided about how certain crawlers are used and the crawl-repeatability and specificity challenges faced by the research community. Second, our qualitative analysis of crawlers broadly categorizes parameters exposed by different crawlers. This parameter generalization introduces the possibility of generalizing crawler (in)capabilities to the point of missing analysis of certain features that are important to specific communities – e.g., we do not explicitly verify the possibility of the crawler having or granting access to connected devices (e.g., cameras and microphones) – the topic of emerging studies in the privacy domain. Finally, our empirical analysis is complicated by several confounds: first, we are impacted by the several dynamic and real-time processes on the web (e.g., real-time bidding). Further, it is possible that our data has

sources of bias and inconsistency since we currently only rely on IP intelligence data from Cisco's Talos. We enable other developers with access to more threat streams to circumvent this limitation by providing an easy to use extended API. Despite these limitations, our framework replicates and in many cases pushes the boundaries of the state-of-the-art in crawling.

Recommendations. Despite the above limitations, our study allows us to make recommendations aimed at improving the repeatability and correctness of research relying on data generated by web crawling tools. First, we urge researchers and reviewers to enforce standards of crawl specificity to improve repeatability – e.g., at the very least, require mention of the tool, tool version, and (if applicable) browser version. Currently, over one-third of all crawler-dependant research do not mention even the name of the tool used for data gathering. Second, research communities need to understand how crawling tools might impact their inferences – particularly when inferences are drawn over multiple studies, each using a different crawling methodology. As shown in our analysis of H_{wf}, H_{tp} and H_{sb}, this can lead to an incorrect understanding of the research domain. Therefore, we recommend that researchers aim to replicate exactly the crawling methodologies of prior work when proposing advancements to the problem domain. Alternately, there is value in evaluating results and analysis on datasets generated by multiple crawlers to ensure the correctness of inferences. To facilitate our call for repeatability and cross-crawler evaluation, we are releasing our crawling framework at `https://sparta.cs.uiowa.edu/projects/methodologies.html`.

Acknowledgements. We would like to thank our colleagues – Danyal Saeed Mirza, Hammas Bin Tanveer, Hussam Habib, John Thiede, Maaz Bin Musa, Muhammad Hassan, Muhammad Haroon, Ralph Nahra, Xiaoyu Xing, Yao Wang, and Zain Humayun – for their help in our survey. We also thank the anonymous reviewers whose valuable comments helped shaped the paper to it's final form. This work was partially funded by the Spanish national grant TIN2017- 88749-R (DiscoEdge) and the Region of Madrid EdgeData-CM program (P2018/TCS-4499).

REFERENCES

[1] 2008. Blockpages Archived. https://advox.globalvoices.org/past-projects/blockpages/
[2] 2015. Collection of censorship blockpages as collected by various sources. https://github.com/citizenlab/blockpages
[3] 2015. ghost.py - Ghost.py 0.2.2 documentation. https://ghost-py.readthedocs.io/en/latest/
[4] 2015. grub.org - Distributed Internet Crawler download | SourceForge.net. https://sourceforge.net/projects/grub/
[5] 2016. Fast and powerful scraping and web crawling framework. https://scrapy.org/
[6] 2016. PhantomJS v2.1.1. https://phantomjs.org/release-2.1.html
[7] 2017. Wget v1.17.1. https://packages.ubuntu.com/xenial/wget
[8] 2018. OWASP ZAP Spider. https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
[9] 2018. Raccoon: reconnaissance and vulnerability scanning. https://www.prodefence.org/raccoon-reconnaissance-and-vulnerability-scanning/
[10] 2018. Selenium - Web Browser Automation. http://www.seleniumhq.org/
[11] 2018. Selenium v3.141.0. https://rubygems.org/gems/selenium-webdriver/versions/3.141.0
[12] 2018. Tor-Browser-Crawler. https://github.com/webfp/tor-browser-crawler
[13] 2019. ACM CCS. https://www.sigsac.org/ccs.html
[14] 2019. Alexa Top Sites. https://docs.aws.amazon.com/AlexaTopSites/latest/ApiReference_TopSitesAction.html
[15] 2019. Block Bots with Bot Mitigation and Detection Tools | Distil Networks. https://www.distilnetworks.com/block-bot-detection/
[16] 2019. Cisco Talos Intelligence Group - Comprehensive Threat Intelligence. https://www.talosintelligence.com/
[17] 2019. Cisco Umbrella 1 Million - OpenDNS Umbrella Blog. https://umbrella.cisco.com/blog/2016/12/14/cisco-umbrella-1-million/
[18] 2019. Curl - Command Line Tool. https://curl.haxx.se/
[19] 2019. cURL v7.64.0. https://curl.haxx.se/changes.html#7_64_0
[20] 2019. EasyList - Overview. https://easylist.to/
[21] 2019. GNU Wget 1.20 Manual. https://www.gnu.org/software/wget/manual/wget.html#Robot-Exclusion
[22] 2019. IEEE Symposium on Security and Privacy 2019. https://www.ieee-security.org/TC/SP2019/
[23] 2019. IMC Conference | acm sigcomm. https://www.sigcomm.org/events/imc-conference
[24] 2019. International Conference on Web and Social Media Papers (ICWSM). http://www.aaai.org/Library/ICWSM/icwsm-library.php
[25] 2019. International World Wide Web Conference Committee (WWW). https://www.iw3c2.org
[26] 2019. internetarchive/heritrix3: Heritrix is the Internet Archive's open-source, extensible, web-scale, archival-quality web crawler project. https://github.com/internetarchive/heritrix3
[27] 2019. NDSS Symposium - The Network and Distributed System Security Symposium (NDSS). https://www.ndss-symposium.org/
[28] 2019. OpemWPM v0.8.0. https://github.com/mozilla/OpenWPM/blob/b3ead7e38892095950806e8bcbb2e1129c27ca96/VERSION
[29] 2019. OWASP ZAP v2.7.0. https://github.com/zaproxy/zap-core-help/wiki/HelpReleases2_7_0
[30] 2019. OWASP Zed Attack Proxy Project - OWASP. https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
[31] 2019. PoPETs/PETS. https://petsymposium.org/
[32] 2019. Puppeteer v1.12.1. https://github.com/GoogleChrome/puppeteer/releases
[33] 2019. RedTeam Pentesting on Twitter. https://twitter.com/RedTeamPT/status/1110843396657238016
[34] 2019. USENIX Security Symposia | USENIX. https://www.usenix.org/conferences/byname/108
[35] 2019. Wget - Command Line Tool. https://www.gnu.org/software/wget/
[36] 2019. zmap/zgrab2: Go Application Layer Scanner. https://github.com/zmap/zgrab2
[37] 2020. Javascript Usage Statistics. https://trends.builtwith.com/docinfo/Javascript
[38] Athanasios Andreou, Giridhari Venkatadri, Oana Goga, Krishna P. Gummadi, Patrick Loiseau, and Alan Mislove. 2018. Investigating Ad Transparency Mechanisms in Social Media: A Case Study of Facebook's Explanations. https://doi.org/10.14722/ndss.2018.23204
[39] Anu. 2015. A Survey on Web Forum Crawling Techniques.
[40] Ioannis Avraam and Ioannis Anagnostopoulos. 2011. A Comparison over Focused Web Crawling Strategies. In *2011 15th Panhellenic Conference on Informatics*. 245–249. https://doi.org/10.1109/PCI.2011.53
[41] Luca Becchetti, Luca, Carlos Castillo, Carlos, Debora Donato, Debora, Fazzone, and Adriano. 2006. A comparison of sampling techniques for Web characterization.
[42] Xiang Cai, Rishab Nithyanand, Tao Wang, Rob Johnson, and Ian Goldberg. 2014. A systematic approach to developing and evaluating website fingerprinting defenses. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 227–238.
[43] Neha Chachra, Stefan Savage, and Geoffrey M. Voelker. 2015. Affiliate Crookies: Characterizing Affiliate Marketing Abuse. In *Proceedings of the 2015 Internet Measurement Conference (IMC '15)*. ACM, New York, NY, USA, 41–47. https://doi.org/10.1145/2815675.2815720
[44] Christian Collberg and Todd A. Proebsting. 2016. Repeatability in Computer Systems Research. *Commun. ACM* 59, 3 (Feb. 2016), 62–69. https://doi.org/10.1145/2812803
[45] John Cook, Rishab Nithyanand, and Zubair Shafiq. 2019. Inferring Tracker-Advertiser Relationships in the Online Advertising Ecosystem using Header Bidding. *CoRR* abs/1907.07275 (2019). arXiv:1907.07275 http://arxiv.org/abs/1907.07275
[46] Dominik Dary. 2019. Selendroid: Selenium for Android. http://selendroid.io/
[47] Anupam Das, Gunes Acar, Nikita Borisov, and Amogh Pradeep. 2018. The Web's Sixth Sense: A Study of Scripts Accessing Smartphone Sensors. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS '18)*. ACM, New York, NY, USA, 1515–1532. https://doi.org/10.1145/3243734.3243860
[48] Wilfried Elmenreich, Philipp Moll, Sebastian Theuermann, and Mathias Lux. 2019. Making simulation results reproducible-Survey, guidelines, and examples based on Gradle and Docker. *PeerJ Computer Science* 5 (12 2019), e240. https://doi.org/10.7717/peerj-cs.240
[49] Steven Englehardt and Arvind Narayanan. 2016. Online Tracking: A 1-million-site Measurement and Analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, New York, NY, USA, 1388–1401. https://doi.org/10.1145/2976749.2978313
[50] Steven Englehardt and Arvind Narayanan. 2018. OpenWPM - Web Privacy Measurement Framework. https://github.com/mozilla/OpenWPM
[51] Jeffrey Erman, Martin Arlitt, and Anirban Mahanti. 2006. Traffic Classification Using Clustering Algorithms. In *Proceedings of the 2006 SIGCOMM Workshop on Mining Network Data (MineNet '06)*. ACM, New York, NY, USA, 281–286. https://doi.org/10.1145/1162678.1162679
[52] Matthias Flittner, Mohamed Naoufal Mahfoudi, Damien Saucez, Matthias Wählisch, Luigi Iannone, Vaibhav Bajpai, and Alex Afanasyev. 2018. A Survey on Artifacts from CoNEXT, ICN, IMC, and SIGCOMM Conferences in 2017. *SIGCOMM Comput. Commun. Rev.* 48, 1 (April 2018), 75–80. https://doi.org/10.1145/3211852.3211864
[53] Sergey Frolov and Eric Wustrow. 2019. The use of TLS in Censorship Circumvention. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. https://www.ndss-symposium.org/ndss-paper/the-use-of-tls-in-censorship-circumvention/
[54] Daniel Glez-Peña, Anália Lourenço, Hugo López-Fernández, Miguel Reboiro-Jato, and Florentino Fdez-Riverola. 2013. Web scraping technologies in an API world. *Briefings in Bioinformatics* 15, 5 (04 2013), 788–797. https://doi.org/10.1093/bib/bbt026 arXiv:https://academic.oup.com/bib/article-pdf/15/5/788/17488715/bbt026.pdf
[55] Roberto Gonzalez, Claudio Soriente, and Nikolaos Laoutaris. 2016. User Profiling in the Time of HTTPS. In *IMC '16*.
[56] Google. 2019. Chrome Puppeteer. https://github.com/GoogleChrome/puppeteer
[57] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. 2009. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial NaïVe-bayes Classifier. In *Proceedings of the 2009 ACM Workshop on Cloud Computing Security (CCSW '09)*. ACM, New York, NY, USA, 31–42. https://doi.org/10.1145/1655008.1655013

[58] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. 2009. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *Proceedings of the 2009 ACM workshop on Cloud computing security*. ACM, 31–42.

[59] Ariya Hidayat. 2018. PhantomJS - Headless Web Browser. http://phantomjs.org

[60] Ben Jones, Tzu-Wen Lee, Nick Feamster, and Phillipa Gill. 2014. Automated Detection and Fingerprinting of Censorship Block Pages. In *Proceedings of the 2014 Conference on Internet Measurement Conference (IMC '14)*. ACM, New York, NY, USA, 299–304. https://doi.org/10.1145/2663716.2663722

[61] Manish Kumar, Rajesh Bhatia, and Dhavleesh Rattan. 2017. A survey of Web crawlers for information retrieval. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 7 (08 2017). https://doi.org/10.1002/widm.1218

[62] Ada Lerner, Anna Kornfeld Simpson, Tadayoshi Kohno, and Franziska Roesner. 2016. 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, Austin, TX. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/lerner

[63] Olivier Levillain, Arnaud Ébalard, Benjamin Morin, and Hervé Debar. 2012. One Year of SSL Internet Measurement. In *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC '12)*. ACM, New York, NY, USA, 11–20. https://doi.org/10.1145/2420950.2420953

[64] Marc Liberatore and Brian Neil Levine. 2006. Inferring the Source of Encrypted HTTP Connections. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS '06)*. ACM, New York, NY, USA, 255–263. https://doi.org/10.1145/1180405.1180437

[65] Allison McDonald, Matthew Bernhard, Luke Valenta, Benjamin VanderSloot, Will Scott, Nick Sullivan, J. Alex Halderman, and Roya Ensafi. 2018. 403 Forbidden: A Global View of CDN Geoblocking. In *Proceedings of the Internet Measurement Conference 2018 (IMC '18)*. ACM, New York, NY, USA, 218–230. https://doi.org/10.1145/3278532.3278552

[66] Allison McDonald, Matthew Bernhard, Luke Valenta, Benjamin VanderSloot, Will Scott, Nick Sullivan, J. Alex Halderman, and Roya Ensafi. 2018. 403 Forbidden: A Global View of CDN Geoblocking. In *IMC '18*.

[67] Pedro Miranda, Matti Siekkinen, and Heikki Waris. 2011. TLS and energy consumption on a mobile device: A measurement study. In *2011 IEEE Symposium on Computers and Communications (ISCC)*. 983–989. https://doi.org/10.1109/ISCC.2011.5983970

[68] Arian Akhavan Niaki, Shinyoung Cho, Zachary Weinberg, Nguyen Phong Hoang, Abbas Razaghpanah, Nicolas Christin, and Phillipa Gill. 2019. ICLab: A Global, Longitudinal Internet Censorship Measurement Platform. *CoRR* abs/1907.04245 (2019). arXiv:1907.04245 http://arxiv.org/abs/1907.04245

[69] Panagiotis Papadopoulos, Nicolas Kourtellis, Pablo Rodriguez Rodriguez, and Nikolaos Laoutaris. 2017. If You Are Not Paying for It, You Are the Product: How Much Do Advertisers Pay to Reach You?. In *Proceedings of the 2017 Internet Measurement Conference (IMC '17)*. ACM, New York, NY, USA, 142–156. https://doi.org/10.1145/3131365.3131397

[70] Watir Project. 2019. Watir Project. http://watir.com/

[71] Qixiang Sun, D. R. Simon, Yi-Min Wang, W. Russell, V. N. Padmanabhan, and Lili Qiu. 2002. Statistical identification of encrypted Web browsing traffic. In *Proceedings 2002 IEEE Symposium on Security and Privacy*. 19–30. https://doi.org/10.1109/SECPRI.2002.1004359

[72] Abbas Razaghpanah, Anke Li, Arturo Filasto, Rishab Nithyanand, Vasilis Ververis, Will Scott, and Phillipa Gill. 2016. Exploring the design space of longitudinal censorship measurement platforms. *arXiv preprint arXiv:1606.01979* (2016).

[73] Andre Ricardo and Carlos Serrao. 2013. Comparison of existing open source tools for web crawling and indexing of free music. In *Journal of Telecommunications Vol. 18*.

[74] Roberto Gonzalez Sanchez, Claudio Soriente, and Nikolaos Laoutaris. 2017. Method for performing user profiling from encrypted network traffic flows. US Patent App. 15/486,318.

[75] Quirin Scheitle, Oliver Hohlfeld, Julien Gamba, Jonas Jelten, Torsten Zimmermann, Stephen D. Strowes, and Narseo Vallina-Rodriguez. 2018. A Long Way to the Top: Significance, Structure, and Stability of Internet Top Lists. In *Proceedings of the Internet Measurement Conference 2018 (IMC '18)*. Association for Computing Machinery, New York, NY, USA, 478–493. https://doi.org/10.1145/3278532.3278574

[76] Rachee Singh, Rishab Nithyanand, Sadia Afroz, Paul Pearce, Michael Carl Tschantz, Phillipa Gill, and Vern Paxson. 2017. Characterizing the Nature and Dynamics of Tor Exit Blocking. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 325–341. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/singh

[77] Nicolas Viennot, Edward Garcia, and Jason Nieh. 2014. A Measurement Study of Google Play. In *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '14)*. ACM, New York, NY, USA, 221–233. https://doi.org/10.1145/2591971.2592003

[78] Tao Wang. 2017. File directory: Website fingerprinting attacks. https://www.cse.ust.hk/~taow/wf/attacks/

[79] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective attacks and provable defenses for website fingerprinting. In *23rd {USENIX} Security Symposium ({USENIX} Security 14)*. 143–157.

[80] Zachary Weinberg, Mahmood Sharif, Janos Szurdi, and Nicolas Christin. 2017. Topics of Controversy: An Empirical Analysis of Web Censorship Lists. *PoPETs* 2017, 1 (2017), 42–61.

[81] Monika Yadav and Neha Goyal. 2015. Comparison of Open Source Crawlers-A Review.

[82] Nicholas Yuan, Fuzheng Zhang, Defu Lian, Kai Zheng, Siyu Yu, and Xing Xie. 2013. We know how you live: Exploring the spectrum of urban lifestyles. *COSN 2013 - Proceedings of the 2013 Conference on Online Social Networks*, 3–14. https://doi.org/10.1145/2512938.2512945