How Tracking Companies Circumvent Ad Blockers Using WebSockets

Muhammad Ahmad Bashir, Sajjad Arshad, Engin Kirda, William Robertson, Christo Wilson
Northeastern University
{ahmad, arshad, ek, wkr, cbw}@ccs.neu.edu

***Abstract*—In this study of 100,000 websites, we document how Advertising and Analytics (*A&A*) companies have used WebSockets to bypass ad blocking, exfiltrate user tracking data, and deliver advertisements. Specifically, we leverage a long-standing bug in Chrome (the world's most popular browser) in the `chrome.webRequest` API that prevented blocking extensions from being able to interpose on WebSocket connections. We conducted large-scale crawls of top publishers before and after this bug was patched in April 2017 to examine which A&A companies were using WebSockets, what information was being transferred, and whether companies altered their behavior after the patch. We find that a small but persistent group of A&A companies use WebSockets, and that several of them are engaging in troubling behavior, such as browser fingerprinting, exfiltrating the DOM, and serving advertisements, that would have circumvented blocking due to the Chrome bug.**

I. Introduction

The use of techniques to block online ads and prevent tracking on the web has proliferated in recent years. Measurements studies estimate that Adblock Plus is used by roughly 16–37% of web users [41], [30], and numerous other extensions like Ghostery, Disconnect, Privacy Badger, and uBlock Origin have devoted user bases.

In response to the proliferation of blocking and privacy tools, online Advertising and Analytics (A&A) companies have fought back in a variety of ways. This includes industry self-regulation such as the Ad Choices initiative [3], as well as technological mechanisms like anti-adblocking scripts [33], [36]. Most alarmingly, some companies have attempted to circumvent privacy tools, with the most famous case being Google's evasion of Safari's third-party cookie blocking policy, which resulted in a $22.5M settlement with the FTC [12].

In August 2016, privacy conscious users began to notice ads appearing on specific websites on Chrome, despite the use of ad blocking extensions [21], [42]. Online sleuths determined that (1) these ads were being downloaded via WebSockets because (2) a long dormant bug in the `chrome.webRequest` API in Chromium [23] allowed WebSocket connections to bypass ad blocking extensions. We refer to this issue as the webRequest Bug (WRB). Google patched the WRB in Chrome 58, released on April 19, 2017 [40].

In this paper, we study the behavior of A&A companies with respect to the WRB. Prior to April 19, 2017, there existed a five year window in which blocking extensions in Chrome (the world's most popular web browser [48]) could be circumvented through the use of WebSockets. We ask the following questions: which A&A companies, if any, decided to leverage this bug? Similarly, after the release of Chrome 58, did A&A companies continue to use WebSockets, or did they revert to HTTP/S? These questions are important, as they speak to the lengths that A&A companies are willing to go to track users and monetize impressions.

To answer these questions, we performed four crawls of the top Alexa websites: two just prior to the release of Chrome 58, and two after. Our crawls were conducted using stock Chrome coupled with custom instrumentation to record the *inclusion tree* of resources within each webpage [4], [7], [29] (see section III for details).

Using this data, we make the following key findings:

- Although we find that WebSocket usage is rare across the web ($\sim$2% of publishers), 55–61% of WebSockets are related to tracking and advertising in some way. Furthermore, we find that A&A sockets are more prevalent on Alexa top-10K publishers.
- We observe 91 A&A domains initiating and 17 A&A domains receiving WebSocket connections, including some of the largest players in the online advertising ecosystem (e.g., Google and Facebook).
- We find that the overall frequency of WebSockets use by A&A domains did not change after the release of Chrome 58, although the number of unique initiators dropped from 72 to 19, as major ad networks (e.g., Google) discontinued their use.
- We find sensitive information being sent over WebSockets to A&A companies. 33across collects browser fingerprints [43], [35], [1], [25], [15], while Hotjar collects the entire DOM, which can contain sensitive information such as search queries, unsent messages, etc., within the given webpage. Lockerdome was using WebSockets to serve URLs to ads. These results highlight that the WRB did enable A&A companies to circumvent blocking extensions in ways that users may find objectionable.

II. Background

We begin by providing an overview of WebSockets, the `webRequest` API, and a brief timeline of the WRB.

A. WebSockets

The WebSocket protocol, standardized by RFC 6455 in 2011, gave JavaScript developers access to a bidirectional,

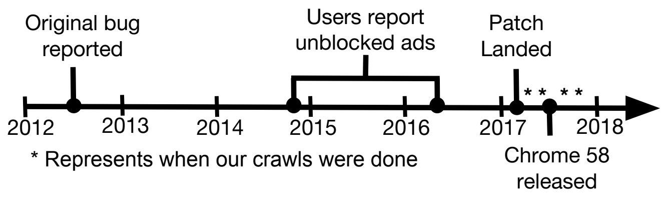


Fig. 1: Timeline for the WRB.

socket-like network interface, in which client-side JavaScript can open a WebSocket connection (with or without TLS encryption) to a server. This protocol enables developers to create web applications that receive real-time information or "pushed" messages from the server-side, without wasting bytes or incurring latency due to the constant construction of new TCP connections.

B. *webRequest API*

As of 2017, major browser vendors like Firefox and Edge support the Chrome extension API. One of its key capabilities is the `chrome.webRequest` API, which allows extensions to inspect, modify, and even drop outgoing network requests. The `chrome.webRequest.onBeforeRequest` callback is often used by ad blockers and privacy preserving tools to filter undesirable outgoing network requests [23].

C. *The Rise and Fall of a Bug*

In May 2012, users created a bug report in the Chromium issue tracker after observing that WebSocket connections did not trigger the `chrome.webRequest.onBeforeRequest` callback [23]. We refer to this as the webRequest Bug (WRB).

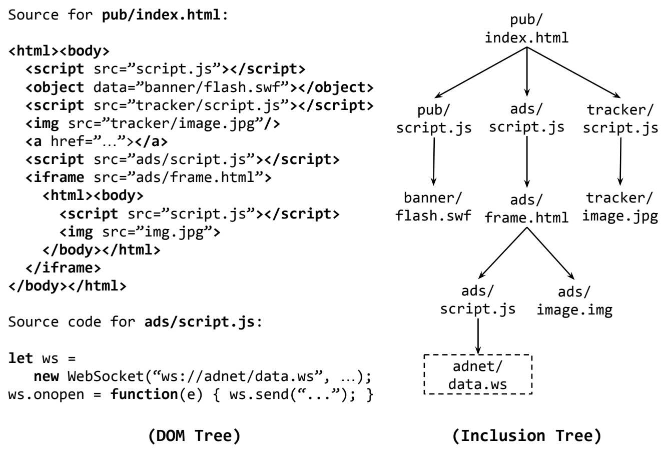
The WRB languished unpatched for four years. In late 2014, AdBlock Plus users began to report that unblockable ads were appearing on specific webpages, but only in Chrome [2]. By mid-2016, EasyList and uBlock Origin users were also observing unblockable ads [21], [42]. The users investigated and determined that the ads were being loaded via WebSockets, i.e., the WRB was being leveraged by some ad networks to circumvent blockers. Blocking extensions implemented complicated workarounds to mitigate the WRB in the absence of a permanent bugfix [2], [20]. The bug was finally patched in Chrome 58, released in April 19, 2017 [40]. Figure 1 shows the timeline of the key events related to the WRB.

III. Methodology

The goal of our study is to analyze the usage of WebSockets on the web, and to determine whether A&A companies are using them to bypass ad blockers. In this section, we outline our data collection methodology. We also describe the inclusion trees produced by our crawler, and explain how we use these to detect WebSockets.

A. *Inclusion Tree*

To determine which A&A companies are using WebSockets to circumvent ad blocking, we are not only interested in determining the existence of a WebSocket on a webpage, but



```
Source for pub/index.html:

<html><body>
  <script src="script.js"></script>
  <object data="banner/flash.swf"></object>
  <script src="tracker/script.js"></script>
  <img src="tracker/image.jpg"/>
  <a href="..."></a>
  <script src="ads/script.js"></script>
  <iframe src="ads/frame.html">
    <html><body>
      <script src="script.js"></script>
      <img src="img.jpg">
    </body></html>
  </iframe>
</body></html>

Source code for ads/script.js:

let ws =
   new WebSocket("ws://adnet/data.ws", ...);
ws.onopen = function(e) { ws.send("..."); }
```

Fig. 2: Sample DOM tree with corresponding *inclusion tree*. Note how a WebSocket request becomes a child to the requesting JavaScript resource.

also figuring out which parties established the socket in the first place. Prior studies have shown that relying on HTTP requests to figure out resource inclusions can be misleading due to dynamic code (e.g., JavaScript, Flash etc.) from third parties [7]. This occurs because the HTTP *Referer* header is set to the first-party domain, even if the resource making the request originated from a third-party. Furthermore, using DOM trees to capture resource inclusions also does not work because the DOM captures *syntactic* structures rather than *semantic* relationships between resource inclusions.

To solve this problem, we use *inclusion trees*, originally introduced by Arshad et al. [4]. Inclusion trees capture the semantic relationship between resource inclusions in websites. Figure 2 shows a sample DOM tree and its corresponding inclusion tree. We capture inclusion trees from Chrome by leveraging the Chrome Debugging Protocol [11]. Specifically, to capture the inclusion relationships within Chrome using the `Debugger` domain, we track JavaScript by collecting the `scriptParsed` events, which are triggered by the execution of inline and remote scripts. We observe further resource requests via the `requestWillBeSent` and `responseReceived` events in the `Network` domain. Using these two events, we can capture most of the dynamic inclusion chains. To capture the inclusion of `iframes`, we collect `frameNavigated` events in the `Page` domain.

B. *WebSocket Detection and Labeling*

A main distinguishing feature of our tool from previous work [4], [7], [29] is its ability to detect WebSocket requests initiated by JavaScript. In our implementation, we treat WebSockets as child nodes of the JavaScript node responsible for initiating them. Figure 2 shows how adnet/data.ws becomes the child of ads/script.js. To identify WebSocket requests, we capture a number of events in the `Network` domain: `webSocketCreated`, `webSocketWillSendHandshakeRequest`, and `webSocketHandshakeResponseReceived` for socket initiation; `webSocketFrameSent` and `webSocketFrameReceived` for data collection; and `webSocketClosed` for socket termination.

Detecting A&A Resources. To determine whether a socket was initiated by scripts or objects that originated from A&A domains, we first derive a set of A&A domains from the inclusion chains provided by Bashir et al. [7]. Each resource in [7] is tagged as A&A or non-A&A using the EasyList and EasyPrivacy rule lists; from this dataset, we extract a set of all 2^{nd}-level domains D. Let $a(d)$ and $n(d)$ be the number of times a given 2^{nd}-level domain $d \in D$ was labeled as A&A and non-A&A, respectively. We construct our final A&A set D' containing all $d \in D$ where $a(d) \geq 0.1 * n(d)$, i.e., we filter out 2^{nd}-level domains that are labeled as A&A less than 10% of the time to eliminate false positives.

To detect WebSockets that were initiated by A&A resources, we descend the branch of the inclusion tree that includes the socket. If the domains of any of the parent resources are present in D', we consider the socket to be included by an A&A resource. We refer to such sockets as *A&A sockets*.

C. Data Collection

As an initial seed of websites to crawl, we collected 1.8 million unique websites from 17 distinct categories provided by Alexa *Top Categories*. We sampled the top 5.8K websites from each category. Additionally, we sampled 5.8K websites from Alexa's top 1 million. This approach gives us a wide diversity of popular and unpopular websites across many different categories. After removing duplicates around 100K websites remained, which we use for our crawls.

We built a crawler on top of the Chrome Remote Debugging Protocol to drive the Chrome browser. The crawler works as follows: for every website w in our list, it visits the homepage. It then proceeds to extract all links L from the homepage that points to w. Our crawler randomly visits 15 links from L, waiting one minute between subsequent visits.

Overall, we performed four crawls over our sampled 100K websites. Two crawls were performed just prior to the release of Chrome 58 (which included the patch for the WRB bug) [40] between April 2–April 5 and April 11–April 16, 2017. To observe if the patch affected the usage of WebSockets by websites and A&A companies, we ran two more crawls after the release of Chrome 58. The first of these crawls was performed right after the patch between May 07–May 12, 2017. The second crawl was performed between October 12–October 16, 2017. Table I shows the high-level statistics for all four crawls in our study.

IV. Analysis

In this section, we analyze our dataset to understand the usage of WebSockets, the A&A companies involved, and the content being sent and received over the socket.

TABLE I: High-level statistics for our crawls.

Crawl Dates	% Sites w/ Sockets	% Sockets w/ A&A Initiators	# Unique A&A Initiators	% Sockets w/ A&A Receivers	# Unique A&A Receivers
Apr 02–05, 2017	2.1	60.2	72	72.0	14
Apr 11–16, 2017	2.4	61.0	61	73.0	16
May 07–12, 2017	1.6	60.2	17	68.3	13
Oct 12–16, 2017	2.5	54.9	19	55.1	14

A. Overall WebSocket Usage

We begin by providing an overview of WebSocket usage in Table I. We observe that only $\sim 2\%$ of the websites use WebSockets (column 2), with 6–12 WebSocket connections on average per website that uses the technology.

Among the WebSockets we observe, >90% contact a third-party domain (i.e., the WebSocket was cross-origin) and 55–73% contact an A&A domain (column 5). Across all four crawls, we observe 383 unique third-party domains and 17 A&A domains being contacted through WebSockets. Similarly, 55–61% of the WebSockets are initiated by a resource from an A&A domain (column 3). In total, we observe resources from 91 unique A&A domains initiating WebSockets.

Figure 3 shows the CDF of the number of WebSocket connections with respect to initiators and receivers. We see that A&A initiators and receivers are involved in an order of magnitude more WebSocket connections than non-A&A initiators and receivers. This reinforces our findings from Table I: of the domains that use WebSockets, the heaviest users are involved in A&A.

Figure 4 shows the CDF of number of unique parties contacted by initiators and contacting receivers. We see that A&A initiators contact only a few select partners, whereas A&A receivers are contacted by many parties. More than 47% of the A&A receivers were contacted by $\geq$10 parties. This accords with the results from Table I: there is dramatic fan-in from the 91 A&A initiators to the 17 A&A receivers.

Our overall observations about WebSocket usage are similar to those of Snyder et al. [44]. In [44], the authors crawled the Alexa Top-10K websites in 2016 and observed 544 (5.4%) sites using WebSockets, while 65% of those connections were blocked by AdBlock Plus[1]. Contrasting this to our results in Table I, we observed fewer websites using WebSockets, and a somewhat lower fraction of A&A sockets. These differences may be due to the larger sample size and broader coverage of less popular websites in our crawls.

Publishers. Now that we know that WebSockets are being used by some A&A domains, we ask: Is this practice widespread across publishers? To answer this question, we plot Figure 5, showing the fraction of A&A and non-A&A WebSockets observed over publishers sorted by Alexa rank. We see that the fraction of A&A sockets is twice that of non-A&A sockets across all ranks. We also see that both types of WebSockets are most prevalent on highly-ranked domains, with a drop occurring between 10K and 20K. The fraction of A&A sockets in top 10K publishers is 4.5 times higher than

[1]Snyder et al. crawled using Firefox, which was not impacted by the WRB.

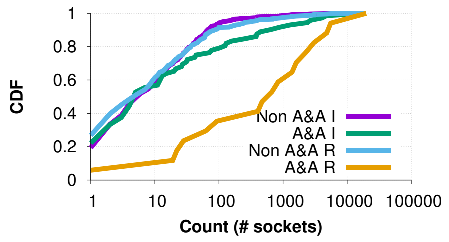


Fig. 3: **I**nitiators and **R**eceivers involved in WebSocket creations.

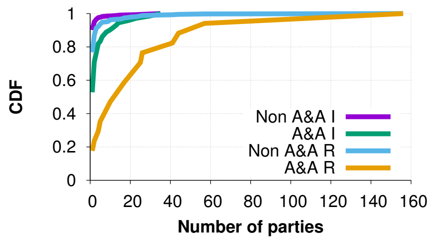


Fig. 4: Number of parties contacted by WebSocket **I**nitiators and **R**eceivers.

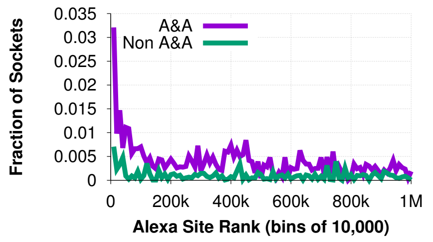


Fig. 5: Alexa rank of publishers that have WebSockets.

TABLE II: Top 15 WebSocket initiators sorted by the total number of unique receivers. A&A initiators are in bold.

	# Receivers		
Initiator	**Total**	**A&A**	**Socket Count**
facebook	35	11	441
espncdn	35	0	92
h-cdn	30	0	39
doubleclick	29	9	250
slither	25	0	33
google	23	11	381
youtube	18	8	129
cloudflare	15	1	873
addthis	14	8	101
hotjar	13	6	2407
cloudfront	13	4	4039
googlesyndication	10	6	71
adnxs	8	3	31
googleapis	7	0	157
sharethis	6	4	20

TABLE III: Top 15 A&A domains receiving WebSocket connections, sorted by the total number of unique initiators.

	# Initiators		
Receiver	**Total**	**A&A**	**Socket Count**
intercom	156	15	5534
33across	57	19	1381
zopim	44	12	19820
realtime	41	27	1612
smartsupp	26	4	670
feedjit	25	10	3017
inspectlet	25	6	836
hotjar	17	11	2255
disqus	17	13	4802
freshrelevance	10	2	404
lockerdome	10	8	471
pusher	5	3	22
velaro	4	3	62
inside-graph	2	2	28
simpleheatmaps	1	1	93

TABLE IV: Top 15 initiator/receiver pairs communicating via WebSockets. A&A domains are in bold.

Initiator	Receiver	Socket Count
cloudfront	visitors	3304
webspectator	**realtime**	1285
cloudfront	**pusher**	298
cloudfront	**freshrelevance**	281
google	**zopim**	172
blogger	**feedjit**	158
hotjar	**intercom**	144
cdn77	**smartsupp**	122
acenterforrecovery	**intercom**	114
facebook	**zopim**	112
vatit	**intercom**	110
plymouthart	**intercom**	108
welchllp	**intercom**	105
biozone	**intercom**	101
getambassador	**pusher**	101
A&A domain to itself		41,056

that of non-A&A sockets. This demonstrates that WebSocket usage, especially A&A sockets, is widespread amongst top publishers.

Before and After. Chrome rolled out the patch for WRB in version 58 on April 19, 2017. To understand if the release of the patch affected the usage of WebSockets, we can compare the statistics in Table I from our crawls before and after this date. Although we see that there has been a significant drop in the number of unique A&A domains initiating WebSockets over time (column 4), the fraction of A&A-initiated sockets has essentially remained the same (column 3). In total, 57 A&A initiators disappeared between our first and last crawl, including Google, Facebook, and AddThis. It is unclear why these major advertising companies abandoned WebSockets.

With respect to A&A socket receivers, Table I shows that there has been essentially no change over time (column 6). As we show in § IV-B, many of the A&A receivers provide services that are dependent on WebSockets (e.g., real-time commenting and chat), thus it is not surprising that these companies have not altered their software.

B. Initiators and Receivers

Next, we take a deeper look into the domains that initiate and receive WebSockets. Table II shows the top 15 domains whose resources initiate WebSockets; A&A domains are shown in bold. We observe that scripts from some of the largest players in the online advertising ecosystem (e.g., Google, Facebook, and AppNexus) create WebSockets to multiple other A&A domains. This demonstrates that major ad exchanges have embraced WebSockets, although as noted above, some have discontinued this practice.

Table III shows A&A domains that we observe receiving WebSocket connections. Only 2.5% of the initiators that create WebSockets to these domains are A&A domains, meaning many of the incoming connections are initiated by benign domains, or even first-party publishers. These WebSockets are particularly problematic with respect to the WRB: since the scripts that initiated the WebSockets were not blocked by AdBlock Plus, the only way to stop these connections would be to block the WebSockets themselves, which was not possible prior to Chrome 58.

In contrast to the initiators in Table II, the A&A receivers are less well-known companies that provide a variety of services. The most recognizable company, Disqus, provides user comment boards as a service to publishers; it is also an ad network that enables publishers to monetize their comment boards by displaying targeted ads. 33across and Lockerdome are advertising platforms. Inspectlet, Hotjar, and SimpleHeatmaps are *session replay* services that track user interactions within websites to generate detailed analytical heatmaps of mouse movements, click, and keystrokes [13]. Zopim, Velaro, Smartsupp, and Intercom provide customer service live-chat widgets.

The variety of business models offered by the receivers in Table II reveals an important point: WebSockets are being used to serve advertisements **and** to track users. The former was

TABLE V: Items being sent by and received by A&A domains via WebSockets and HTTP/S.

	WebSockets		HTTP/S	
Sent Item	**Count**	**%**	**Count**	**%**
User Agent	39,893	100.0	99,942,662	100.0
Cookie	27,968	70.11	22,752,063	22.77
IP	2,458	6.16	896,162	0.90
User ID	1,694	4.25	1,116,111	1.12
Screen	1,370	3.43	104,794	0.10
Device	1,370	3.43	177,101	0.18
Browser	1,368	3.43	89,614	0.09
Viewport	1,366	3.42	336,704	0.34
Scroll Position	1,366	3.42	291	0.00
Orientation	1,366	3.42	71	0.00
First Seen	1,366	3.42	8,148	0.01
Resolution	1,366	3.42	132,742	0.13
Language	722	1.81	914,628	0.92
DOM	581	1.46	8,587	0.01
Binary	397	1.00	6,267	0.01
No data	7,150	17.92	-	-
Received Item	**Count**	**%**	**Count**	**%**
HTML	18,976	47.57	11,599,601	11.61
JSON	4,976	12.47	1,633,849	1.63
JavaScript	356	0.89	27,027,458	27.04
Image	125	0.31	21,324,840	21.34
Binary	100	0.25	496,929	0.50
No data	8,575	21.49	-	-

noticed by users [21], [42] and led to the patching of the WRB, but to the best of our knowledge the latter has not been reported. We examine the contents of WebSocket messages in detail in § IV-C.

Table IV shows the top 15 initiator/receiver pairs that created A&A sockets (i.e., one or both of the parties must be an A&A domain), sorted by total WebSockets. Note that we aggregate cases where the initiator and receiver are the same and present the total in the last row of Table IV. Unsurprisingly, the vast majority of A&A sockets fall into this category (e.g., we observe 17,968 WebSockets initiated by Zopim to themselves). The cases where the initiator and receiver are different are more interesting, in the sense that these pairs of companies made explicit choices to interface via WebSockets. These cases are also more troubling from a privacy perspective, since the WRB may have prevented blockers from halting information flows to third-parties (assuming the initiator's script was not blocked in the first place).

C. Content Analysis

In this section, we investigate the content of messages being sent and received over the WebSockets. For sent messages, we would like to know if any Personally Identifiable Information (PII) or fingerprinting-related browser state are being sent, since A&A domains can use this information to track users [43], [35], [1], [25], [15]. For received messages, we are interested in whether ad images or JavaScript (that can be used to further exfiltrate data or retrieve ads) are being downloaded.

Table V shows different items that we observe being sent and received over the A&A sockets. For comparison, we also present statistics on how frequently we observed those same items being sent/received over HTTP/S to any A&A domain. Many of the items, such as user-agents, cookies, and IP addresses, are self-explanatory. "User ID" refers to unique identifiers related to the user such as *Account ID*, *Client ID*, and *User ID* itself. "Browser" contains the fingerprinting variables used to identify *Browser Type* and *Browser Family*, whereas "Device" refers to *Device Type* and *Device Family*. "First seen" appears to be the date on which the user's cookie was created. We extracted all of these variables from raw network traffic by manually inspecting the flows and building up a large library of regular expressions.

In all cases, we observe a greater percentage of private information being exfiltrated via WebSockets than over HTTP/S. This includes typical stateful-tracking data such as cookies, IP addresses, and unique identifiers. Perhaps more surprising are the ∼3.4% of WebSockets where browser fingerprinting data (e.g., screen size and orientation) was exfiltrated; we observed 59 initiator/receiver pairs involved in this practice, with 33across being the receiver in 97% of the pairs. Most surprising were the ∼1.5% of WebSockets where the entire DOM was serialized and uploaded, in this case to Hotjar, for the purposes of enabling session replays of user activity [13]. The DOM is potentially very privacy-sensitive, as it may reveal search queries, sensitive interests, unsent messages, etc., within the given webpage. Finally, we observed binary-encoded data being sent on 1% of WebSockets, but we were unable to decode it. The results in Table V highlight that the WRB allowed trackers to circumvent blockers and implement aggressive tracking techniques.

Next, we examine the information received over A&A sockets. Of the 78.5% WebSockets that did receive any data, WebSockets downloaded a greater percentage of HTML and JSON, as compared to JavaScript and images which were downloaded more often over HTTP/S.

We did not observe any ad images being sent directly over WebSockets (we checked for binary and base64 encoded media files). However, we did find that Lockerdome was sending URLs to ad images in their WebSocket responses, along with meta-data such as image captions, heights, and widths. These images were hosted on cdn1.lockerdome.com, which was not blacklisted in EasyList, meaning that the WRB was effectively allowing Lockerdome to circumvent ad blockers. Figure 6 shows three examples of these ads, which are emblematic of the low-quality "clickbait" that is served by unscrupulous ad networks and Content Recommendation Networks [8]. Furthermore, these are the same types of ads that were flagged by users in the WRB bug reports [21], [42]. This demonstrates that there are ad networks who were willing to exploit the WRB to serve ads, and that unsurprisingly, these shady ad networks cater to shady advertisers.

V. Related Work

The Online Ad Ecosystem. There are a plethora of empirical studies that have measured the online advertising ecosystem. Barford et al. [6] looked at the major ad networks, targeted ads, and associated user characteristics on the web by mapping the online *adscape*, whereas Rodriguez et al. measured the ad ecosystem on mobile devices [47]. Gill et al. [18] used browsing traces to study the economy of online advertising and discovered that most of the revenue is skewed

Fig. 6: Example of ads received over WebSockets. *Left*: "Odd Trick To Fix Sagging Skin Is Absolutely Genius". *Center*: "Study Reveals What Just A Single Diet Soda Does To You". *Right*: "Win an iPad Air 2 from Addicting Games!"

towards a few big companies. Acar et al. [1] conducted crawls over the Alexa Top-3K to find user identifiers being shared across domains. Similarly, Cahn et al. [9] observed that $<1\%$ of the trackers are present on 75% of Alexa Top-10K websites. Falahrastegar et al. [16] take a look at online trackers across geographic regions.

Other empirical studies have focused more on the individual implications of targeted advertising. Guha et al. [19] developed a controlled and systematic method for measuring online ads on the web based on trained *personas*. Carrascosa et al. [10] used these methods to prove that advertisers use sensitive attributes about users when targeting ads. Bashir et al. [7] used retargeted ads to determine information flows between ad exchanges. Olejnik et al. [38] noticed winning bid prices being leaked during Real Time Bidding (RTB) auctions and used this information to discover ad exchanges involved in cookie matching, as well as the relative value of different users.

Researchers have also studied malicious and bad practices in the advertising ecosystem. Zarras et al. [49] studied malicious ad campaigns and the ad networks associated with them, whereas Bashir et al. [8] found that some advertisers are not following industry guidelines and are serving poor quality ads.

Tracking Mechanisms. Krishnamurthy et al. were one of the first to bring attention to the pervasiveness of trackers and their privacy implications for users [26]. Since then, several studies have documented the evolution of online tracking on the web [27], [28], [9], [14].

Advertisers have upgraded their tracking techniques over time. Some of the techniques they employ include persistent cookies [24], local state in browser plugins [45], [5], browsing history through extensions [46], and fingerprinting methods [32], [37], [43], [35], [1], [25], [15], [14]. To expand users' interest profiles, advertisers share tracking information with each other through cookie matching [1], [38], [17], [7].

Anti-tracking. To avoid pervasive tracking, users are increasingly adopting tools that block trackers and ads [41], [30]. Papaodyssefs et al. [39] proposed the use of private cookies to mitigate tracking, while Nikiforakis et al. added entropy to the browser to combat fingerprinting [34]. Merzdovnik et al. and Iqbal et al. performed large scale measurements of blocking extensions and techniques to determine which are most effective [31], [22]. Snyder et al. [44] performed a browser feature usage survey and showed that ad and tracking blocking extensions do not block all standards equally, with WebSockets being blocked 65% of the times.

VI. Discussion

The Good. Overall, our measurements demonstrate that the WRB was not leveraged to circumvent blockers by the vast majority of A&A companies. Although we find that $\sim$67% of WebSockets on the open web are initiated or received by A&A domains (see § IV-A), most of these companies have a legitimate reason to be using WebSockets. For example, Disqus, Zopim, Velaro, and Intercom all offer real-time services that are ideal use-cases for WebSockets (see § IV-B).

The Strange. A troubling finding of our study is that major ad and tracking platforms, like Google, Facebook, AddThis, and AppNexus, adopted WebSockets (see Tables II and IV). This is extremely concerning, since these companies dominate the online display ad ecosystem and are essentially omnipresent on the web. Yet strangely, we do not observe these major ad platforms initiating WebSocket connections after the release of Chrome 58 (when the WRB was patched, see § IV-A). The observational nature of our study prevents us of from drawing causal conclusions about this finding, and indeed, it may be coincidental.

The Bad. Previous studies of online tracking have repeatedly identified "innovators" attempting to use bleeding-edge techniques to gain an advantage against privacy-conscious users. Examples include the use of persistent cookies and various kinds of fingerprinting [24], [45], [5], [46], [32], [37], [43], [35], [1], [25], [15], [14].

We identify three companies that appear to have been using the WRB to circumvent blocking extensions: 33across was harvesting large amounts of browser state that could be used for fingerprinting; Lockerdome was downloading URLs to ads (see § IV-C and Figure 6); and Hotjar was downloading the entire DOM from webpages. These results highlight an important facet of the WRB debacle: although users clamored for a patch after observing ads slipping through blockers [21], [42], our results demonstrate that invisible tracking was an equally important and disturbing implication of the WRB.

Acknowledgments

We thank the anonymous reviewers for their helpful comments. This research was supported in part by NSF grants IIS-1408345 and IIS-1553088. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proc. of CCS*, 2014.
[2] WebSocket connections can't be blocked. AdBlock Plus Issue Tracker, Dec. 2014. https://issues.adblockplus.org/ticket/1727.
[3] Put the YourAdChoices Icon to Work for You. Digital Advertising Alliance, 2017. http://youradchoices.com/learn.
[4] S. Arshad, A. Kharraz, and W. Robertson. Include me out: In-browser detection of malicious third-party content inclusions. In *Proc. of Intl. Conf. on Financial Cryptography*, 2016.
[5] M. Ayenson, D. J. Wambach, A. Soltani, N. Good, and C. J. Hoofnagle. Flash cookies and privacy ii: Now with html5 and etag respawning. *Available at SSRN 1898390*, 2011.

[6] P. Barford, I. Canadi, D. Krushevskaja, Q. Ma, and S. Muthukrishnan. Adscape: Harvesting and analyzing online display ads. In *Proc. of WWW*, 2014.

[7] M. A. Bashir, S. Arshad, , W. Robertson, and C. Wilson. Tracing information flows between ad exchanges using retargeted ads. In *Proc. of USENIX Security Symposium*, 2016.

[8] M. A. Bashir, S. Arshad, and C. Wilson. "Recommended For You": A First Look at Content Recommendation Networks. In *Proc. of IMC*, 2016.

[9] A. Cahn, S. Alfeld, P. Barford, and S. Muthukrishnan. An empirical study of web cookies. In *Proc. of WWW*, 2016.

[10] J. M. Carrascosa, J. Mikians, R. Cuevas, V. Erramilli, and N. Laoutaris. I always feel like somebody's watching me: Measuring online behavioural advertising. In *Proc. of ACM CoNEXT*, 2015.

[11] Chrome devtools protocol viewer. GitHub. https://developer.chrome.com/devtools/docs/debugger-protocol.

[12] S. Cowley and J. Pepitone. Google to pay record $22.5 million fine for Safari privacy evasion. CNNMoney, Aug. 2012. http://money.cnn.com/2012/08/09/technology/google-safari-settle/index.html.

[13] S. Englehardt. No boundaries: Exfiltration of personal data by session-replay scripts. Nov. 2017. https://freedom-to-tinker.com/2017/11/15/no-boundaries-exfiltration-of-personal-data-by-session-replay-scripts.

[14] S. Englehardt and A. Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Proc. of CCS*, 2016.

[15] S. Englehardt, D. Reisman, C. Eubank, P. Zimmerman, J. Mayer, A. Narayanan, and E. W. Felten. Cookies that give you away: The surveillance implications of web tracking. In *Proc. of WWW*, 2015.

[16] M. Falahrastegar, H. Haddadi, S. Uhlig, and R. Mortier. The rise of panopticons: Examining region-specific third-party web tracking. In *Proc of. Traffic Monitoring and Analysis*, 2014.

[17] M. Falahrastegar, H. Haddadi, S. Uhlig, and R. Mortier. Tracking personal identifiers across the web. In *Proc. of PAM*, 2016.

[18] P. Gill, V. Erramilli, A. Chaintreau, B. Krishnamurthy, K. Papagiannaki, and P. Rodriguez. Follow the money: Understanding economics of online aggregation and advertising. In *Proc. of IMC*, 2013.

[19] S. Guha, B. Cheng, and P. Francis. Challenges in measuring online advertising systems. In *Proc. of IMC*, 2010.

[20] R. Hill. A companion extension to uBlock Origin. GitHub. https://github.com/gorhill/uBO-Extra.

[21] R. Hill. ws-gateway websocket circumvention ? #1936. GitHub, Aug. 2016. https://github.com/gorhill/uBlock/issues/1936.

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

[23] chrome.webRequest.onBeforeRequest doesn't intercept WebSocket requests. Chromium Bugs, May 2012. https://bugs.chromium.org/p/chromium/issues/detail?id=129353.

[24] S. Kamkar. Evercookie - virtually irrevocable persistent cookies., September 2010. http://samy.pl/evercookie/.

[25] T. Kohno, A. Broido, and K. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, 2005.

[26] B. Krishnamurthy, D. Malandrino, and C. E. Wills. Measuring privacy loss and the impact of privacy protection in web browsing. In *Proc. of SOUPS*, 2007.

[27] B. Krishnamurthy, K. Naryshkin, and C. Wills. Privacy diffusion on the web: A longitudinal perspective. In *Proc. of WWW*, 2009.

[28] B. Krishnamurthy and C. Wills. Privacy leakage vs. protection measures: the growing disconnect. In *Proc. of W2SP*, 2011.

[29] T. Lauinger, A. Chaabane, S. Arshad, W. Robertson, C. Wilson, and E. Kirda. Thou Shalt Not Depend on Me: Analysing the Use of Outdated JavaScript Libraries on the Web. In *Proc of NDSS*, 2017.

[30] M. Malloy, M. McNamara, A. Cahn, and P. Barford. Ad blockers: Global prevalence and impact. In *Proc. of IMC*, 2016.

[31] G. Merzdovnik, M. Huber, D. Buhov, N. Nikiforakis, S. Neuner, M. Schmiedecker, and E. R. Weippl. Block me if you can: A large-scale study of tracker-blocking tools. In *Proc. of Euro S&P*, 2017.

[32] K. Mowery and H. Shacham. Pixel perfect: Fingerprinting canvas in html5. In *Proc. of W2SP*, 2012.

[33] M. H. Mughees, Z. Qian, and Z. Shafiq. Detecting anti ad-blockers in the wild. *PoPETs*, 2017(3):130, 2017.

[34] N. Nikiforakis, W. Joosen, and B. Livshits. Privaricator: Deceiving fingerprinters with little white lies. In *Proc. of WWW*, 2015.

[35] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Proc. of IEEE Symposium on Security and Privacy*, 2013.

[36] R. Nithyanand, S. Khattak, M. Javed, N. Vallina-Rodriguez, M. Falahrastegar, J. E. Powles, E. D. Cristofaro, H. Haddadi, and S. J. Murdoch. Adblocking and counter blocking: A slice of the arms race. In *Proc. of FOCI*, 2016.

[37] L. Olejnik, C. Castelluccia, and A. Janc. Why Johnny Can't Browse in Peace: On the Uniqueness of Web Browsing History Patterns. In *Proc. of HotPETs*, 2012.

[38] L. Olejnik, T. Minh-Dung, and C. Castelluccia. Selling off privacy at auction. In *Proc of NDSS*, 2014.

[39] F. Papaodyssefs, C. Iordanou, J. Blackburn, N. Laoutaris, and K. Papagiannaki. Web identity translator: Behavioral advertising and identity privacy with wit. In *Proc. of HotNets*, 2015.

[40] pkalinnikov. Issue 2449913002: Support websocket in webrequest api. (closed). Chromium Code Reviews. https://codereview.chromium.org/2449913002/.

[41] E. Pujol, O. Hohlfeld, and A. Feldmann. Annoyed users: Ads and ad-block usage in the wild. In *Proc. of IMC*, 2015.

[42] Technobuffalo.com. EasyList Forum, July 2016. https://forums.lanik.us/viewtopic.php?p=110902.

[43] F. Roesner, T. Kohno, and D. Wetherall. Detecting and defending against third-party tracking on the web. In *Proc. of NSDI*, 2012.

[44] P. Snyder, L. Ansari, C. Taylor, and C. Kanich. Browser feature usage on the modern web. In *Proc. of IMC*, 2016.

[45] A. Soltani, S. Canty, Q. Mayo, L. Thomas, and C. J. Hoofnagle. Flash cookies and privacy. In *AAAI Spring Symposium: Intelligent Information Privacy Management*, 2010.

[46] O. Starov and N. Nikiforakis. Extended tracking powers: Measuring the privacy diffusion enabled by browser extensions. In *Proc. of WWW*, 2017.

[47] N. Vallina-Rodriguez, J. Shah, A. Finamore, Y. Grunenberger, K. Papagiannaki, H. Haddadi, and J. Crowcroft. Breaking for commercials: Characterizing mobile advertising. In *Proc. of IMC*, 2012.

[48] S. J. Vaughan-Nichols. Chrome is the most popular web browser of all. ZDNet, Jan. 2017. http://www.zdnet.com/article/chrome-is-the-most-popular-web-browser-of-all/.

[49] A. Zarras, A. Kapravelos, G. Stringhini, T. Holz, C. Kruegel, and G. Vigna. The dark alleys of madison avenue: Understanding malicious advertisements. In *Proc. of IMC*, 2014.