



Arvind Narayanan
Department of Computer Science
Princeton University
arvindn@cs.princeton.edu
609-258-9302

Tom Wheeler
Chairman
Federal Communications Commission
445 12th Street, SW Washington, DC 20554

RE: Docket No. 16-106, Protecting the Privacy of Customers of Broadband and Other Telecommunications Services

May 27, 2016

Dear Chairman Wheeler:

I am an Assistant Professor of Computer Science at Princeton University. I study information security and privacy. I helped develop the Do Not Track standard and have co-authored numerous peer-reviewed publications on web privacy. My work has received several awards, including (twice) the PET Award, an international award for Outstanding Research in Privacy Enhancing Technologies, co-sponsored by Microsoft and the Privacy Commissioner of Ontario.

I wish to make three key points regarding the ability of Broadband Internet Access Service providers to infer the details of customers' web browsing activities. In support of these points, I have attached a publication (Englehardt et al., 2015) and a draft paper (Englehardt & Narayanan, 2016) that resulted from research projects that I led.

Sincerely,

A handwritten signature in black ink that reads "Arvind N." with a stylized flourish at the end.

Arvind Narayanan

1. The percentage of websites adopting HTTPS remains low.

In Section 5.3 of our study of online tracking (Englehardt & Narayanan 2016), we find that only 14.2% of the top 55,000 websites default to HTTPS on their home pages as of January 2016. This number falls to 8.6% on the top 1 million websites. Only a further 2.9% of the top 55,000 sites even offer HTTPS as an option.

2. There are serious impediments to the continued adoption of HTTPS.

Today's web is a mash-up, with websites incorporating a plethora of "third parties" to provide services such as advertising, analytics, and social-media functionality. In the same Section of the attached paper (Englehardt & Narayanan 2016), we find that many of these third parties do not support encryption, and that this impedes the adoption of HTTPS by websites. Specifically, a significant fraction of today's unencrypted sites will be unable to upgrade to HTTPS without jettisoning one or more of the third parties whose services they currently rely on.

3. Internet service providers benefit from the third-party web tracking ecosystem.

In the attached 2015 paper we studied the ability of a passive eavesdropper to track users by utilizing third-party cookies (Englehardt et al. 2015). Essentially all of our findings are applicable to Internet Service Providers (ISPs).

Multiple people may use the same Internet access subscription, and online tracking companies may sometimes be in a better position than ISPs to tell such users apart (say, if they use different browser profiles). However, any cookies created by those online trackers are then visible to the ISP on unencrypted traffic between the user and the website, increasing the ISP's visibility into its own users' activities. We quantified this ability in our 2015 paper and showed that it is surprisingly revealing.

Similarly, if a portable device connects to the Internet via different subscriptions or access points served by the same ISP, that ISPs can connect the dots using cookies found in that device's web traffic.

In other words, while much of today's privacy debate has focused on the online tracking economy due to its sophistication, this is not a reason to have lax privacy rules for ISPs. On the contrary, ISPs can in fact benefit from this tracking infrastructure.

To clarify, I claim no knowledge of whether today's ISPs use web cookies for tracking or profiling their customers. Rather, my point is that such tracking is technically feasible and effective.

References

Englehardt, Steven, et al. "Cookies that give you away: The surveillance implications of web tracking." Proceedings of the 24th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee, 2015.

Englehardt, Steven, and Narayanan, Arvind. "Online tracking: A 1-million-site measurement and analysis." Manuscript, 2016.

Cookies That Give You Away: The Surveillance Implications of Web Tracking

Steven Englehardt
Princeton University
ste@princeton.edu

Dillon Reisman
Princeton University
dreisman@princeton.edu

Christian Eubank
Princeton University
cge@princeton.edu

Peter Zimmerman
Princeton University
peterz@princeton.edu

Jonathan Mayer
Stanford University
jmayer@stanford.edu

Arvind Narayanan
Princeton University
arvindn@princeton.edu

Edward W. Felten
Princeton University
felten@princeton.edu

ABSTRACT

We study the ability of a passive eavesdropper to leverage “third-party” HTTP tracking cookies for mass surveillance. If two web pages embed the same tracker which tags the browser with a unique cookie, then the adversary can link visits to those pages from the same user (i.e., browser instance) even if the user’s IP address varies. Further, many popular websites leak a logged-in user’s identity to an eavesdropper in unencrypted traffic.

To evaluate the effectiveness of our attack, we introduce a methodology that combines web measurement and network measurement. Using OpenWPM, our web privacy measurement platform, we simulate users browsing the web and find that the adversary can reconstruct 62–73% of a typical user’s browsing history. We then analyze the effect of the physical location of the wiretap as well as legal restrictions such as the NSA’s “one-end foreign” rule. Using measurement units in various locations—Asia, Europe, and the United States—we show that foreign users are highly vulnerable to the NSA’s dragnet surveillance due to the concentration of third-party trackers in the U.S. Finally, we find that some browser-based privacy tools mitigate the attack while others are largely ineffective.

1. INTRODUCTION

How much can an adversary learn about an average user by surveilling web traffic? This question is surprisingly tricky to answer accurately, as it depends on four things: the structure of the web, the mapping of web resources to the topology of the global Internet, the web browsing behavior of a typical user, and the technical capabilities and policy restrictions of the adversary. We introduce a methodology for quantifying the efficacy of passive surveillance. Our technique combines web measurement, network measurement, a client model (that incorporates user browsing behavior, web browser policies and settings, and privacy-protecting extensions), and an adversary model.

More specifically, the adversary has the ability to inspect packet contents and wishes to either track an individual target user or surveil users *en masse*. A key challenge for the

adversary is the lack of persistent identifiers visible on the network (in Section 3 we discuss why IP addresses are inadequate). However, the adversary can observe HTTP cookies in transit. Indeed, both the NSA and GCHQ are known to use such cookies for surveillance (Section 3).

Our work starts with three insights. First, the presence of *multiple unrelated* third-party cookies on most web pages, albeit pseudonymous, can tie together most of a user’s web traffic without having to rely on IP addresses (Figure 1). Thus the adversary can separate network traffic into clusters, with each cluster corresponding to only one user (or more precisely, one browser instance). A single user’s traffic may span more than one cluster if the linking is imperfect.

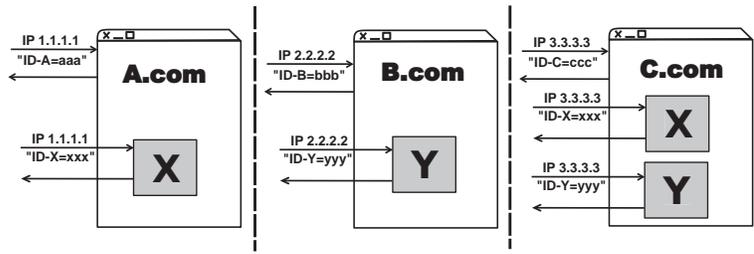
Second, a significant portion of a typical user’s traffic traverses U.S. borders even when the user is outside the U.S. and browses local content. As it turns out, such sites frequently include third-party resources such as analytics scripts, tracking pixels, and advertisements from U.S. servers. This leaves foreign users particularly vulnerable to the NSA’s wiretaps within the U.S. under the “one-end foreign” rule (Section 3).

Third, although most popular websites now deploy HTTPS for authentication, many web pages reveal an already logged-in user’s *identity* in plaintext. Thus, an adversary that can wiretap the network can not only cluster together the web pages visited by a user, but can then attach real-world identities to those clusters. This technique relies on nothing other than the network traffic itself for identifying targets.

Figure 1 illustrates the basis for our work. The adversary observes the user visit three different web pages which embed trackers *X*, *Y* or both. The user’s IP address may change between visits to each page, though we assume it is consistent for the request to site *A* and the request to *A*’s embedded tracker *X*. But there is no way to tie together her visits to pages *A* and *B* until she visits *C* after which all three visits can be connected. The unique cookie from *X* connects *A* and *C* while the one from *Y* connects *B* and *C*. We assume here that the user has visited pages with both trackers before so that cookies have already been set in her browser and will be sent with each request. While IP address is a *convenient* method to link a request to a first party page to the corresponding request to an embedded third party tracker, it is not necessary. In Section 6.1 we show how this linkage can be achieved even if the IP address cannot be observed at all or if an IP address is shared by many users.

Contributions. Our contributions are both conceptual and empirical. First, we identify and formalize a new pri-

Figure 1: Illustration of link between each of a single browser’s visits to three first-party pages using two different third-party tracking cookies. The user accesses the web at three different times, behind three different IP addresses.



vacy threat from packet sniffing. While the technique of utilizing cookies to target users is well known, we formulate the attack concretely in terms of the following steps: (1) automatically classifying cookies as unique identifiers (2) using multiple ID cookies to make *transitive* inferences and *clustering* HTTP traffic, (3) geographically tagging the flow of traffic, and (4) inferring real-world identity from HTTP request and response bodies. We believe this attack to be the strongest known way for a passive adversary to utilize web traffic for surveillance.

Second, we rigorously evaluate the above attack model using a novel methodology that combines web measurement, network measurement, a user model, and an adversary model (Section 4). We simulate realistic browsing behavior and measure the actual cookie ecosystem. This requires nuanced techniques along several fronts: (1) a crawling infrastructure based on *browser automation* to more closely simulate real users (2) a model of browsing history derived from real user behavior, and (3) network measurements to help verify the robustness of geolocation data.

Third, we present an exhaustive empirical evaluation of browser privacy settings and privacy extensions to discover how well a proactive user can protect himself against the attack model we’ve defined. Specifically, we measure how each of the following privacy measures affect the effectiveness of the attack: blocking all third-party cookies, blocking only those from sites not visited directly, setting Do Not Track (DNT), using Ghostery, and using HTTPS Everywhere.¹

Results. We simulate users that make 0 to 300 web page visits spread out over a 2-3 month period. We consider users located in several possible countries. For each such set of visits, we perform clustering using the method described above and find the “giant connected component.” For non-U.S. users, we consider an adversary with wiretaps in the target user’s country as well as one with wiretaps in the U.S.

At a high level, our results show that for a U.S. user, over 73% of visits fall into this connected component. The clustering effect is extremely strong and is robust to differences in the models of browsing behavior. Clustering even occurs when the adversary is able to observe only a small, random subset of the user’s requests.

Non-U.S. locations show a lower degree of clustering due to a lower number of embedded third-parties: over 59% of traffic falls into the giant connected component. If the adversary is further restricted to be in the U.S., the clustering level does drop significantly (12% – 20%), but this is still surprisingly high given that these users are browsing local content.

Second, we measure the presence of identifying information in plaintext among popular (Alexa Top 50 U.S.) websites. 56% of sites transmit some form of identifying information in plaintext once a user logs in, whether first name, first and last name, username, or email address. The majority of these (42% of websites overall) present *unique* identifiers (username or email address) in the clear.

Third, we show that many built-in browser protections are able to reduce but not fully mitigate the attack. The most effective blocking solution, Ghostery, still allows 24.2% of a user’s traffic to be clustered, while alternative solutions have far less of an impact.

Implications An adversary interested in targeted surveillance can proceed as follows: (1) Either scan for the target’s identity in plaintext HTTP traffic, or use auxiliary methods to obtain the target’s cookie ID on some first-party page (2) From this starting point, transitively connect the target’s known first-party cookie to other third-party and first-party cookies of the target. On the other hand, an adversary interested in *en masse* surveillance can first cluster all observed HTTP traffic, albeit at a high computational cost, and then attach identities to these clusters using the methods above. Our attacks show the feasibility of either adversary’s goal.

What can the adversary do after attaching an identity to a cluster of web traffic? First, browsing history itself could be the information of interest, as in the NSA plan to discredit ‘radicals’ based on browsing behavior, such as pornography [4]. Second, further sensitive information might be found in unencrypted web content such as preferences, purchase history, address, etc. Finally, it can enable active attacks such as delivering malware to a targeted user [47, 19].

2. RELATED WORK

Our work draws from two previously independent bodies of research. The first analyzes the privacy implications of third-party cookies and the second analyzes the ability of a network eavesdropper to infer sensitive information. We describe each in turn. To our knowledge, the two sets of ideas have not been combined before.

Third-party tracking: prevalence and privacy implications. There have been several notable results uncovering or quantifying various types of online tracking: cookies [28, 45], flash cookies (LSOs) including respawning behavior [46, 9, 38, 6], and browser fingerprinting [41, 7, 6]. The ability of trackers to compile information about users is further aided by PII leaks from first parties to third parties [29, 30, 27] and “cookie syncing”, or different third-party trackers linking their pseudonymous cookies to each other [42, 6].

While this body of research helps us understand what trackers themselves can learn, it does not address the question we are interested in, which is what an *eavesdropper* can

¹DNT: <http://donottrack.us/>, Ghostery: <https://www.ghostery.com>, HTTPS-E: <https://www.eff.org/https-everywhere>

learn through cookies and identifier leaks. The latter is influenced by many additional factors including geographic location of the trackers and the adoption of HTTPS by websites.

Yen et al. show how IP, cookies and usernames can be combined to track devices reliably even when any one of these identifiers may be individually unreliable [50]. The similarities to our work are superficial: we study linkage by an eavesdropper who utilizes third-party cookies rather than a website that uses its first-party cookies. The goals are also different: learning users’ web histories vs. ID-ing devices.

There are various client-side tools to block, limit or visualize third-party tracking. These are too numerous to list exhaustively, but a sampling include Adblock Plus, Ghostery, ShareMeNot [1], Lightbeam, and TrackingObserver [2]. Studies that have quantified the privacy effect of these tools include [35, 11, 20].

Surveillance: attacks, defenses, and measurement. While there is a large body of work on what a passive adversary can infer about users, virtually all of it concerns attacks arising from side-channels in encrypted traffic, particularly Tor. While Tor is insecure against a global passive adversary, traffic analysis attacks have been studied with respect to passive and active adversaries with less comprehensive access to the network [39, 40]. *Website fingerprinting* allows a local eavesdropper to determine which of a set of web pages the user is visiting, even if the connection is encrypted, by observing packet lengths and other features [25, 24, 43]. Other side-channel attacks include timing attacks on SSH [48], leaks in web forms, [15] and inferring spoken phrases from VoIP [49].

By contrast, we study users who do *not* use a properly configured Tor browser. The adversary’s main challenge is not linking origin and destination, but rather linking different traffic flows to the same (real-world) identity.

Closer to our work, Arnbak and Goldberg studied how the NSA could actively redirect U.S. traffic abroad, so as to bring it within broader surveillance authorities [8].² The IXMaps tool allows users to interactively view the routes taken by their traffic and intersection with known NSA wiretapping sites [16].

3. BACKGROUND AND THREAT MODEL

In recent years, a combination of technical research, leaks, and declassifications has provided unprecedented transparency into Internet surveillance by governments. Some nations, such as Iran and Bahrain [17], practice near-total Internet monitoring. Others, including the United States and Britain, have large-scale technical capacity—but subject to legal limits. This section explains how the National Security Agency (NSA) and Government Communication Headquarters (GCHQ) have used third-party cookies in their respective surveillance programs, as well as briefly discusses the laws that govern surveillance of Internet traffic within the United States. It then sets out a threat model that motivates our study.

NSA and GCHQ use of third-party cookies. Leaked documents reflect at least three ways in which the NSA has used third-party cookies obtained from its Internet intercepts. First, the agency has investigated passively identify-

²It is not apparent whether the NSA has redirected traffic in this manner, nor is it apparent whether the agency would consider the practice lawful.

ing Tor users by associating cookies with non-Tor sessions. Specifically, the NSA attempted to link a Google third-party advertising cookie between Tor and non-Tor sessions [5].

Second, the agency has an active, man-in-the-middle system (“QUANTUMCOOKIE”) that induces cookie disclosure [5]. Applications include identifying Tor users and targeting malware delivery.

Third, the agency has used passively obtained cookies to target active man-in-the-middle exploitation. On at least one occasion, the NSA offered a Google cookie to single out a user for exploitation [47].

In addition to these specific applications, HTTP analytical tools (such as “XKEYSCORE”) incorporate cookie data. An analyst could easily take advantage of third-party cookies when querying intercepted data [21].

Several leaked documents also reveal two GCHQ programs for surveilling and targeting users via third-party tracking data, both from web browsers and mobile applications. One program, “MUTANT BROTH”, a repository of tracking cookies linked with additional metadata such as IP addresses and User-Agent strings. This repository is reported to have been used for targeted malware delivery [19].

The other program, “BADASS”, offers a similar repository and search interface for querying information leakage from mobile apps. The system collects and extracts leaked identifiers, device and operating system details, and additional information transmitted in plaintext [31].

United States Internet monitoring. The law surrounding NSA authority derives from a complex mixture of constitutional doctrine, statutory restrictions, and executive regulation. One emergent property is that, when at least one party to a communication is outside the United States, it is eligible for warrantless surveillance.³ “Upstream” interception devices, controlled by the NSA and foreign partners, are exposed to large volumes of this “one-end foreign” Internet traffic. While the details remain classified, it also appears that a substantial quantity of one-end foreign traffic is temporarily retained. Leaks indicate that at least some installations of “XKEYSCORE,” a distributed analysis system, maintain a multi-day buffer of Internet traffic [12].

Threat model. In developing a threat model, there are two extremes, neither of which is desirable. The first is to assume that the adversary is all-powerful, as in cryptographic security arguments. Such a model is both uninteresting and largely irrelevant to the real world. The other extreme is to focus too closely on the NSA or GCHQ’s activities. Such a model may not yield insights that apply to other surveillance programs and the results may be invalidated by changes to the respective agency’s programs. We seek a careful middle ground and arrive at a model that we believe is realistic enough to influence policy debates and privacy tool development, yet general enough for our analyses and algorithms to be of independent scientific interest and for our results to hold up well over time.

We consider only passive attacks for several reasons. First, passive attacks appear to be more powerful than generally

³One-end foreign wireline interceptions inside the United States are generally governed by Section 702 of the FISA Amendments Act [22]. Two-end foreign interceptions and one-end foreign wireless interceptions inside the United States are generally governed by Executive Order 12333. Interceptions outside the United States are also generally governed by Executive Order 12333 [3].

realized, and we wish to highlight this fact. Second, even an active attack must usually begin with passive eavesdropping. An adversary must have refined criteria for targeting the active attack. Finally, almost all active attacks carry some risk of detection. Passive attacks much easier to mount, especially at large scale.

We consider a powerful adversary with the ability to observe a substantial portion of web traffic on the Internet backbone. The adversary’s view of a given user’s traffic may be complete or partial. We model partial coverage in two ways: by assuming that a random subset of the user’s HTTP requests and responses flows through one of the adversary’s wiretaps, or that the adversary taps the portion of the user’s traffic that traverses United States borders. While the NSA has many interception points outside U.S. borders as well, the U.S.-only model provides a useful, approximate lower bound of the agency’s abilities. We also assume that the adversary cannot routinely compromise HTTPS, so cookies or other identifying information sent over HTTPS are of no use.

The adversary may have one of two goals: first, he might want to target a specific individual for surveillance. In this case the adversary knows either the target’s real-world identity or a single ID cookie known to belong to the target (whether on a first or third party domain). Second, the adversary might be engaged in mass surveillance. This adversary would like to “scoop up” web traffic and automatically associate real-world identities with as much of it as possible.

The adversary’s task is complicated by the fact that the IP addresses of the target(s) may change frequently. A user’s IP address could change because she is physically mobile, her ISP assigns IP addresses dynamically, or she is using Tor. Leaked GCHQ documents show that their search interface even warns analysts to take care when selecting data on dynamic IPs [19]. Browsing from a smartphone is a case worth highlighting: Balakrishnan et al. find that “individual cell phones can expose different IP addresses to servers within time spans of a few minutes” and that “cell phone IP addresses do not embed geographical information at reasonable fidelity” [10].

To link users across different networks and over time, the adversary aims to utilize first-party and third-party unique cookies assigned to browser instances by websites. He can easily sniff these on the network by observing the “Cookie” field in HTTP request headers and the “Set-Cookie” field in HTTP response headers. Cookies set by an “origin” (roughly, a domain) that have not expired are automatically sent as part of requests to the same origin.

4. METHODOLOGY

In this study, we wish to simulate real users browsing over a period of time, detect the creation of unique identifiers, and measure the flow of both unique pseudonymous and real-world identifiers to adversaries with differing collection capabilities. We present a summary of our methodology below, and provide detailed descriptions of our measurement and analysis methodology in the following subsections.

1. Define all clients and adversaries to be studied, according to the following models:
 - client: (location, browsing model, browser configuration) which encodes the client’s geographic and network location, which sites the client visits, and the browser settings and plugins the client browses with.

- adversary: (location, policy restrictions) which encodes the adversary’s geographic and network location, and the policy restrictions on data use and collection.
2. For each unique (user location, browsing model) pair of interest, generate N simulated browsing profiles as defined in Section 4.1 and create a corresponding client instance for each one.
 3. Use the measurement infrastructure (Section 4.2) to simulate the users defined in Step 2 and collect all network traffic (i.e. HTTP requests, responses, and cookies). Our measurements are summarized in Section 4.3.
 4. For each (client location, web resource) pair of interest, determine the geographic path of traffic using the procedure described in Section 4.4.
 5. Run the ID cookie detection algorithm detailed in Section 4.5 to flag identifying cookies
 6. For each (client, adversary) pair of interest, do the following for all instances of the client and average the results:
 - filter the list of requests based on the geographic location and policy restrictions of the adversary using the geographic mapping created in Step 4.
 - run the cookie linking algorithm detailed in Section 4.6 using the ID cookies detected in Step 5.
 - report the size of the connected components in the linking graph (as a ratio of total number of visits)
 - report the number of sites known to leak real-world identifiers (Section 4.7) contained in each component.

4.1 Browsing models

We use two browsing models to create simulated user profiles. One of our models was a naive one – the user visits random subsets of the Alexa top 500 sites local to the location of the measurement instance. For example, a measurement instance in Japan would sample the Alexa top-500 sites for users in Japan, while a measurement instance in Ireland would sample from the Alexa top-500 sites for users in Ireland.

Our other browsing model aims for realism by making use of the AOL search query log dataset. The dataset contains queries made by 650,000 anonymous users over a three month period (March–May 2006). We create a browsing profile from a user’s search queries as follows. First, we remove repeated queries. Next, for every search query performed by the user, we submit the query to Google search and retrieve the links for the first five results. Users were selected on the basis that they performed between 50 to 100 unique queries which resulted in browsing profiles of 250 to 500 URLs. This is almost identical to the method that was used in [32].

Of course, only a subset of real users’ web browsing results from web searches. Nevertheless, we hypothesize that our profiles model two important aspects of real browsing histories: the distribution of popularity of web pages visited, and the topical interest distribution of real users. Popular websites may embed more trackers on average than less popular sites, and websites on the same topic may be more interconnected in terms of common embedded trackers. Failure to model these aspects correctly could skew our results.

The reason we recreate the users’ searches on a current search engine rather than simply using the sites visited by the AOL users (available in the dataset) is that the distribution of websites visited by real users changes over time as websites rise/fade in popularity, whereas the distribution of users’ interests can be expected to be more stable over time.

4.2 Measurement infrastructure

We built our study on top of a web measurement platform, OpenWPM [18], which we developed in earlier work. The platform has the ability to drive full browsers, such as Firefox, with any set of browser extensions and collects a wide range of measurement data as it visits sites.⁴ In our study, we use version 0.1.0 of OpenWPM to drive Firefox measurement instances from which we record all HTTP data for analysis. We configure the measurement instances to browse with profiles generated from the models described in Section 4.1, and deploy the crawls on cloud machines in the United States, Japan, and Ireland.

4.3 Measurements

We deployed OpenWPM on Amazon EC2⁵ instances in three regions: Northern Virginia, United States, Dublin, Ireland, and Tokyo, Japan. We chose these to achieve as much geographic diversity as possible from Amazon’s limited set of regions. Each measurement took place on an m3.medium instance of Ubuntu 14.04 in June 2014. All measurements were ran using 25 simulated profiles for each (user location, browsing model, browser configuration) combination.

When making measurements from within the U.S., we were able to utilize the more realistic AOL browsing model. We used it under several browser configurations: no cookie blocking, blocking third-party cookies from sites which the user has not yet visited as a first-party, blocking all third-party cookies, setting the DNT flag, browsing with HTTPS Everywhere installed, and browsing with Ghostery installed and configured to block all possible entities.

For measurements outside of the United States, we were not able to utilize the AOL browsing model as the search terms and results are likely biased towards U.S. users. Instead, we fall back to the Alexa browsing model when doing comparisons between geographic locations. To compare measurements between the United States, Japan, and Ireland we used an Alexa browsing model localized to the most popular sites within each country.

To run the identifying cookie detection algorithm described in Section 4.5, we also require synchronized measurements of each site visit from two separate machines. We ran these measurements from the Northern Virginia location and visited all of the links which may be visited by any other measurement instance (13,644 links total).

For all measurements, web pages were visited approximately once every ten seconds. We set a 60 second timeout per visit and restarted the browser with consistent state in the event of a crash.

4.4 HTTP Traffic geolocation

In order to determine the if an HTTP request is bound for a specific location of interest, we augment commercially available geolocation data with additional measurement data. After each measurement instance finished browsing, we ran a `traceroute`⁶ to each unique hostname and recorded the full output. All IPs returned in each hop of the traceroute were geo-located with the MaxMind GeoLite2⁷ country databases.

⁴<https://github.com/citp/OpenWPM>

⁵<https://aws.amazon.com/ec2/>

⁶Our traceroutes were configured to use a single probe per hop with a maximum of 25 hops.

⁷<http://dev.maxmind.com/geoip/geoip2/geolite2/>

The mapping between IP and physical location is not one-to-one. Instead, there may be many servers in different locations which all share the same IP address for various purposes. One such example is *anycasting*, the process by which several nodes share the same address and the user is routed to the nearest node.⁸

Thus, when determining if an HTTP request enters a specific country it is not sufficient to simply geolocate the IPs returned from a traceroute to that host. As a solution, we implement a simplified version of the *geo-inconsistency* check proposed by Madory, et.al. [33]. We check that the minimum round-trip time (RTT) returned by a traceroute to each hop is greater than the physical minimum RTT assuming a direct fiber-optic connection between the two locations.

Algorithm 1 summarizes the steps we take to perform this origin-specific geolocation check. Broadly, if the geolocation of a specific hop returns as being within the country of interest, we find the travel time between the (latitude, longitude) pairs of the origin server and the geolocated IP. If geolocated IP’s location is on the country level, we choose the closest point in the geolocated country from the origin location. We then use the haversine formula to calculate the distance between the two points and find the minimum RTT:

$$\text{minRTT} = 2 * \frac{\text{haversine distance} * n}{c}$$

where c is the speed of light in units matching the distance measurement and n is the optical index of the fiber. In our study, we use $n = 1.52$ as the reference optical fiber index.

Data: httpRequest, testCountry

Result: True/False if httpRequest enters testCountry

origin ← latitude/longitude of origin server

hostname ← parse httpRequest.url

for each hop in hostname traceroute **do**

location ← geolocate(hop.IP)

if location not in testCountry **then**

| **continue**

if location not city code **then**

| location ← closest point to origin within country

dist ← haversine(origin,location)

minRTT ← 2 * minimum time to travel dist

if hop.RTT > minRTT **then**

| **return** True

end

return False

Algorithm 1: Origin-specific geolocation check

This check does not guarantee that a specific request enters a country, as network delays could artificially push a traceroute RTT above the threshold. Our assumption of a straight-line connection and optical fiber index is also unlikely to hold in practice. Instead, this check provides a more realistic upper-bound on the amount of traffic an adversary at a specific geographic location can monitor. For example, this check eliminated the numerous examples we observed of traceroutes originating in Ireland and Japan having geolocations within the United States with RTTs of <5ms.

We use a simplified version of this check when examining if requests are exiting the United States. Since a request can

⁸CloudFlare, for example, claims to use anycasting as part of their content delivery network: <https://www.cloudflare.com/features-cdn>

be bound for any non-U.S. destination, we do not attempt to find the closest point in each country. Instead, we only check that the observed RTT is greater than the minimum RTT to the geolocation point regardless of the point’s location.

4.5 Detecting unique identifier cookies

An essential task to quantifying our attack is the ability to detect which cookies are identifying. Identifiers can be stored in many locations (e.g. HTTP cookies, Flash cookies), but to be sent back to trackers the identifiers must be included in HTTP cookies or query parameters of the request. We choose to focus HTTP cookies as they are included in every request and thus provide a generic approach that does not necessitate the parsing of URL parameters for all sites under surveillance. Furthermore, non-cookie tracking techniques are ordinarily paired with tracking cookies; our approach indirectly incorporates an adversary’s capabilities against those technologies.

For our analysis, we are interested in the host (the domain that set the cookie), name, and value fields of a cookie and determine if the data stored in the value field is identifying. This algorithm is a modified version of one we used in a previous study [6].

To be useful to the adversary as identifiers, cookie values must have two important properties: persistence over time and uniqueness across different browser instances. Based on these criteria we develop an algorithm that classifies cookies as identifiers. Our algorithm is intentionally conservative, since false positives risk exaggerating the severity of the attack. Our method does have some false negatives, but this is acceptable since it is in line with our goal of establishing lower bounds for the feasibility of the attack.

We define a cookie to be an identifier cookie if it meets the following criteria: (1) It is *long-lived*, with expiry times longer than three months. The three month cut-off matches our AOL dataset three month window. (2) It’s value is *stable*, and remains constant through all page visits. Dynamic value strings may be timestamps or other non-identifiers. (3) Has a *constant-length* across all our datasets. (4) Is *user-specific*, so the values are unique across different browser instances in our dataset. (5) Has a *high-entropy* value string, with values sufficiently different between machines to enable unique identification. To test for this, we used the Ratcliff-Obershelp [13] algorithm to compute similarity scores between value strings. We filtered out all cookies with value strings that were more than 55% similar to value strings from the corresponding cookies of different measurements.

We run our detection algorithm on the synchronized measurement data described in Section 4.3. By using synchronized measurements, we avoid the problem of sites changing their cookie interaction behavior depending on a user’s browsing time. For instance, in relation to the entropy heuristic, cookies with values that depend on time stamps will be easier to detect and ignore if the crawls have nearly the same timestamps for all actions. For other measurements, we extract identifying cookie values by searching for cookies which match the identifying (host, name) pairs classified in the synchronized measurements.

4.6 Transitive Cookie Linking

Building the graph. Once we determine which cookies contain unique identifiers, we use the `http_requests`, `http_responses`, and `http_cookies` tables of the OpenWPM

crawl database to cluster traffic. From these tables, we construct cookie linking graph using Algorithm 2, which creates a graph with two node types: *URL* nodes and *Cookie* nodes. URL nodes are identified by the tuple (U, <node url>, <request’s geographic destination>) and cookie nodes consisting of the tuple (C, <cookie_value>).

Edges are created under the assumption that a network adversary will be able to link all requests and responses for a single page visit together if he can both follow the chain of referrer and redirect headers for HTTP requests from a single IP. URL — URL edges are created under two conditions: (1) one url node was observed as the `referrer` on a request to the connected url node or (2) one url node was returned in the location field of a 301 or 302 redirect response to the request for the connected url. An adversary is only able to link together different page visits by the shared cookie values loaded on each page. As such, URL — Cookie edges are created whenever a cookie value is observed in the `Cookie` field of an HTTP Request header or the `Set-Cookie` field of an HTTP Response header. Notice that the only linking between separate page visits in the graph occurs when two HTTP requests/responses happen to link to the same Cookie node, while referrer and redirection chaining provides linking within a page visit.

Data: `httpRequests` and `httpResponses` for `useri`

Result: Graph G_i for `useri` with *URL* & *Cookie* nodes

```

for each visited url do
  for httpRequest do
    if HTTPS then
      | continue
    urlNode ← createNode (U, req.url, req.inUS)
    G.addNode(urlNode)
    if req.referrer is not empty then
      | refNode ← createNode (U, ref.url)
      | G.addNode(refNode)
      | G.addEdge(refNode, urlNode, req.inUS)
    if req.cookie is not empty and is identifying then
      | cookieNode ← createNode (C, cookie.value)
      | G.addNode(cookieNode)
      | G.addEdge(cookieNode, urlNode, req.inUS)
    end
  for httpResponse with Set-Cookie do
    if HTTPS then
      | continue
    if cookie is identifying then
      | cookieNode ← createNode (C, cookie.value)
      | urlNode ← node for requested url
      | G.addNode(cookieNode)
      | G.addEdge(cookieNode, urlNode, req.inUS)
    end
  for httpResponse with location field do
    if HTTPS then
      | continue
    urlNode ← node for requested url
    locNode ← createNode (U, loc.url)
    G.addNode(locNode)
    G.addEdge(locNode, urlNode, req.inUS)
  end
end

```

Algorithm 2: Cookie Linking algorithm

Analyzing the graph. In our analysis all graphs only contain traffic for a single user. This allows us to find the connected components within the graph and utilize the giant connected component (GCC) to find the amount of a single user’s traffic an adversary is able to link. Once the GCC is found, we take the intersection of the set of URLs contained in the GCC with the set of URLs visited during the measurement to find the amount of top-level page visits an adversary is able to observe. When the adversary applies the attack in a multi-user setting, they will have many disjoint subgraphs per user of varying size. Depending on the adversary’s goal, these clusters could be processed to link them individually to real world identities, or disambiguated by identifying disjoint sets of cookies for the same sites.

When evaluating adversaries restricted by policy or geographic constraints, an additional pre-processing step is required before finding the GCC. Utilizing the geolocation data from Section 4.4, we are able to filter nodes from the cookie linking graph based on geographic restrictions. In order to determine the amount of traffic for a specific user that a U.S. restricted adversary has access to, we filter all edges from the cookie linking graph that were not added due to U.S. bound requests. We create a subgraph from this filtered edge list and continue the normal linking analysis.

4.7 Identity leakage in popular websites

We conducted a manual analysis of identity leaks on the most popular pages which allow account creation. The top-50 pages are a useful representation for how heavily-used sites with user accounts manage data, and are more likely to be visited by a real user. We identified 50 of the Alexa top 68 U.S. sites that allow for account creation and signed up test accounts when possible. We then examined the homepage, account page, and several random pages on each site to see if any of identifiers are displayed on an HTTP page. If so, an adversary collecting HTTP traffic for that user could inspect the contents of the page to find the identifier and link it to any tracking identifiers also loaded on the page.

5. RESULTS

In the course of our measurements we make nearly 100,000 page visits to 13,644 distinct sites under several client and adversary models. Of these 13,644 sites, nearly all of them (13,266) make requests for external content from a host different than the domain of the host visited.

5.1 Clustering

In this section, we evaluate the effectiveness of our proposed attack under several (adversary, client) models. We are primarily interested in the number of web pages visited by a user which are located in the giant connected component (GCC) relative to the number of pages with embedded third-parties. We focus on large connected components because the probability that a cluster will have at least one page visit that transmits the user’s real-world identity in plaintext increases with the size of the cluster. Manual inspection shows that page visits not in the large connected components belong to small clusters, typically singletons, and are thus unlikely to be useful to the adversary.

An adversary’s incomplete view. We must consider that the adversary’s position on the network may not give them a full view of any user’s traffic. The adversary’s view may be limited due to its policy or geographic restrictions

as described in Section 3, however even with these considerations an adversary may not see all of a user’s traffic. A user may change locations on the network or alternative routes taken by packets through the network might result in gaps in the adversary’s data collection. To model the adversary’s partial view of the user’s browsing activity, we repeat our analysis with random subsets of web pages of various sizes.

For illustration, Figure 2a shows how the GCC of a single AOL user’s page visits (y -axis) grows as we vary the completeness of the adversary’s view of the user’s HTTP traffic (x -axis). Each data point was computed by taking 50 independently random samples of page visits. For each sample we apply the clustering algorithm and compute the fraction contained in the GCC. We then average the fractions across the 50 samples. Since we wish to simulate these page visits being spread out over time, only cookies with expiration times at least three months into the future were included when computing the clusters.

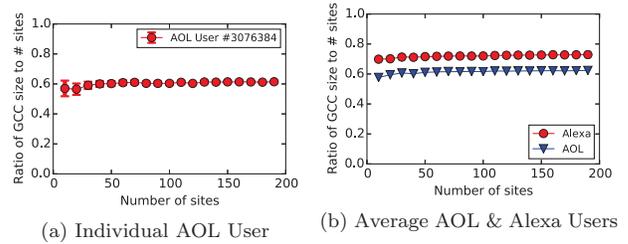


Figure 2: Clustering, random subsets of traffic

Thus for each (x,y) pair we can say that if the adversary captures x web page visits by a user in the course of a wiretap, they could link approximately $y\%$ of those visits into a single cluster. The numbers we see for this user are typical — the fraction is around 55% for even very small clusters and exceeds 60% as the cluster size increases. As discussed in Section 4, we average all results over $N = 25$ client instances for all (client, adversary combinations).

We alternatively examined an adversary who sees subsets of web pages that the user visited in chronological order (perhaps the adversary only observed the user for a short period of time). These results had no statistically significant differences from random subsets. As such, we present the remainder of the graphs using random subsets.

Unrestricted Adversary — AOL User Profiles We first examine the level of clustering an adversary can achieve when it is not subject to any policy or geographic restriction. The results for users simulated under the AOL browsing model and no blocking tools are included in Figure 2b. These results show that the clustering remains very similar to the single user case of Figure 2a, except that no inconsistencies remain. After just 55 web page visits observed by the adversary, the growth of the GCC flattens out to $62.4 \pm 3.2\%$ after 200 sites. For the remaining results, we will only present values for this asymptotic case of 200 sites, as the shape of the GCC growth is nearly identical in all cases.

Unrestricted Adversary — Alexa profiles Next we examine the effect of the browser model on the ability of an unrestricted adversary to cluster user data. We hold the user location and browser configuration set to browsing within the U.S. with no blocking settings. Figure 2b compares the results for Alexa profiles for U.S. users against the AOL profiles. The Alexa clustering shows a similar growth

pattern with an offset around 10% higher on average, with an overall level of clustering after two sites of $72.9 \pm 1\%$.

5.2 U.S. Users Under One-End Foreign

We now consider an adversary located within the United States who is constrained by the “one-end foreign” rule when collecting data on U.S. users. The client model used in evaluating this adversary is U.S.-based users browsing random subsets of the Alexa top-500 U.S. sites with no blocking tools. The size of the largest cluster observed reduces to just $0.9 \pm 0.2\%$ of visited sites averaged across all instances.

To understand why this occurs, we must look at the composition of HTTP traffic. For the average user in this (adversary, client) pair, at least one non-U.S. sub-resource request occurs for 31.7% of the Alexa top-500 sites in the U.S. However, the overall number of HTTP Requests leaving the United States is comparatively small, accounting for just 2.0% of all requests. Only considering traffic where an adversary could learn the top-level domain through the referrer headers, this reduces to 22.7% of visits and 1.6% of requests.⁹ Although nearly a quarter of a user’s browsing history is visible to the adversary, we show that cookie linking is an ineffective method to cluster this traffic.

5.3 Cookie Linking in Non-U.S. Traffic

We now explore the level of clustering that is possible for traffic with a non-U.S. origin. We examine two different cases in this section: we first show the level of clustering possible under the assumption that the adversary will see all web requests that occur for a specific page visit and then we show what an adversary observing U.S.-bound traffic would see. A key point to note is that even if the majority of a website’s resources are requested from a local server, embedded third-parties may cause U.S.-bound requests to occur which have the domain of the first-party as a referrer.

User Location	Unrestricted Adver.	US-based Adver.
Japan	$59.6 \pm 1.2\%$	$20.9 \pm 0.7\%$
Ireland	$63.8 \pm 1.2\%$	$12.8 \pm 1.1\%$

Table 1: Clustering of non-U.S. users by an adversary with no restrictions vs. one restricted to U.S. bound traffic.

Unrestricted Adversary — non-U.S. profiles When all requests are considered, the level of clustering is similar to that of the U.S.-based Alexa user simulations. Table 1 shows amount of clustering that occurs for users in Ireland and Japan under the random subset clustering model. Simulated users in Ireland can expect around 63% of traffic to be clustered, while users in Japan can expect nearly 60%. We believe the differences between user simulations generated using Alexa’s top U.S., Japan, and Ireland lists arises from the difference in the number of included third parties on the first party (i.e., 62, 38, and 30 on average, respectively).

U.S. based Adversary — non-U.S. profiles We then restrict the clustering to only consider requests which are U.S. bound, and cluster based on the information available to a geographically restricted adversary. This could include an adversary within the United States, or an adversary sitting at the entrance to undersea cables returning to the United States. Table 1 shows clustering capabilities of an

⁹These results are broadly consistent with measurements taken by several of this paper’s authors in past work [36].

adversary restricted to these conditions. In Ireland, we see a reduction to around 13% of page visits and in Japan we see a less severe decrease to 20% of visited sites.

5.4 Cookie Linking Under Blocking Tools

We now investigate several ways users may mitigate a clustering attacking using currently available consumer tools. For all blocking configurations, we make measurements using the AOL browsing model and we examine the ability of an unrestricted adversary to cluster traffic. Measurements are run using several different privacy settings within the browser and two popular privacy and security add-ons.

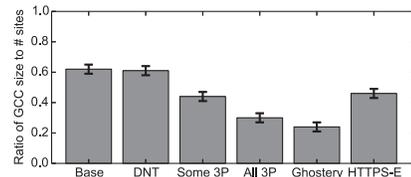


Figure 3: Clustering under several privacy settings.

Baseline displays the level of clustering with no privacy settings or blocking tools enabled. This represents an upper bound on the level of tracking we would expect to see under the other configurations.

DNT had a negligible effect on clustering, showing no statistically significant difference than without blocking.

Cookie blocking policies are more effective, particularly when an adversary sees a low number of page visits. We block cookies for sites that have not been visited in a first-party context by setting “Accept third-party cookies: From visited” in Firefox (this is also the default in Safari). When set, the level of clustering reduces to $43.9 \pm 3.2\%$. Blocking all third-party cookies further reduces this to $30.2 \pm 3.0\%$.

HTTPS Everywhere is an extension created by the EFF to force HTTPS connections whenever available. Since HTTPS requests are not visible to an attacker, and HTTP requests from HTTPS origins will not include a *referrer* (preventing the attacker from linking requests back the the original site). A site can fall into one of four categories: no support for HTTPS, supported but not the default, supported and used by default, and finally, HTTPS-only. This measurement provides a picture of what happens as more sites support and use HTTPS by default. Under our browsing model, the number of HTTPS requests increases from 12.7% to 34.0% and the level of clustering is reduced to $46.1 \pm 3.2\%$.

Ghostery is a popular list-based browser extension for blocking third-party requests to domains considered to be trackers. This proves to be the most effective solution for users, reducing the level of clustering to $24.2 \pm 2.8\%$ of visited sites. Enabling Ghostery and configuring it to block as much as possible reduces the average number of inclusions from external hosts to just 5.2 per first-party.

5.5 Identity Leakage

Table 2 summarizes our results from a manual survey of the Alexa U.S. sites. We picked the top 50 sites that support account creation. 44 of the 50 websites used HTTPS to secure login pages.¹⁰ Only 19 of those sites continued to

¹⁰5 of the remaining 6 made POSTs with credentials and 1 made a GET with the credentials as a URL query parameter

use HTTPS to secure future interactions with the user after logged in. We summarize the cleartext identity leaks for the websites which no longer continue to use HTTPS after login.

Although a majority of sites secure user credentials on login pages, personally identifying information (name, username, email address) is transmitted much more frequently via HTTP. Over half of the surveyed sites leak at least one type of identifier, and 42% (not shown in table) leak either username or email address, which can be used to uniquely infer the user’s real-world identity. Past work [34, 29] has found a roughly equivalent level of leakage to occur through the Request-URI and Referer.

A representative example of the web’s identity-leak problem is `imdb.com`. IMDB provides a secure login page, but once logged in, users return to an HTTP page. This page includes the user’s full name on the homepage of the site. Every time a user visits the site while logged in, a passive attacker can extract the name from the unencrypted traffic.

Plaintext Leak Type	Percentage of Sites
First Name	28%
Full Name	14%
Username	36%
Email Address	18%
At least one of the above	56%

Table 2: Leakage on Alexa Top 50 supporting user accounts

Furthermore, we verified that pages from these popular sites that leak identity occur in the clusters of web pages found in our attack. Specifically, at least 5 (and an average 9.92) of the 28 sites we found to leak some type of identity were found in the giant connected component of every one of the 25 Alexa U.S. users. Due to global popularity of the top-50 U.S. sites, an average of 4.4 and 6.4 of these identity leaking sites are found in the GCC’s of the Alexa Japan and Alexa Ireland users, respectively. Additionally, for the AOL profiles with no blocking, 9 of the 25 simulated users had at least 1 identity leaker in the GCC. Of course, there are likely also many sites outside the top 50 that leak identity and are found in these clusters, but we did not measure these.

Taken together with our results on clustering, our measurements show that a passive attack is highly feasible: after observing only a fraction of a user’s web traffic the adversary will be able to link the majority of the user’s web requests together and furthermore, use the contents of the responses to infer the user’s real-world identity.

6. DISCUSSION

6.1 Linking without IP address

So far we have assumed that the adversary sees the same source IP address on a request to a first-party site and its corresponding third-party tracker, and that this can be used to link the two requests. There are at least two scenarios in which this assumption is problematic. The first is a NAT. If two users, Alice and Bob, behind the same NAT visit the same web page at roughly the same time, the adversary sees the same IP address on all ensuing HTTP requests. The other scenario is when the user employs Tor without proper application layer anonymization, and the adversary is able to sniff cookies only on the path from the exit node to the web server. (If the user is using a properly configured Tor

setup, such as the Tor browser bundle, this attack does not work at all). Since Tor will, in general, use different circuits for communicating with different servers, the adversary will see different source IPs for the two requests (or may be able to observe only one of the requests).

However the well-known “intersection attack” can be used to link requests without using the IP address: if a cookie value a associated with page A’s domain and a cookie value x associated with an embedded tracker domain X are observed *multiple times* near-simultaneously (e.g. within 1 second of each other), then a and x are probably associated with the same user. Intuition suggests that for all but the busiest of web pages, two or three visits may be sufficient to link the first-party and tracker cookies with each other. However, this claim cannot be rigorously evaluated without access to large-scale HTTP traffic and so we leave this as a hypothesis.

6.2 NSA Surveillance

Our results bear directly on the NSA’s technical capabilities, against individuals both within and external to the United States. For non-U.S. individuals, our data indicates that the agency could have access to a majority of a person’s browsing history (71.3% of visits in Ireland and 61.1% of visits in Japan), without ever collecting data outside the United States.¹¹ Furthermore, we show that the agency can link a non-trivial portion of this traffic through cookies.

Nearly a quarter of U.S. page visits are visible outside the U.S. through third parties and referers. While we show that linking is infeasible, this does not imply that U.S. users browsing domestic sites are safe from NSA surveillance. Passive tracking techniques, or a static IP address, could cause the user’s traffic to be just as vulnerable.

6.3 Mitigation by users

Figure 3 shows that users can minimize their exposure to surveillance through cookies, but can not eliminate it all together. Since the identifiers of advertising networks play a key part in the transitive linking of page visits, ad filtering lists (e.g. Ghostery) are currently the most effective solution for users. However even after blocking these lists, ISP level identifiers like Verizon’s UIDH would cause the vulnerability to persist [26]. Firefox’s built-in cookie blocking can also be effective in reducing the level of traffic clustering, though even the most restrictive option leaves nearly a third of a user’s traffic vulnerable to clustering.

Users have very little control over identity leakage outside of stopping the use of services. Forcing HTTPS connections after login can help (i.e. using HTTPS Everywhere), but as we show in our measurements, two-thirds of sites still do not support HTTPS after this step is taken. Users can also be careful to not reuse usernames between sites as this could provide an additional identifier to link page visits.

6.4 Mitigation by trackers

Trackers can prevent a passive eavesdropper from piggy-backing on their unique identifiers if they are only transmitted over HTTPS. Some trackers have already deployed HTTPS to avoid mixed content warnings when embedding in HTTPS pages. There are also subtle issues, such session

¹¹Alternatively, the agency could rely on collection points just outside the U.S. That approach would fall under Executive Order 12333, which affords more latitude than Section 702 of the FISA Amendments Act.

tickets used for TLS session resumption, which can be used by an eavesdropper to link multiple HTTPS connections to the same browser instance similar to a cookie.

Unfortunately, a large fraction of trackers would need to deploy such mitigation strategies for them to make a dent in the adversary’s overall chances. As we showed in Section 5.4 with our HTTPS Everywhere measurements, the feasibility of traffic clustering only drops by 23% when the amount requests occurring over HTTPS more than doubles.

6.5 Limitations

A couple of important limitations of the attack must be pointed out. First, using the Tor browser bundle likely defeats it. “Cross-origin identifier unlinkability” is a first-order design goal of the Tor browser, which is achieved through a variety of mechanisms such as double-keying cookies by first-party and third-party [44]. In other words, the same tracker on different websites will see different cookies. However, our measurements on identifier leakage on popular websites apply to Tor browser usage as well. Preventing such leakage is not a Tor browser design goal.

Simply disabling third-party cookies will also deter the attack, but it is not clear if it will completely stop it. There are a variety of stateful tracking mechanisms in addition to cookies [14, 23, 51, 37, 6], although most are not as prevalent on the web as cookies are.

We also mention two limitations of our study. First, while a significant fraction of popular sites transmit identities of logged-in users in the clear, we have not actually measured how frequently typical users are logged in to the sites that do so. Anecdotal evidence suggests that this number must be high, but experiments on actual user sessions are required for an accurate estimation of vulnerability.

Second, we use commercial geolocation data with an additional custom metric to determine if requests enter the United States for users in Japan and Ireland, and to determine if requests leave the United States for users within the U.S. Even with this additional check, two scenarios can occur: requests outside the U.S. can be marked as entering the U.S. due to an incorrect geolocation and high network latency, and requests inside the U.S. can be marked as staying in the U.S. if they are incorrectly geolocated as being further away than the actual location of the destination (which may still be external to the U.S.).

6.6 Other applications of our methodology

Stated in general terms, we study an adversary with a given set of technical capabilities, network positions, and policy restrictions, and ask, for a given user browsing model, how much of her activity is vulnerable. Our general algorithm in Section 4 can be easily adapted to study a variety of questions that fit this framework. For example, we might assume an adversary who can compel certificate creation and uses this to spoof some third parties on secure websites to launch active attacks. Here the relevant measurement would be the prevalence of active third-party content on websites visited by typical users and the geographic distribution of third parties serving such content.

7. CONCLUSION

While much has been said from a legal, ethical and policy standpoint about the recent revelations of NSA tracking, many interesting *technical* questions deserve to be consid-

ered. In this paper we studied what can be inferred from the surveillance of web traffic and established that utilizing third-party tracking cookies enables an adversary to attribute traffic to users much more effectively than methods such as considering IP address alone. We hope that these findings will inform the policy debate on tracking, raise user awareness of subtle, inferential privacy breaches, and lead to the development of better defenses and greater adoption of existing ones.

8. ACKNOWLEDGMENTS

We would like to thank Jennifer Rexford, Doug Madory, Harlan Yu, and our anonymous reviewers for their insightful comments, as well as Andrew Clement and Colin McCann for their willingness to share data related to this study.

9. REFERENCES

- [1] ShareMeNot: Protecting against tracking from third-party social media buttons while still allowing you to use them. <https://sharemenot.cs.washington.edu>.
- [2] TrackingObserver: A Browser-Based Web Tracking Detection Platform. <http://trackingobserver.cs.washington.edu>.
- [3] Executive Order 12333—United States intelligence activities. <http://www.archives.gov/federal-register/codification/executive-order/12333.html>, 1981.
- [4] NSA ‘planned to discredit radicals over web-porn use’. <http://www.bbc.co.uk/news/technology-25118156>, November 2013.
- [5] ‘Tor Stinks’ presentation - read the full document. <http://www.theguardian.com/world/interactive/2013/oct/04/tor-stinks-nsa-presentation-document>, October 2013.
- [6] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. The Web never forgets: Persistent tracking mechanisms in the wild. In *Conference on Computer and Communications Security (CCS)*. ACM, 2014.
- [7] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel. FPDetective: dusting the web for fingerprinters. In *Conference on Computer and Communications Security (CCS)*. ACM, 2013.
- [8] A. Arnbak and S. Goldberg. Loopholes for circumventing the constitution: Warrantless bulk surveillance on americans by collecting network traffic abroad, 2014.
- [9] M. Ayenson, D. J. Wambach, A. Soltani, N. Good, and C. J. Hoofnagle. Flash cookies and privacy II: Now with HTML5 and ETag respawning. *World Wide Web Internet And Web Information Systems*, 2011.
- [10] M. Balakrishnan, I. Mohamed, and V. Ramasubramanian. Where’s that phone?: geolocating IP addresses on 3G networks. In *Internet Measurement Conference (IMC)*. ACM, 2009.
- [11] R. Balebako, P. Leon, R. Shay, B. Ur, Y. Wang, and L. Cranor. Measuring the Effectiveness of Privacy Tools for Limiting Behavioral Advertising. In *Web 2.0 Security & Privacy (W2SP)*. IEEE, 2012.
- [12] J. Ball. NSA stores metadata of millions of web users for up to a year, secret files show. <http://www.theguardian.com/world/2013/sep/30/nsa-americans-metadata-year-documents>, 2013.

- [13] P. E. Black. Ratcliff/Obershelp pattern recognition. <http://xlinux.nist.gov/dads/HTML/ratcliffObershelp.html>, December 2004.
- [14] E. Bursztein. Tracking users that block cookies with a HTTP redirect. <http://www.elie.net/blog/security/tracking-users-that-block-cookies-with-a-http-redirect>, 2011.
- [15] S. Chen, R. Wang, X. Wang, and K. Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *Security and Privacy (S&P)*. IEEE, 2010.
- [16] A. Clement. IXmaps—Tracking your personal data through the NSA’s warrantless wiretapping sites. In *International Symposium on Technology and Society (ISTAS)*. IEEE, 2013.
- [17] B. Elgin and V. Silver. The Surveillance Market and Its Victims. <http://www.bloomberg.com/data-visualization/wired-for-repression/>, 2011.
- [18] S. Englehardt, C. Eubank, P. Zimmerman, D. Reisman, and A. Narayanan. Web Privacy Measurement: Scientific principles, engineering platform, and new results. Manuscript, 2014.
- [19] R. Gallagher. Operation Socialist: The Inside Story of How British Spies Hacked Belgium’s Largest Telco. <https://firstlook.org/theintercept/2014/12/13/belgacom-hack-gchq-inside-story/>, 2014.
- [20] Ghostery. Are we private yet? <http://www.arenweprivateyet.com/>.
- [21] S. Gorman and J. Valentino-Devries. New Details Show Broader NSA Surveillance Reach. <http://online.wsj.com/news/articles/SB10001424127887324108204579022874091732470>, 2013.
- [22] G. Greenwald and S. Ackerman. How the NSA is still harvesting your online data. <http://www.theguardian.com/world/2013/jun/27/nsa-online-metadata-collection>, 2013.
- [23] M. Hastak and M. J. Culnan. Persistent and unblockable cookies using HTTP headers. <http://www.nikcub.com/posts/persistent-and-unblockable-cookies-using-http-headers>, 2011.
- [24] D. Herrmann, R. Wendolsky, and H. Federrath. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naive-Bayes Classifier. In *Workshop on Cloud Computing Security (CCSW)*. ACM, 2009.
- [25] A. Hintz. Fingerprinting Websites Using Traffic Analysis. In *Privacy Enhancing Technologies*. Springer, 2003.
- [26] J. Hoffman-Andrews. Verizon Injecting Perma-Cookies to Track Mobile Customers, Bypassing Privacy Controls. <https://www.eff.org/deeplinks/2014/11/verizon-x-uidh>, 2014.
- [27] B. Krishnamurthy, K. Naryshkin, and C. Wills. Privacy leakage vs. Protection measures: the growing disconnect. In *Web 2.0 Security & Privacy (W2SP)*. IEEE, 2011.
- [28] B. Krishnamurthy and C. Wills. Privacy diffusion on the Web: a longitudinal perspective. In *International Conference on World Wide Web (WWW)*. ACM, 2009.
- [29] B. Krishnamurthy and C. E. Wills. On the Leakage of Personally Identifiable Information Via Online Social Networks. In *Workshop on Online Social Networks (WOSN)*. ACM, 2009.
- [30] B. Krishnamurthy and C. E. Wills. Privacy leakage in mobile online social networks. In *Conference on Online Social Networks (COSN)*. USENIX, 2010.
- [31] M. Lee. Secret “BADASS” Intelligence Program Spied on Smartphones. <https://firstlook.org/theintercept/2015/01/26/secret-badass-spy-program/>, 2015.
- [32] B. Liu, A. Sheth, U. Weinsberg, J. Chandrashekar, and R. Govindan. AdReveal: Improving Transparency Into Online Targeted Advertising. In *Workshop on Hot Topics in Networks (HotNets)*. ACM, 2013.
- [33] D. Madory, C. Cook, and K. Miao. Who Are the Anycasters? In *Proceedings of NANOG59*, 10 2013.
- [34] D. Malandrino, A. Petta, V. Scarano, L. Serra, and R. Spinelli. Privacy awareness about information leakage: Who knows what about me? In *Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2013.
- [35] J. Mayer. Tracking the Trackers: Self-Help Tools. <https://cyberlaw.stanford.edu/blog/2011/09/tracking-trackers-self-help-tools>, September 2011.
- [36] J. Mayer and E. W. Felten. The Web is Flat. <http://webpolicy.org/2013/10/30/the-web-is-flat/>, 2013.
- [37] J. R. Mayer and J. C. Mitchell. Third-party web tracking: Policy and technology. In *Security and Privacy (S&P)*. IEEE, 2012.
- [38] A. M. McDonald and L. F. Cranor. Survey of the use of Adobe Flash local shared objects to respawn HTTP cookies. *ISJLP*, 7:639, 2011.
- [39] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of Tor. In *Security and Privacy (S&P)*. IEEE, 2005.
- [40] S. J. Murdoch and P. Zieliński. Sampled Traffic Analysis by Internet-Exchange-Level Adversaries. In *Privacy Enhancing Technologies*. Springer, 2007.
- [41] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Security and Privacy (S&P)*. IEEE, 2013.
- [42] L. Olejnik, T. Minh-Dung, C. Castelluccia, et al. Selling Off Privacy at Auction. 2013.
- [43] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2011.
- [44] M. Perry, E. Clark, and S. Murdoch. The design and implementation of the Tor browser [DRAFT]. <https://www.torproject.org/projects/torbrowser/design>, November 2014.
- [45] F. Roesner, T. Kohno, and D. Wetherall. Detecting and Defending Against Third-Party Tracking on the Web. In *Networked Systems Design and Implementation (NDSI)*. USENIX, 2012.
- [46] 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.
- [47] A. Soltani, A. Peterson, and B. Gellman. NSA uses Google cookies to pinpoint targets for hacking. <http://www.washingtonpost.com/blogs/the->

- switch/wp/2013/12/10/nsa-uses-google-cookies-to-pinpoint-targets-for-hacking, December 2013.
- [48] D. X. Song, D. Wagner, and X. Tian. Timing Analysis of Keystrokes and Timing Attacks on SSH. In *Security Symposium*. USENIX, 2001.
 - [49] A. M. White, A. R. Matthews, K. Z. Snow, and F. Monroe. Phonotactic reconstruction of encrypted VoIP conversations: Hookt on fon-iks. In *Security and Privacy (S&P)*. IEEE, 2011.
 - [50] T.-F. Yen, Y. Xie, F. Yu, R. P. Yu, and M. Abadi. Host fingerprinting and tracking on the web: Privacy and security implications. In *Network and Distributed System Security Symposium (NDSS)*. IEEE, 2012.
 - [51] M. Zalewski. Rapid history extraction through non-destructive cache timing (v8). <http://lcamtuf.coredump.cx/cachetime/>.

Online tracking: A 1-million-site measurement and analysis

Draft: May 18, 2016

Steven Englehardt
Princeton University
ste@cs.princeton.edu

Arvind Narayanan
Princeton University
arvindn@cs.princeton.edu

Abstract

We present the largest and most detailed measurement of online tracking conducted to date, based on a crawl of the top 1 million websites. We make 15 types of measurements on each site, including stateful (cookie-based) and stateless (fingerprinting-based) tracking, the effect of browser privacy tools, and the exchange of tracking data between different sites (“cookie syncing”). Our findings include multiple sophisticated fingerprinting techniques never before measured in the wild.

This measurement is made possible by our web privacy measurement tool, OpenWPM, which uses an automated version of a full-fledged consumer browser. It supports parallelism for speed and scale, automatic recovery from failures of the underlying browser, and comprehensive browser instrumentation. OpenWPM is open-source¹ and has already been used as the basis of seven published studies on web privacy and security.

1 Introduction

Web privacy measurement — observing websites and services to detect, characterize and quantify privacy-impacting behaviors — has repeatedly forced companies to improve their privacy practices due to public pressure, press coverage, and regulatory action [5, 13]. On the other hand, web privacy measurement presents formidable engineering and methodological challenges. In the absence of a generic tool, it has been largely confined to a niche community of researchers.

We seek to transform web privacy measurement into a widespread practice by creating a tool that is useful not just to our colleagues but also to regulators, self-regulators, the press, activists, and website operators, who are often in the dark about third-party tracking on their own domains. We also seek to lessen the burden

of *continual* oversight of web tracking and privacy, by developing a robust and modular platform for repeated studies.

Our tool, **OpenWPM** (Section 3), solves three key systems challenges faced by the web privacy measurement community. It does so by building on the strengths of past work, while avoiding the pitfalls made apparent in by previous engineering efforts. (1) We achieve scale through the parallelism and robustness by utilizing isolated measurement processes similar to FPDetective’s platform [2], while still supporting stateful measurements. We’re able to scale to 1 million sites, without having to resort to a stripped-down browser [29] (a limitation we explore in detail in Section 3.2). (2) We provide comprehensive instrumentation by building upon the rich browser extension instrumentation of FourthParty [32], without requiring the researcher to write their own automation code. (3) We reduce duplication of work by providing a modular architecture to enable code re-use between studies.

Solving these problems is hard because the web is not designed for automation or instrumentation. Selenium,² the main tool for automated browsing through a full-fledged browser, is intended for developers to test their *own* websites. As a result it performs poorly on websites not controlled by the user and breaks frequently if used for large-scale measurements. Browsers themselves tend to suffer memory leaks over long sessions. In addition, *instrumenting* the browser to collect a variety of data for later analysis presents formidable challenges. For full coverage, we’ve found it necessary to have three separate measurement points: a network proxy, a browser extension, and a disk state monitor. Further, we must link data collected from these disparate points into a uniform schema, duplicating much of the browser’s own internal logic in parsing traffic.

OpenWPM is a mature open-source tool that has

¹<https://github.com/citp/OpenWPM>

²<http://www.seleniumhq.org/>

already been used to provide the measurements of seven published studies since 2014 (Section 3.3). These studies have already led to improvements and fixes to privacy and security. These experiments performed in these studies have crucially benefited from several advanced features of OpenWPM such as the ability to automatically log into websites using specified credentials. Using these case studies, we show in detail how OpenWPM’s capabilities enable quickly designing and running an experiment (Section 3.2).

A large-scale view of web tracking and privacy. Since June 2015 we have been conducting regular measurements of online tracking, incrementally adding features and fixing scale bottlenecks. The results we report in this paper (Section 4) are all based on our January 2016 measurement of the top 1 million sites. In future work, we will publish additional analyses of the evolution of tracking and privacy over time.

Our scale enables a variety of new insights. We observe for the first time that online tracking has a “long tail”, but we find a surprisingly quick drop-off in the scale of individual trackers: trackers in the tail are found on very few sites (Section 5.1). Using a new metric for quantifying tracking (Section 5.2), we find that the tracking-protection tool Ghostery (<https://www.ghostery.com/>) is effective, with some caveats (Section 5.5). We quantify the impact of trackers and third parties on HTTPS deployment (Section 5.3) and show that cookie syncing is pervasive (Section 5.6).

Turning to browser fingerprinting, we revisit an influential 2014 study on canvas fingerprinting [1] with updated and improved methodology (Section 6.1). Next, we report on several types of fingerprinting never before measured at scale: font fingerprinting using canvas (which is distinct from canvas fingerprinting; Section 6.2), and fingerprinting by abusing the WebRTC API (Section 6.3), the AudioContext API (Section 6.4), and the Battery Status API (6.5). Finally, we show that in contrast to our results in Section 5.5, existing privacy tools are *not* effective at detecting these newer and more obscure fingerprinting techniques.

Overall, our results show cause for concern, but also encouraging signs. In particular, several of our results suggest that while online tracking presents few barriers to entry, trackers in the tail of the distribution are found on very few sites and are far less likely to be encountered by the average user. Those at the head of the distribution, on the other hand, are owned by relatively few companies and are responsive to the scrutiny resulting from privacy studies.

We envision a future where measurement provides a key layer of oversight of online privacy. This will be especially important given that perfectly anticipating and preventing all possible privacy problems (whether

through blocking tools or careful engineering of web APIs) has proved infeasible. To enable such oversight, we plan to make all our data publicly available (OpenWPM is already open-source). We expect that measurement will be useful to developers of privacy tools, to regulators and policy makers, journalists, and many others.

2 Background and related work

Background: third-party online tracking. As users browse and interact with websites, they are observed by both “first parties,” which are the sites the user visits directly, and “third parties” which are typically hidden trackers such as ad networks embedded on most web pages. Third parties can obtain users’ browsing histories through a combination of cookies and other tracking technologies that allow them to uniquely identify users, and the “referrer” header that tells the third party which first-party site the user is currently visiting. Other sensitive information such as email addresses may also be leaked to third parties via the referrer header.

Web privacy measurement platforms. There are two main ways to collect large-scale data for web privacy measurement: crowd-sourcing and simulating users, i.e., running bots. Our focus is on the latter type, but there are many similarities between the two types of studies.

The closest comparisons to OpenWPM are other open web privacy measurement platforms, which we now review. We consider a tool to be a platform if it is publicly available and there is some generality to the types of studies that can be performed using it. In some cases, OpenWPM has directly built upon existing platforms, which we make explicit note of.

FPDetective is the most similar platform to OpenWPM. *FPDetective* uses a hybrid PhantomJS and Chromium based automation infrastructure [2], with both native browser code and a proxy for instrumentation. In the published study, the platform was used for the detection and analysis of fingerprinters, and much of the included instrumentation was built to support that. The platform allows researchers to conduct additional experiments by replacing a script which is executed with each page visit, and the authors state the platform can be easily extended for non-fingerprinting studies.

OpenWPM differs in several ways from *FPDetective*: (1) it supports both stateful and stateless measurements, whereas *FPDetective* only supports stateless (2) it includes generic instrumentation for both stateless and stateful tracking, enabling a wider range of privacy studies without additional changes to the infrastructure (3) none of the included instrumentation requires native browser code, making it easier to upgrade to new or different versions of the browser, and (4) OpenWPM uses

a high-level command-based architecture, which allows purpose built commands to be re-used between studies.

Chameleon Crawler is a Chromium based crawler that utilizes the Chameleon³ browser extension for detecting browser fingerprinting. Chameleon Crawler uses similar automation components, but supports a subset of OpenWPM’s instrumentation.

FourthParty is a Firefox plug-in for instrumentation and does not handle automation [32]. OpenWPM has incorporated and expanded upon nearly all of FourthParty’s instrumentation within its own extension (Section 3).

TrackingObserver is a Chrome extension that detects tracking and exposes APIs for extending its functionality such as measurement and blocking [46].

XRay is a platform for *differential correlation*: inferring input-output relationships in any personalized web service [26]. XRay handles the analysis phase of web privacy measurement in a generic way, but not driving the browser or instrumentation. This is precisely the converse of OpenWPM, suggesting the exciting possibility of using the two tools in concert to achieve an even greater degree of generic automation.

AdFisher is a tool for running automated experiments on personalized ad settings [8]. It contains a barebones automation framework with similar components as ours (Selenium, xvfb), but the key technology is a machine-learning system for causality attribution. Again there is the possibility of running OpenWPM’s automation and instrumentation together with AdFisher’s analytic component.

WebXray is a PhantomJS based tool for measuring HTTP traffic [29]. It has been used to study third-party inclusions on the top 1 million sites, but as we show in Section 3.1, measurements with a stripped-down browser have the potential to miss a large number of resource loads.

Several research groups have built or deployed crowd-sourcing platforms for web privacy measurement, including \$heriff and Bobble [34, 60]. Some challenges here include providing value to users to incentivize participation, participant privacy, etc.

Previous findings. Krishnarmurthy and Wills [23] provide much of the early insight into web tracking, showing the growth of the largest third-party organizations from 10% to 20-60% of top sites between 2005 and 2008. Roesner et al. provide a classification framework for third-party domains, arguing “it is incorrect to bundle together different classes of trackers”, such as cross-site trackers and analytics trackers [47]. In the years following Krishnarmurthy’s measurements, studies show a continual increase in third-party tracking and in the di-

versity of tracking techniques [1, 2, 4, 19, 32, 47]. More recently, Libert studies third-party HTTP requests on the top 1 million sites [29], providing view of tracking across the web. In this study, Libert compiled a comprehensive mapping of third-party domains to organizations, showing that Google can track users across nearly 80% of sites through its various third-party domains.

Web tracking has expanded from simple HTTP cookies to include more persistent tracking techniques. Soltani et al. first examined the use of flash cookies to “respawn” or re-instantiate HTTP cookies [50], and Ayenson et al. showed how sites were using cache E-Tags and HTML5 localStorage for the same purpose [6]. These discoveries led to media backlash [28, 35] and legal settlements [9, 49] against the companies participating in the practice. However, several follow up studies by other research groups confirmed that, despite a reduction in usage (particularly within the US), the technique is still used for tracking [1, 33, 47].

Device fingerprinting is a persistent tracking technique which does not require a tracker to set any state in the user’s browser. Instead, trackers attempt to identify users by a combination of the device’s properties. Within samples of over 100,000 browsers, 80-90% of desktop and 81% of mobile devices have a unique fingerprint [10, 25]. New fingerprinting techniques are continually discovered [14, 36, 41], and are subsequently used to track users on the web [1, 2, 39]. In Section 6.1 we present several new fingerprinting techniques discovered during our measurements.

Personalization measurement. Measurement of tracking is closely related to measurement of personalization, since the question of what data is collected leads to the question of how that data is used. The primary purpose of online tracking is behavioral advertising — showing ads based on the user’s past activity. Datta et al. highlight the incompleteness of Google’s Ad Settings transparency page and provide several empirical examples of discriminatory and predatory ads [8]. Lécuyer et al. develop XRay, a system for inferring which pieces of user data are used for personalization [26]. Another system by some of the same authors is Sunlight which improves upon their previous methodology to provide statistical confidence of their targeting inferences [27].

Many other practices that raise privacy or ethical concerns have been studied: *price discrimination*, where a site shows different prices to different consumers for the same product [18, 58]; *steering*, a gentler form of price discrimination where a product search shows differently-priced results for different users [31]; and the *filter bubble*, the supposed effect that occurs when online information systems personalize what is shown to a user based on what the user viewed in the past [60].

Web security measurement. Web security studies of-

³<https://github.com/ghostwords/chameleon>

ten use similar methods as web privacy measurement, and the boundary is not always clear. Yue and Wang modified the Firefox browser source code in order to perform a measurement of insecure Javascript implementations on the web [61]. Nikiforakis et al. utilized a headless browser to measure the amount of third-party Javascript inclusions across many popular sites and the vulnerabilities that arise from how the script is embedded [38]. Van Goethem et al. likewise used a headless browser to measure the presence of security seals on the top 1 million sites [57]. Zarras et al. used Selenium to drive crawls that measured and categorized malicious advertisements displayed while browsing popular sites [62]. Rafique et al. also used a Selenium-based crawler to measure the presence of malware and other vulnerabilities on live streaming websites [44]. Other studies have analyzed Flash and Javascript elements of webpages to measure security vulnerabilities and error-prone implementations [40, 56].

3 Measurement Platform

An infrastructure for automated web privacy measurement has three components: simulating users, recording observations (response metadata, cookies, behavior of scripts, etc.), and analysis. We set out to build a platform that can automate the first two components and can ease the researcher’s analysis task. We sought to make OpenWPM general, modular, and scalable enough to support essentially any privacy measurement.

3.1 Design and Implementation

We divided our browser automation and data collection infrastructure into three main modules: *browser managers* which act as an abstraction layer for automating individual browser instances, a user-facing *task manager* which serves to distribute commands to browser managers, and a *data aggregator*, which acts as an abstraction layer for browser instrumentation. The researcher interacts with the task manager via an extensible, high-level, domain-specific language for crawling and controlling the browser instance. The entire platform is built using Python and Python-compatible libraries.

Browser driver: Providing realism and support for web technologies. We considered a variety of choices to *drive* measurements, i.e., to instruct the browser to visit a set of pages (and possibly to perform a set of actions on each). The two main categories to choose from are lightweight browsers like PhantomJS (an implementation of WebKit), and full-fledged browsers like Firefox and Chrome. We chose to use Selenium, a cross-platform web driver for Firefox, Chrome, Internet Explorer, and

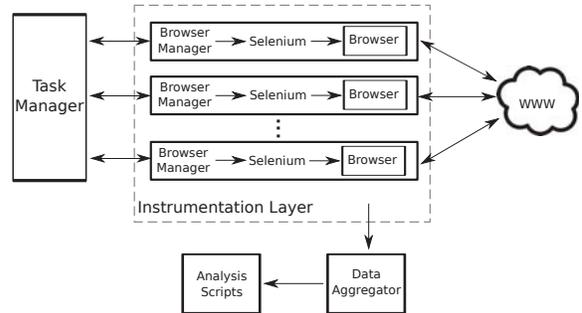


Figure 1: High-level overview of OpenWPM

The task manager monitors browser managers, which convert high-level commands into automated browser actions. The data aggregator receives and pre-processes data from instrumentation.

PhantomJS. We currently use Selenium to drive Firefox, but Selenium’s common interface into all browsers makes it easy to support other browsers in the future.

By using a consumer browser, all technologies that a typical user would have access to (e.g., HTML5 storage options, Adobe Flash) are also supported by measurement instances. The alternative, PhantomJS, does not support WebGL, HTML5 Audio and Video, CSS 3-D, and browser plugins (like Flash), making it impossible to run measurements on the use of these technologies [43].

In retrospect this has proved to be a sound choice. Without full support for new web technologies we would not have been able to discover and measure the use of the `AudioContext` API for device fingerprinting as discussed in Section 6.4.

We can quantify the loss of coverage if we were to use a stripped-down browser. A recent study that used PhantomJS to visit the Alexa top 1 million sites found a total of roughly 35 million requests to 21 million resources [29]. By contrast, in our top 1 million site measurement, we find a total of roughly 90 million requests to 68 million resources. We further study the completeness of OpenWPM in Section 3.2.

Finally the use of real browsers also allows us to test the effects of consumer browser extensions. We support running measurements with extensions such as Ghostery and HTTPS Everywhere as well as enabling Firefox privacy settings such third-party cookie blocking and the new Tracking Protection feature. New extensions can easily be supported with only a few extra lines of code. See Section 5.3 and Section 5.5 for analyses of measurements run with these browser settings.

Browser managers: Providing stability. During the course of a long measurement, a variety of unpredictable events such as page timeouts or browser crashes could halt the measurement’s progress or cause data loss or corruption. A key disadvantage of Selenium is that it frequently hangs indefinitely due to its blocking API [48],

as it was designed to be a tool for webmasters to test their own sites rather than an engine for large-scale measurements. Browser managers provide an abstraction layer around Selenium, isolating it from the rest of the components.

Each browser manager instantiates a Selenium instance with a specified configuration of user preferences, such as blocking third-party cookies. It is responsible for converting high-level platform commands (e.g. visiting a site) into specific Selenium subroutines. It encapsulates per-browser preferences and state, enabling recovery from browser failures. To isolate failures, each browser manager runs as a separate process.

We support launching measurement instances in a “headless” container, enabling greater parallelization due to lower memory consumption and deploying measurements on remote machines. Full browsers have no headless option as they are built for graphical user interaction. To solve this problem, we use the `pyvirtualdisplay` tool to interface with `Xvfb`, which draws the graphical interface of the browser to a virtual frame buffer. They retain the ability to easily generate screenshots of rendered sites when necessary.

Task manager: Providing scalability and abstraction. The task manager provides a scriptable command-line interface for controlling multiple browsers simultaneously. Commands can be distributed to browsers either synchronized or first-come-first-serve. Each command is launched in a per-browser command execution thread.

The command-execution thread handles errors in its corresponding browser manager automatically. If the browser manager crashes, times out, or exceeds memory limits, the thread enters a crash recovery routine. In this routine, the manager archives the current browser profile, kills all current processes, and loads the archive (which includes cookies and history) into a fresh browser with the same configuration options.

Data Aggregator: Providing repeatability. To promote scientific rigor, the platform should enable researchers to easily reproduce experiments. This can be achieved by logging data in a standardized format, so research groups can easily share scripts and data. To support logging in a standard schema, we use data aggregation components to gather results from all instrumentation components in a central and structured location. The data aggregator receives data during the measurement, manipulates it as necessary, and saves it on disk keyed back to a specific page visit and browser. The aggregator exists within its own process, and is accessed through a socket interface which can easily be connected to from any number of browser managers or instrumentation processes.

We currently support two data aggregators: a structured SQLite aggregator for storing relational data and

a LevelDB aggregator for storing compressed web content. The SQLite aggregator stores the majority of the measurement data, including data from both the proxy and the extension (described below). The LevelDB aggregator is designed to store de-duplicated web content, such as Javascript or HTML files. The aggregator checks if a hash of the content is present in the database, and if not compresses the content and adds it to the database. The hash can be used as a key to access the content during analysis and to link it with data in the SQLite database (such as an HTTP response).

In comparison to client-server databases, local databases do not require end-user setup of a database server and enable researchers to easily share self-contained files. However, the use of socket connections in both aggregators simplify the process of migrating to other server-based solutions such as MySQL or cloud hosted databases as studies grow in scale.

Instrumentation: Supporting compatibility and modularity. We provide the researcher with data access at several points: (1) raw data on disk, (2) at the network level with an HTTP proxy, and (3) at the Javascript level with a Firefox extension. This provides nearly full coverage of a browser’s interaction with the web and the system. Each level of instrumentation keys data with the top level site being visited and the current browser id, making it possible to combine measurement data from multiple instrumentation sources for each page visit.

Disk Access — We include instrumentation that collects changes to Flash LSOs and the Firefox cookie database after each page visit. This allows a researcher to determine which sites and third parties are setting Flash cookies, and to record access to cookies in the absence of an HTTP proxy or Firefox extension.

HTTP Data — After examining several Python based HTTP proxies, we chose to use Mitmproxy (<https://mitmproxy.org/>) to record all HTTP Request and Response headers. We generate and load a certificate into Firefox to capture HTTPS data alongside HTTP. Additionally, we use the HTTP proxy to dump the content of any Javascript file requested during a page visit.

Javascript files are detected by first checking if the `Content-Type` header contains the string “`javascript`”, and if not, by checking the file extension of the URL path. Once a script is detected, it is decompressed (if necessary) and hashed. The hash and content are sent through a socket to the LevelDBAggregator for de-duplication.

Javascript Access — We provide the researcher with a Javascript interface to pages visited through a Firefox extension. Our extension builds on the work of Fourthparty [32]. In particular, we utilize Fourthparty’s Javascript instrumentation, which defines custom getters and setters on the `window.navigator` and `window.screen`

interfaces⁴. We updated and extended this functionality to record access to the prototypes of the `Storage`, `HTMLCanvasElement`, `CanvasRenderingContext2D`, `RTCPeerConnection`, `AudioContext` objects, as well as the prototypes of several children of `AudioNode`. This records the setting and getting of all object properties and calls of all object methods for any object built from these prototypes. Alongside this, we record the new property values set, and the arguments to all method calls.

In addition to recording access to instrumented objects, we record the URL of the script responsible for the property or method access. To do so, we throw an Error and parse the stack trace after each call or property intercept. This method is successful for 99.9% of Javascript files we encountered, and even works for Javascript files which have been minified or obfuscated with `eval`. A minor limitation is that the function calls of a script which gets passed into the `eval` method of a second script will have their URL labeled as the second script. This method is adapted with minor modifications from the Privacy Badger Firefox Extension⁵.

We also replace Fourthparty’s direct SQLite logging with a socket-based solution logging to our SQLite DataAggregator. This eliminates the need to merge databases post-measurement and makes data management significantly easier for multi-browser measurements.

Workflow. As an example workflow, the researcher issues a command to the task manager and specifies that it should synchronously execute on all browser managers. The task manager checks all of the command execution threads and blocks until all browsers are available to execute a new command. Then it creates new command execution threads for all browsers and sends the command and command parameters over a pipe to the browser manager process. The browser manager interprets this command and runs the necessary Selenium code to execute the command in the browser. If the command is a “Get” command, which causes the browser to visit a new domain, the browser manager distributes the browser id and top-level page being visited to all current instrumentation. The instrumentation uses this information to properly key data for the new page visit. Once the Selenium code returns, the browser manager can send returned data (e.g. the parsed contents of a page) to the SQLite aggregator. All instrumentation is also simultaneously sending data to the respective aggregators from separate threads or processes. Once the command is complete, the browser manager notifies the task manager that it is ready for a new command.

⁴In the latest public version of Fourthparty (May 2015), this instrumentation is not functional due to API changes.

⁵<https://github.com/EFForg/privacybadgerfirefox>

3.2 Evaluation

We now evaluate OpenWPM’s stability, completeness, performance, and generality. In Section 3.3 we look at several studies which have used our platform to conduct multiple large-scale experiments.

Stability. We tested the stability of vanilla Selenium without our infrastructure in a variety of settings. The best average we were able to obtain was roughly 800 pages without a freeze or crash. Even in small-scale studies, the lack of recovery led to loss or corruption of measurement data. Using the isolation provided by our browser manager and task manager, we recover from all browser crashes and have observed no data corruption during stateful measurements of 100,000 sites. During the course of our stateless 1 million site measurement in January 2016 (Section 5), we observe over 90 million requests and nearly 300 million Javascript calls. A single OpenWPM browser can visit around 3500 sites per day, requiring no manual interaction during that time. This represents a significant improvement over the 800 page limit mentioned previously. The scale and speed of the overall measurement depends on the hardware used and the measurement configuration (See “Resource Usage” below).

Completeness. OpenWPM reproduces a human user’s web browsing experience since it uses a full-fledged browser. However, researchers have often used stripped-down browsers such as PhantomJS for studies, trading off fidelity for speed and simplicity.

To test the importance of using a full-fledged browser, we examined the differences between OpenWPM and PhantomJS (version 2.1.1) on the top 100 Alexa sites. We averaged our results over 6 measurements of each site with each tool. Both tools were configured with a time-out of 10 seconds and we excluded a small number of sites that didn’t complete loading.

Unsurprisingly, PhantomJS does not load Flash, HTML5 Video, or HTML5 Audio objects (which it does not support); OpenWPM loads nearly 300 instances of those across all sites. More interestingly, PhantomJS loads about 30% fewer HTML files, and about 50% fewer resources with plain text and stream content types. Upon further examination, one major reason for this is that many sites don’t serve ads to PhantomJS. This makes tracking measurements using PhantomJS problematic.

We also tested PhantomJS with the user-agent string spoofed to look like Firefox, so as to try to prevent sites from treating PhantomJS differently. Here the differences were less extreme, but still present (10% fewer requests of html resources, 15% for plain text, and 30% for stream). However, many sites (such as `dropbox.com`) seem to break when PhantomJS presents the incorrect user-agent string. This is because sites may

Study	Year	Browser automation	Stateful crawls	Persistent profiles	Fine-grained profiles	Advanced profiles	Automated plugin support	Detect tracking login	Monitor tracking cookies	Javascript Instrumentation	Content extraction
Persistent tracking mechanisms [1]	2014	•	•	•	•	•	•	•	•		
FB Connect login permissions [45]	2014	•					•				○
Surveillance implications of web tracking [12]	2015	•	•			•		•			
HSTS and key pinning misconfigurations [20]	2015	•	•			•	○				•
The Web Privacy Census [4]	2015	•	•			•			•		
Geographic Variations in Tracking [15]	2015	•				•					
Analysis of Malicious Web Shells [52]	2016	•									
This study (Sections 5 & 6)	2016	•	•	•	•	•		•	•	•	

Table 1: Seven published studies from five separate research groups which utilize OpenWPM. An unfilled circle indicates that the feature was useful but application-specific programming or manual effort was still required.

expect certain capabilities that PhantomJS does not have or may attempt to access APIs using Firefox-specific names. One site, `weibo.com`, redirected PhantomJS (with either user-agent string) to an entirely different landing page than OpenWPM. In conclusion, these findings support our view that OpenWPM enables significantly more complete and realistic web measurement and tracking measurement than do stripped-down browsers like PhantomJS.

Resource usage. When using the headless configuration, we are able to run up to 10 stateful browser instances on an Amazon EC2 “c4.2xlarge” virtual machine⁶. This virtual machine costs around \$300 per month using price estimates from May 2016. Due to Firefox’s memory consumption, stateful parallel measurements are memory-limited while stateless parallel measurements are typically CPU-limited and can support a higher number of instances. On the same machine we can run 20 browser instances in parallel if the browser state is cleared after each page load.

Generality. Table 1 highlights the generality of the platform, where it is used to study both web privacy and web security questions, ranging from the measurement of the variation of tracking in different countries to the analyzing the deployment of HSTS.

The platform minimizes code duplication both across studies and across configurations of a specific study. For example, the Javascript monitoring instrumentation is about 340 lines of Javascript code. Each additional API monitored takes only a single additional line of code. The instrumentation necessary to measure canvas fingerprinting (Section 6.1) is just three additional lines of code, while the WebRTC measurement (Section 6.3) is just a single line of code.

⁶<https://aws.amazon.com/ec2/instance-types/>

Similarly, the code to add support for new extensions or privacy settings is relatively low: 7 lines of code were required to support Ghostery, 8 lines of code to support HTTPS Everywhere, and 7 lines of codes to control Firefox’s cookie blocking policy.

Even measurements themselves require very little additional code on top of the platform. Each configuration listed in Table 2 requires between 70 and 108 lines of code. By comparison, the core infrastructure code and included instrumentation is over 4000 lines of code, showing that the platform saves a significant amount of engineering effort between studies.

3.3 Applications

Seven academic studies have been published in journals, conferences, and workshops, utilizing OpenWPM to perform a variety of web privacy and security measurements. Table 1 summarizes the advanced features of the platform each research group utilized in their measurements.

In addition to *browser automation* and HTTP data dumps, the platform has several advanced capabilities used by both our own measurements and those in other groups. Measurements can keep state, such as cookies and localStorage, within each session via *stateful measurements*, or persist this state across sessions with *persistent profiles*. Persisting state across measurements has been used to measure cookie respawning [1] and to provide seed profiles for larger measurements (Section 5). In general, stateful measurements are useful to replicate the cookie profile of a real user for tracking [4, 12] and cookie syncing analysis [1] (Section 5.6). In addition to recording state, the platform can *detect tracking cookies*.

The platform also provides programmatic control over individual components of this state such as Flash

cookies through *fine-grained profiles* as well as plug-ins via *advanced plug-in support*. Applications built on top of the platform can *monitor state changes* on disk to record access to Flash cookies and browser state. These features are useful in studies which wish to simulate the experience of users with Flash enabled [4, 15] or examine cookie respawning with Flash [1].

Beyond just monitoring and manipulating state, the platform provides the ability to capture any Javascript API call with the included *Javascript instrumentation*. This is used to measure device fingerprinting (Section 6).

Finally, the platform also has a limited ability to extract content from web pages through the *content extraction* module, and a limited ability to automatically log into websites using the Facebook Connect *automated login* capability. Logging in with Facebook has been used to study login permissions [45].

4 Methodology

We run measurements on the homepages of the top 1 million sites to provide a comprehensive view of web tracking and web privacy. These measurements provide updated metrics on the use of tracking and fingerprinting technologies, allowing us to shine a light onto the practices of third parties and trackers across a large portion of the web. We also explore the effectiveness of consumer privacy tools at giving users control over their online privacy.

Measurement Configuration. We run our measurements on a “c4.2xlarge” Amazon EC2 instance, which currently allocates 8 vCPUs and 15 GiB of memory per machine. With this configuration we are able to run 20 browser instances in parallel. All measurements collect HTTP Requests and Responses, Javascript calls, and Javascript files using the instrumentation detailed in Section 3. Table 2 summarizes the measurement instance configurations. The data used in this paper were collected during January 2016.

All of our measurements use the Alexa top 1 million site list (<http://www.alexa.com>), which ranks sites based on their global popularity with Alexa Toolbar users. Before each measurement, OpenWPM retrieves an updated copy of the list. When a measurement configuration calls for less than 1 million sites, we simply truncate the list as necessary. For each site, the browser will visit the homepage and wait until the site has finished loading or until the 90 second timeout is reached. The browser does not interact with the site or visit any other pages within the site. If there is a timeout we kill the process and restart the browser for the next page visit, as described in Section 3.1.

Stateful measurements. To obtain a complete picture of tracking we must carry out stateful measurements in

addition to stateless ones. Stateful measurements do not clear the browser’s profile between page visits, meaning cookie and other browser storage persist from site to site. For some measurements the difference is not material, but for others, such as cookie syncing (Section 5.6), it is essential.

Making stateful measurements is fundamentally at odds with parallelism. But a serial measurement of 1,000,000 sites (or even 100,000 sites) would take unacceptably long. So we make a compromise: we first build a *seed profile* which visits the top 10,000 sites in a serial fashion, and we save the resulting state.

To scale to a larger measurement, the seed profile is loaded into multiple browser instances running in parallel. With this approach, we can approximately simulate visiting each website serially. For our 100,000 site stateless measurement, we used the “ID Detection 2” browser profile as a seed profile.

This method is not without limitations. For example third parties which don’t appear in the top sites if the seed profile will have different cookies set in each of the parallel instances. If these parties are also involved in cookie syncing, the partners that sync with them (and appear in the seed profile) will each receive multiple IDs for each one of their own. This presents a trade-off between the size the seed profile and the number of third parties missed by the profile. We find that a seed profile which has visited the top 10,000 sites will have communicated with 76% of all third-party domains present on more than 5 of the top 100,000 sites.

Handling errors. In presenting our results we only consider sites that loaded successfully. For example, for the 1 Million site measurement, we present statistics for 917,261 sites. The majority of errors are due to the site failing to return a response, primarily due to DNS lookup failures. Other causes of errors are sites returning a non-2XX HTTP status code on the landing page, such as a 404 (Not Found) or a 500 (Internal Server Error).

Detecting ID cookies. Detecting cookies that store unique user identifiers is a key task that enables many of the results that we report in Section 5. We build on the methods used in previous studies [1, 12]. Browsers store cookies in a structured key-value format, allowing sites to provide both a *name string* and *value string*. Many sites further structure the value string of a single cookie to include a set of named parameters. We parse each cookie value string assuming the format:

$$(name_1=)value_1|…|(name_N=)value_N$$

where $|$ represents any character except a-zA-Z0-9_-=. We determine a (cookie-name, parameter-name, parameter-value) tuple to be an ID cookie if it meets the following criteria: (1) the cookie has an expiration date over 90 days in the future (2) $8 \leq \text{length}(\text{parameter-value}) \leq 100$, (3) the parameter-value

Configuration	# Sites	# Success	Timeout %	Flash Enabled	Stateful	Parallel	HTTP Data	Javascript Files	Javascript Calls	Disk Scans	Time to Crawl
Default Stateless	1 Million	917,261	10.58%		•	•	•	•			14 days
Default Stateful	100,000	94,144	8.23%	○	•	•	•	•			3.5 days
Ghostery	55,000	50,023	5.31%		•	•	•	•			0.7 days
Block TP Cookies	55,000	53,688	12.41%		•	•	•	•			0.8 days
HTTPS Everywhere	55,000	53,705	14.77%		•	•	•	•			1 day
ID Detection 1*	10,000	9,707	6.81%	•	•		•	•	•	•	2.9 days
ID Detection 2*	10,000	9,702	6.73%	•	•		•	•	•	•	2.9 days

Table 2: Census measurement configurations.

An unfilled circle indicates that a seed profile of length 10,000 was loaded into each browser instance in a parallel measurement. “# Success” indicates the number of sites that were reachable and returned a response. A Timeout is a request which fails to completely load in 90 seconds. *Indicates that the measurements were run synchronously on different virtual machines.

remains the same throughout the measurement, (4) the parameter-value is different between machines and has a similarity less than 66% according to the Ratcliff-Obershelp algorithm [7]. For the last step, we run two synchronized measurements (see Table 2) on separate machines and compare the resulting cookies, as in previous studies.

What makes a tracker? Every third party is *potentially* a tracker, but for many of our results we need a more conservative definition. We use popular *tracking-protection lists* for this purpose: EasyList, EasyPrivacy, and a commercial privacy tool’s list. All three lists consist of regular expressions and URL sub-strings which are matched against resource loads to determine if a request should be blocked.

Note that we are not simply classifying domains as trackers or non-trackers, but rather classify each instance of a third party on a particular website as a tracking or non-tracking context. We consider a domain to be in the tracking context if a consumer privacy tool would have blocked that resource. Resource loads which wouldn’t have been blocked by these extensions are considered non-tracking.

While there is agreement between the extensions utilizing these lists, we emphasize that they are far from perfect. They contain false positives and especially false negatives. That is, they miss many trackers — new ones in particular. Indeed, much of the impetus for OpenWPM and our measurements comes from the limitations of manually identifying trackers. Thus, tracking-protection lists should be considered an underestimate of the set of trackers, just as considering all third parties to be trackers is an overestimate.

Finally, for readers interested in further details or in reproducing our work, we provide further methodological details in the Appendix: what constitutes distinct domains (C.1), how to detect the landing page of a

site using the data collected by OpenWPM (C.2), how we detect cookie syncing (C.3), why obfuscation of Javascript doesn’t affect our ability to detect fingerprinting (C.4), and a minor limitation of our method of instrumenting JavaScript calls (C.5).

5 Results of our 1-million site census

5.1 The long but thin tail of online tracking

During our January 2016 measurement of the Top 1 million sites, our tool made over 90 million requests, assembling the largest dataset on web tracking to our knowledge.

Our large scale allows us to answer a rather basic question: how many third parties are there? In short, a lot: the total number of third parties present on at least two first parties is over 81,000.

What is more surprising is that the prevalence of third parties quickly drops off: only 123 of these 81,000 are present on more than 1% of sites. This suggests that the number of third parties that a regular user will encounter on a daily basis is relatively small. The effect is accentuated when we consider that different third parties may be owned by the same entity. All of the top 5 third parties, as well as 12 of the top 20, are Google-owned domains. In fact, *Google, Facebook, and Twitter are the only third-party entities present on more than 10% of sites.*

Further, if we use the definition of tracking based on tracking-protection lists, as defined in Section 4, then trackers are even less prevalent. This is clear from Figure 2, which shows the prevalence of the top third parties (a) in any context and (b) only in tracking contexts. Note the absence or reduction of content-delivery domains such as `gstatic.com`, `fbcdn.net`, and `googleusercontent.com`.

These results might come as a bit of a surprise to

the reader jaded by endless reports of an explosion in third-party tracking. Our data suggest that there is a trend toward economic consolidation in the third-party ecosystem, in line with both some press [30] and some of the academic literature [16]. For the hundred or so third parties that are prevalent on 1% or more of sites, we might expect that they are large enough entities that their behavior can be regulated by public-relations pressure and the possibility of legal or enforcement actions. Indeed, measurement research has repeatedly proved capable of bringing about these outcomes [1, 6, 33].

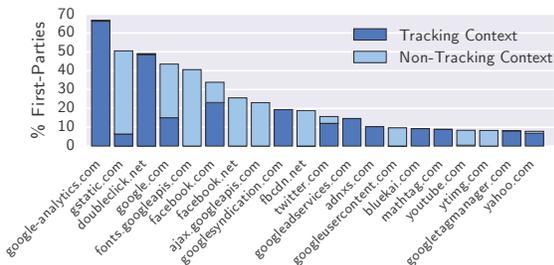


Figure 2: Top third parties on the top 1 million sites. Not all instances of third parties are classified as tracking by our methodology, and in fact the same third party can be classified differently depending on the context. (Section 4).

5.2 Prominence: a metric to rank third parties

In Section 5.1 we ranked third parties by the number of first party sites they appear on. This simple count is a good first approximation, but it has two related drawbacks. A major third party that’s present on (say) 90 of the top 100 sites would have a low score if its prevalence drops off outside the top 100 sites. A related problem is that the rank can be sensitive to the number of websites visited in the measurement. Thus different studies may rank third parties differently.

We also lack a good way to compare third parties (and especially trackers) over time, both individually and in aggregate. Some studies have measured the total number of cookies [4], but we argue that this is a misleading metric, since cookies may not have anything to do with tracking.

To avoid these problems, we propose a principled metric. We start from a model of aggregate browsing behavior. There is some research suggesting that the website traffic follows a power law distribution, with the frequency of visits to the N^{th} ranked website being proportional to $\frac{1}{N}$ [3, 21]. The exact relationship is not important to us; any formula for traffic can be plugged into our prominence metric below.

Definition:

$$\text{Prominence}(t) = \sum_{\text{edge}(s,t)=1} \frac{1}{\text{rank}(s)}$$

where $\text{edge}(s,t)$ indicates whether third party t is present on site s . This simple formula measures the frequency with which an “average” user browsing according to the power-law model will encounter any given third party.

The most important property of prominence is that it de-emphasizes obscure sites, and hence can be adequately approximated by relatively small-scale measurements, as shown in Figure 3. We propose that prominence is the right metric for:

1. Comparing third parties and identifying the top third parties. We present the list of top third parties by prominence in Table 14 (in the Appendix). Prominence ranking produces interesting differences compared to ranking by a simple prevalence count. For example, Content-Distribution Networks become less prominent compared to other types of third parties.
2. Measuring the effect of tracking-protection tools, as we do in Section 5.5.
3. Analyzing the evolution of the tracking ecosystem over time and comparing between studies. The robustness of the *rank-prominence curve* (Figure 3) makes it ideally suited for these purposes.

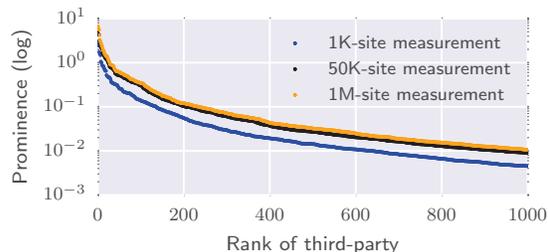


Figure 3: Prominence of third party as a function of prominence rank. We posit that the curve for the 1M-site measurement (which can be adequately approximated by a 50k-site measurement) presents a useful aggregate picture of tracking.

5.3 Third parties impede HTTPS adoption

Table 3 shows the number of first-party sites that support HTTPS and the number that are HTTPS-only. Our results reveal that HTTPS adoption remains rather low despite well-publicized efforts [11]. Publishers have claimed that a major roadblock to adoption is the need to move all embedded third parties and trackers to HTTPS to avoid mixed-content errors [54, 59].

Mixed-content errors occur when HTTP sub-resources are loaded on a secure site. This poses a security problem, leading to browsers to block the resource load or warn the user depending on the content loaded [37]. *Passive* mixed content, that is, non-executable resources loaded over HTTP, cause the browser to display an insecure warning to the user (Figure 4) but still load

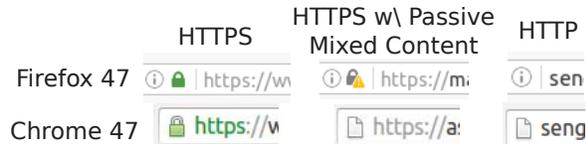


Figure 4: Secure connection UI for Firefox Nightly 47 and Chrome 47. The UI portrays a site with mixed content as less secure than an HTTPS site without mixed content; clicking on the lock icon in Firefox reveals the text “Connection is not secure” when mixed content is present.

	55K Sites	1M Sites
HTTP Only	82.9%	X
HTTPS Only	14.2%	8.6%
HTTPS Opt.	2.9%	X

Table 3: First party HTTPS support on the top 55K and top 1M sites. “HTTP Only” is defined as sites which fail to upgrade when HTTPS Everywhere is enabled. “HTTPS Only” are sites which always redirect to HTTPS. “HTTPS Optional” are sites which provide an option to upgrade, but only do so when HTTPS Everywhere is enabled. We carried out HTTPS-everywhere-enabled measurement for only 55,000 sites, hence the X’s.

the content. *Active* mixed content is a far more serious security vulnerability and is blocked outright by modern browsers; it is not reflected in our measurements.

Third-party support for HTTPS. To test the hypothesis that third parties impede HTTPS adoption, we first characterize the HTTPS support of each third party. If a third party appears on at least 10 sites and is loaded over HTTPS on all of them, we say that it is HTTPS-only. If it is loaded over HTTPS on some but not all of the, we say that it supports HTTPS. If it is loaded over HTTP on all of them, we say that it is HTTP-only. If it appears on less than 10 sites, we do not have enough confidence to make a determination.

Table 4 summarizes the HTTPS support of third party domains which appear on more than 5 sites. A large number of third-party domains are HTTP-only (54%). However, when we weight third parties by prominence, only 5% are HTTP-only. In contrast, 94% of prominence-weighted third parties support both HTTP and HTTPS. This statistic supports our thesis that consolidation of the third-party ecosystem is a plus for security and privacy.

Impact of third-parties. We find that a significant fraction of HTTP-default sites (26%) embed resources from third-parties which do not support HTTPS. These sites would be unable to upgrade to HTTPS without browsers displaying mixed content errors to their users, the majority of which (92%) would contain active content which would be blocked.

Similarly, of the 78,000 first-party sites that are HTTPS-only, 6,000 (7.75%) load with mixed passive content warnings. However, only 11% of these warnings

HTTPS Support	Percent	Prominence weighted %
HTTP Only	54%	5%
HTTPS Only	5%	1%
Both	41%	94%

Table 4: Third party HTTPS support. “HTTP Only” is defined as domains from which resources are only requested over HTTP across all sites on our 1M site measurement. “HTTPS Only” are domains from which resources are only requested over HTTPS. “Both” are domains which have resources requested over both HTTP and HTTPS. Results are limited to third parties embedded on at least 10 first-party sites.

Class	Top 1M % FP	Top 50k % FP
Own	25.4%	29.6%
Favicon	1.4%	2.9%
Tracking	12.2%	27.0%
CDN	1.4%	2.9%
Non-tracking	43.4%	29.6%
Multiple causes	16.2%	8.1%

Table 5: A breakdown of causes of passive mixed-content warnings on the top 1M sites and on the top 50k sites. “Non-tracking” represents third-party content not classified as a tracker or a CDN.

(650) are caused by HTTP-only third parties, suggesting that many domains may be able to mitigate these warnings by ensuring all resources are being loaded over HTTPS when available. We examined the causes of mixed content on these sites, summarized in Table 5. The majority are caused by third parties, rather than the site’s own content, with a surprising 27% caused solely by trackers.

5.4 News sites have the most trackers

The level of tracking on different categories of websites varies considerably — by almost an order of magnitude. To measure variation across categories, we used Alexa’s lists of top 500 sites in each of 16 categories. From each list we sampled 100 sites (the lists contain some URLs that are not home pages, and we excluded those before sampling).

In Figure 5 we show the average number of third parties loaded across 100 of the top sites in each Alexa category. Third parties are classified as trackers if they would have been blocked by one of the tracking protection lists, as discussion in Section 4.

Why is there so much variation? With the exception of the adult category, the sites on the low end of the spectrum are mostly sites which belong to government organizations, universities, and non-profit entities. This suggests that websites may be able to forgo advertising and tracking due to the presence of funding sources

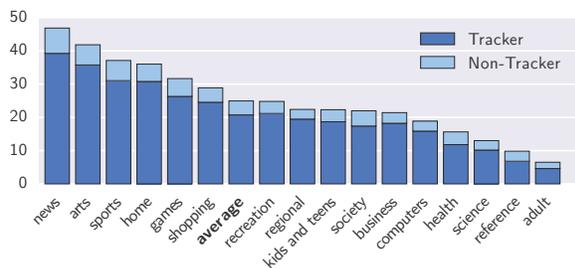


Figure 5: Average number of third parties in each Alexa category.

external to the web. Sites on the high end of the spectrum are largely those which provide editorial content. Since many of these sites provide articles for free, and lack an external funding source, they are pressured to monetize page views with significantly more advertising.

5.5 Does tracking protection work?

Users have two main ways to reduce their exposure to tracking: the browser’s built in privacy features and extensions such as Ghostery or uBlock Origin.

Contrary to previous work questioning the effectiveness of Firefox’s third-party cookie blocking [12], we do find the feature to be effective. Specifically, only 237 sites (0.4%) have any third-party cookies set during our measurement set to block all third-party cookies (“Block TP Cookies” in Table 2). Most of these are for benign reasons, such as redirecting to the U.S. version of a non-U.S. site. We did find exceptions, including 32 that contained ID cookies. For example, there are six Australian news sites that first redirect to `news.com.au` before re-directing back to the initial domain, which seems to be for tracking purposes. While this type of workaround to third-party cookie blocking is not rampant, we suggest that browser vendors should closely monitor it and make changes to the blocking heuristic if necessary.

Another interesting finding is that when third-party cookie blocking was enabled, the average number of third parties per site dropped from 17.7 to 12.6. Our working hypothesis for this drop is that deprived of ID cookies, third parties curtail certain tracking-related requests such as cookie syncing (which we examine in Section 5.6).

We also tested Ghostery, and found that it is effective at reducing the number of third parties and ID cookies (Figure 6). The average number of third-party includes went down from 17.7 to 3.3, of which just 0.3 had third-party cookies (0.1 with IDs). We examined the prominent third parties that are not blocked and found almost all of these to be content-delivery networks like `cloudflare.com` or widgets like `maps.google.com`, which Ghostery does not try to block. So Ghostery

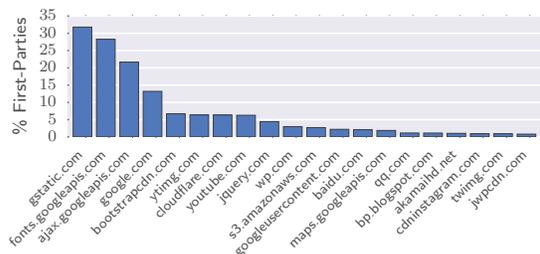


Figure 6: Third-party trackers on the top 55k sites with Ghostery enabled. The majority of the top third-party domains not blocked are CDNs or provide embedded content (such as Google Maps).

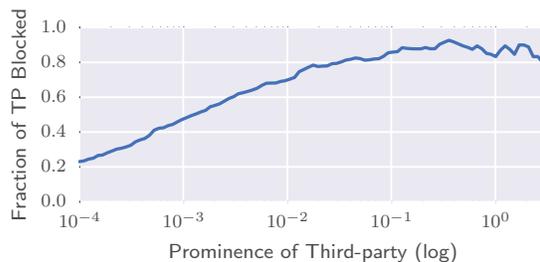


Figure 7: Fraction of third parties blocked by Ghostery as a function of the prominence of the third party. As defined earlier, a third party’s prominence is the sum of the inverse ranks of the sites it appears on.

works well at achieving its stated objectives.

However, the tool is less effective for obscure trackers (prominence < 0.1). We see a similar trend in the detection of fingerprinting scripts (Section 6.6); less prominent scripts are not blocked as frequently by blocking tools. This makes sense given that the block list is manually compiled and the developers are less likely to have encountered obscure trackers. It suggests that large-scale measurement techniques like ours will be useful for tool developers to minimize gaps in their coverage.

5.6 How common is cookie syncing?

Cookie syncing, a workaround to the Same-Origin Policy, allows different trackers to share user identifiers with each other. Besides being hard to detect, cookie syncing enables back-end server-to-server data merges hidden from public view, which makes it a privacy concern.

Our ID cookie detection methodology (Section 4) allows us to detect instances of cookie syncing. If tracker A wants to share its ID for a user with tracker B, it can do so in one of two ways: embedding the ID in the request URL to tracker B, or in the referer URL. We therefore look for instances of IDs in referer, request, and response URLs, accounting for URL encoding and other subtleties. We describe the full details of our methodology in Appendix C.3, with an important caveat

that our methodology captures both intentional and accidental ID sharing.

Most third parties are involved in cookie syncing. We run our analysis on the top 100,000 site stateful measurement. The most prolific cookie-syncing third party is `doubleclick.net` — it shares 108 different cookies with 118 other third parties (this includes both events where it is a referer and where it is a receiver).

More interestingly, we find that the vast majority of top third parties sync cookies with at least one other party: 45 of the top 50, 85 of the top 100, 157 of the top 200, and 460 of the top 1,000. This adds further evidence that cookie syncing is an underappreciated and under-researched privacy concern.

We also find that third parties are highly connected by synced cookies. Specifically, of the top 50 third parties that are involved in cookie syncing, the probability that a random pair will have at least one cookie in common is 85%. The corresponding probability for the top 100 is 66%.

Implications of “promiscuous cookies” for surveillance. From the Snowden leaks, we learnt that that NSA “piggybacks” on advertising cookies for surveillance and exploitation of targets [17, 51, 53]. How effective can this technique be? We present one answer to this question. We consider a threat model where a surveillance agency has identified a target by a third-party cookie (for example, via leakage of identifiers by first parties, as described in [12, 22, 24]). The adversary uses this identifier to coerce or compromise a third party into enabling surveillance or targeted exploitation.

We find that some cookies get synced over and over again to dozens of third parties; we call these *promiscuous cookies*. It is not yet clear to us why these cookies are synced repeatedly and shared widely. This means that if the adversary has identified a user by such a cookie, their ability to surveil or target malware to that user will be especially good. The most promiscuous cookie that we found belongs to the domain `adverticum.net`; it is synced or leaked to 82 other parties which are collectively present on 752 of the top 1,000 websites! In fact, each of the top 10 most promiscuous cookies is shared with enough third parties to cover 60% or more of the top 1,000 sites.

6 Fingerprinting: a 1-million site view

OpenWPM significantly reduces the engineering requirement of measuring device fingerprinting, making it easy to update old measurements and discover new techniques. In this section, we demonstrate this through several new fingerprinting measurements, two of which have never been measured at scale before, to the best of our knowledge. We show how the number of sites on

which font fingerprinting is used and the number of third parties using canvas fingerprinting have both increased by considerably in the past few years. We also show how WebRTC’s ability to discover local IPs without user permission or interaction is used almost exclusively to track users. We analyze a new fingerprinting technique utilizing `AudioContext`⁷ found during our investigations. Finally, we discuss the use of the Battery API by two fingerprinting scripts.

Our fingerprinting measurement methodology utilizes data collected by the Javascript instrumentation described in Section 3.1. With this instrumentation, we monitor access to all built-in interfaces and objects we suspect may be used for fingerprinting. By monitoring on the interface or object level, we are able to record access to all method calls and property accesses for each interface we thought might be useful for fingerprinting. This allows us to build a detection criterion for each fingerprinting technique after a detailed analysis of example scripts.

Although our detection criteria currently have negligible low false positive rate, we recognize that this may change as new web technologies and applications emerge. However, instrumenting all properties and methods of an API provides a complete picture of each application’s use of the interface, allowing our criteria to also be updated. More importantly, this allows us to replace our detection criteria with machine learning, which is an area of ongoing work (Section 7).

Rank Interval	% of First-parties		
	Canvas	Canvas Font	WebRTC
[0,1K)	5.10%	2.50%	0.60%
[1K,10K)	3.91%	1.98%	0.42%
[10K,100K)	2.45%	0.86%	0.19%
[100K,1M)	1.31%	0.25%	0.06%

Table 6: Prevalence of fingerprinting scripts on different slices of the top sites. More popular sites are more likely to have fingerprinting scripts.

6.1 Canvas Fingerprinting

Privacy threat. The HTML Canvas allows web application to draw graphics in real time, with functions to support drawing shapes, arcs, and text to a custom canvas element. In 2012 Mowery and Schacham demonstrated how the HTML Canvas could be used to fingerprint devices [36]. Differences in font rendering, smoothing, anti-aliasing, as well as other device features cause devices to draw the image differently. This allows the resulting pixels to be used as part of a device fingerprint.

⁷<https://developer.mozilla.org/en-US/docs/Web/API/AudioContext>

Detection methodology. We build on a 2014 measurement study by Acar et.al. [1]. Since that study, the canvas API has received broader adoption for non-fingerprinting purposes, so we make several changes to reduce false positives. In our measurements we record access to nearly all of properties and methods of the `HTMLCanvasElement`⁸ interface and of the `CanvasRenderingContext2D`⁹ interface. We filter scripts according to the following criteria:

1. The canvas element’s height and width properties must not be set below 16 px.¹⁰
2. Text must be written to canvas with least two colors or at least 10 distinct characters.
3. The script should not call the `save`, `restore`, or `addEventListener` methods of the rendering context.
4. The script extracts an image with `toDataURL` or with a single call to `getImageData` that specifies an area with a minimum size of $16\text{px} \times 16\text{px}$.

This heuristic is designed to filter out scripts which are unlikely to have sufficient complexity or size to act as an identifier. We manually verified the accuracy of our detection methodology by inspecting the images drawn and the source code. We found a mere 4 false positives out of 3493 scripts identified on a 1 million site measurement. Each of the 4 is only present on a single first-party.

Results. We found canvas fingerprinting on 14,371 (1.6%) sites. The vast majority (98.2%) are from third-party scripts. These scripts come from about 3,500 URLs hosted on about 400 domains. Table 7 shows the top 5 domains which serve canvas fingerprinting scripts ordered by the number of first-parties they are present on.

Domain	# First-parties
doubleverify.com	7806
lijit.com	2858
alicdn.com	904
audienceinsights.net	499
boo-box.com	303
407 others	2719
TOTAL	15089 (14371 unique)

Table 7: Canvas fingerprinting on the Alexa Top 1 Million sites. For a more complete list of scripts, see Table 11 in the Appendix.

Comparing our results with a 2014 study [1], we find three important trends. First, the most prominent trackers have by-and-large stopped using it, suggesting that the public backlash following that study was effective. Second, the overall number of domains employing it

⁸<https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement>

⁹<https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D>

¹⁰The default canvas size is $300\text{px} \times 150\text{px}$.

has increased considerably, indicating that knowledge of the technique has spread and that more obscure trackers are less concerned about public perception. As the technique evolves, the images used have increased in variety and complexity, as we detail in Figure 11 in the Appendix. Third, the use has shifted from behavioral tracking to fraud detection, in line with the ad industry’s self-regulatory norm regarding acceptable uses of fingerprinting.

6.2 Canvas Font Fingerprinting

Privacy threat. The browser’s font list is very useful for device fingerprinting [10]. The ability to recover the list of fonts through Javascript or Flash is known, and existing tools aim to protect the user against scripts that do that [2, 39]. But can fonts be enumerated using the Canvas interface? The only public discussion of the technique seems to be a Tor Browser Bundle ticket from 2014¹¹. To the best of our knowledge, we are the first to measure its usage in the wild.

Detection methodology. The `CanvasRenderingContext2D` interface provides a `measureText` method, which returns several metrics pertaining to the text size (including its width) when rendered with the current font settings of the rendering context. Our criterion for detecting canvas font fingerprinting is: the script sets the font property to at least 50 distinct, valid values and also calls the `measureText` method at least 50 times on the same text string. We manually examined the source code of each script found this way and verified that there are zero false positives on our 1 million site measurement.

Results. We found canvas-based font fingerprinting present on 3,250 first-party sites. This represents less than 1% of sites, but as Table 6 shows, the technique is more heavily used on the top sites, reaching 2.5% of the top 1000. The vast majority of cases (90%) are served by a single third party, `mathtag.com`. The number of sites with font fingerprinting represents a seven-fold increase over a 2013 study [2], although they did not consider Canvas.

6.3 WebRTC-based fingerprinting

Privacy threat. WebRTC is a framework for peer-to-peer Real Time Communication in the browser, and accessible via Javascript. To discover the best network path between peers, each peer collects all available candidate addresses, including addresses from the local network interfaces (such as ethernet or WiFi) and addresses from the public side of the NAT and makes them available to

¹¹<https://trac.torproject.org/projects/tor/ticket/13400>

the web application *without explicit permission from the user*. This has led to serious privacy concerns: users behind a proxy or VPN can have their ISP’s public IP address exposed [55]. We focus on a slightly different privacy concern: users behind a NAT can have their local IP address revealed, which can be used as an identifier for tracking. A detailed description of the discovery process is given in Appendix B.

Detection methodology. To detect WebRTC local IP discovery, we instrument the `RTCPeerConnection`¹² interface prototype and record access to its method calls and property access. After the measurement is complete, we select the scripts which call the `createDataChannel` and `createOffer` APIs, and access the event handler `onIceCandidate`¹³. We manually verified that scripts that call these functions are in fact retrieving candidate IP addresses, with zero false positives on 1 million sites. Next, we manually tested if such scripts are using these IPs for tracking. Specifically, we check if the code is located in a script that contains other known fingerprinting techniques, in which case we label it tracking. On the other hand, if we manually assess that the code has a clear non-tracking use, we label it non-tracking. If neither of these is the case, we label the script as ‘unknown’. We emphasize that even the non-tracking scripts present the abovementioned privacy concern related to leakage of private IPs.

Results. We found WebRTC being used to discover local IP addresses without user interaction on 715 sites out of the top 1 million. The vast majority of these (659) were done by third-party scripts, loaded from 99 different locations. A large majority (625) were used for tracking. The top 10 scripts accounted for 83% of usage, in line with our other observations about the small number of third parties responsible for most tracking. We provide a list of scripts in Appendix Table 13.

The number of confirmed non-tracking uses of unsolicited IP candidate discovery is small, and based on our analysis, none of them is critical to the application. We therefore suggest that WebRTC IP discovery should be private by default, in contrast to the recommendation of a Working Group that recently reviewed the security and privacy concerns [55].

6.4 AudioContext Fingerprinting

The scale of our data gives us a new way to systematically identify new types of fingerprinting not previously reported in the literature. The key insight is that fingerprinting techniques typically aren’t used in isolation but

¹²<https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection>

¹³Although we found it unnecessary for current scripts, instrumenting `localDescription` will cover all possible IP address retrievals.

Classification	# Scripts	# First-parties
Tracking	57	625 (88.7%)
Non-Tracking	10	40 (5.7%)
Unknown	32	40 (5.7%)

Table 8: Summary of WebRTC local IP discovery on the top 1 million Alexa sites.

rather in conjunction with each other. So we monitor known tracking scripts and look for unusual behavior (e.g., use of new APIs) in a semi-automated fashion.

Using this approach we found several fingerprinting scripts utilizing `AudioContext` and related interfaces. A manual analysis of these scripts suggest that trackers are attempting to utilize the `Audio API` to fingerprint users in multiple ways. Their use ranges from simply checking for the API’s existence to deriving a fingerprint from the underlying audio processing. We provide a live demonstration page to compute and visualize a device’s audio fingerprint at: <https://webtap.princeton.edu/audio-fp>.

In the simplest case, a script from the company *Liverail*¹⁴ checks for the existence of an `AudioContext` and `OscillatorNode` to add a single bit of information to a broader fingerprint. More sophisticated scripts process an audio signal generated with an `OscillatorNode` to fingerprint the device. This technique appears conceptually similar to that of canvas fingerprinting. Audio signals processed on different machines or browsers may have slight differences due to hardware or software differences between the machines, while the same combination of machine and browser will produce the same output.

Figure 8 shows two alternate audio fingerprinting configurations found in three scripts. Both configurations process an audio signal from an `OscillatorNode`, before reading the resulting signal and hashing it to create a device audio fingerprint. Full details of the configurations are given in Appendix D.

We tested the output of the scripts on a small sample of machines, and confirmed the values returned are largely stable on the same machine and different for different machines. We did observe a couple examples of instability for the top technique of Figure 8, and several examples of collisions for machines with similar hardware for the bottom technique of Figure 8. We leave a full evaluation of the effectiveness of the technique to future work. See Figure 9 for a visualization of the tail end of the processed FFT from several browsers.

Using a follow-up measurement of the Alexa top 1 million sites from March 2016, we find that this technique is very infrequently used. The *Liverail* scripts

¹⁴<https://www.liverail.com/>

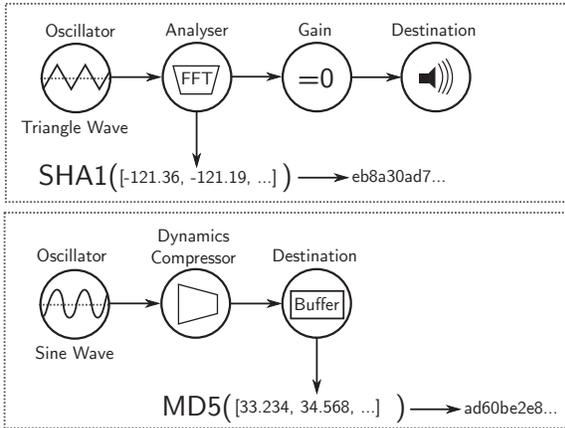


Figure 8: AudioContext node configuration used to generate a fingerprint. **Top:** Used by `www.cdn-net.com/cc.js` in an AudioContext. **Bottom:** Used by `client.a.pxi.pub/*/main.min.js` and `js.ad-score.com/score.min.js` in an OfflineAudioContext. Full details in Appendix D.

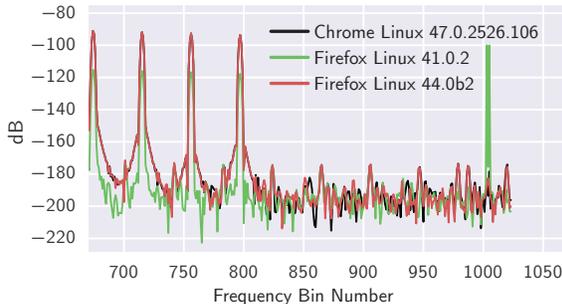


Figure 9: Visualization of processed `OscillatorNode` output from the fingerprinting script `https://www.cdn-net.com/cc.js` for three different browsers on the same machine. We found these values to remain constant for each browser after several checks.

are present on just 512 sites, and the remaining scripts are present on even less. The `cdn-net.com` scripts are included on 49 sites, however the audio fingerprinting section appears to be disabled. The `pxi.pub` and `ad-score.com` scripts are present and actively fingerprinting users on 12 and 6 sites respectively. This shows that, even with very low usage rates, we can successfully bootstrap off of currently known fingerprinting scripts to discover and measure new techniques.

6.5 Battery API Fingerprinting

As a second example of bootstrapping, we analyze the Battery Status API¹⁵, which allows a site to query the browser for the current battery level or charging status of a host device. It was previously identified as

¹⁵<https://www.w3.org/TR/2016/PR-battery-status-20160329/>

a potential fingerprinting vector [41]. We discovered two fingerprinting scripts utilizing the API during our manual analysis of other fingerprinting techniques.

One script, `https://go.lynxbroker.de/eat_heartbeat.js`, retrieves the current charge level of the host device and combines it with several other identifying features. These features include the canvas fingerprint and the user’s local IP address retrieved with WebRTC as described in Section 6.1 and Section 6.3. The second script, `http://js.ad-score.com/score.min.js`, queries all properties of the BatteryManager interface, retrieving the current charging status, the charge level, and the time remaining to discharge or recharge. As with the previous script, these features are combined with other identifying features used to fingerprint a device.

6.6 The wild west of fingerprinting scripts

In Section 5.5 we found the various tracking protection measures to be very effective at reducing third-party tracking. In Table 9 we show how blocking tools miss many of the scripts we detected throughout Section 6, particularly those using lesser-known techniques. Although blocking tools detect the majority of instances of well-known techniques, only a fraction of the total number of scripts are detected.

Technique	Ghostery		EasyList + EasyPrivacy	
	% Scripts	% Sites	% Scripts	% Sites
Canvas	8.4%	80.5%	25.1%	88.3%
Canvas Font	10.3%	90.6%	10.3%	90.6%
WebRTC	1.9%	1.0%	4.8%	5.6%
Audio	11.1%	53.1%	5.6%	1.6%

Table 9: Percentage of fingerprinting scripts blocked by Ghostery or the combination of EasyList and EasyPrivacy for all techniques described in Section 6. Included is the percentage of sites with fingerprinting scripts on which scripts are blocked.

Fingerprinting scripts pose a unique challenge for manually curated block lists. They don’t typically change the rendering of a page and may not be included by an advertising entity. The script content may be obfuscated to the point where manual inspection is difficult and the purpose of the script unclear.

OpenWPM’s active instrumentation (see Section 3.1) detects a large number of scripts not blocked by the current privacy tools. Ghostery and a combination of EasyList and EasyPrivacy both perform similarly in their block rate. The privacy tools block canvas fingerprinting on over 80% of sites, and block canvas font fingerprinting on over 90%. However, only a fraction of the total number of scripts utilizing the techniques are blocked (between 8% and 25%) showing that less popular third

parties are missed. Lesser-known techniques, like WebRTC IP discovery and Audio fingerprinting have even lower rates of detection.

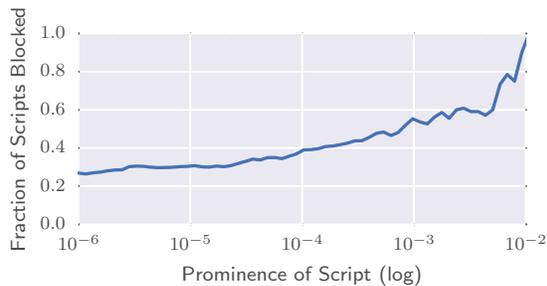


Figure 10: Fraction of fingerprinting scripts with prominence above a given level blocked by Ghostery, EasyList, or EasyPrivacy.

In fact, fingerprinting scripts with a low prominence are blocked much less frequently than those with high prominence. Figure 10 shows the fraction of scripts which are blocked by Ghostery, EasyList, or Easyprivacy for all techniques analyzed in this section. All scripts with a prominence above 0.0125 are detected and blocked by one of the blocking tools, while only 38% of those with a prominence above 0.0001 are. The long tail of fingerprinting scripts are largely unblocked by current privacy tools.

7 Conclusion and future work

Web privacy measurement has the potential to play a key role in keeping online privacy incursions and power imbalances in check. To achieve this potential, measurement tools must be made available broadly rather than just within the research community. In this work, we’ve tried to bring this ambitious goal closer to reality.

The analysis presented in this paper represents a snapshot of results from ongoing, monthly measurements. OpenWPM and census measurements are the first two stages of a multi-year project. We are currently working on two directions that build on the work presented here. The first is the use of machine learning to automatically detect and classify trackers. If successful, this will greatly improve the effectiveness of browser privacy tools. Today such tools use tracking-protection lists that need to be created manually and laboriously, and suffer from significant false positives as well as false negatives. Our large-scale data provide the ideal source of ground truth for training classifiers to detect and categorize trackers.

The second line of work is a web-based analysis platform that makes it easy for a minimally technically skilled analyst to investigate online tracking based on the data we make available. In particular, we are aiming

to make it possible for an analyst to save their analysis scripts and results to the server, share it, and for others to build on it.

8 Acknowledgements

We would like to thank Shivam Agarwal for contributing analysis code used in this study, Christian Eubank and Peter Zimmerman for their work on early versions of OpenWPM, and Gunes Acar for his contributions to OpenWPM and helpful discussions during our investigations, and Dillon Reisman for his technical contributions.

We’re grateful to numerous researchers for useful feedback: Joseph Bonneau, Edward Felten, Steven Goldfeder, Harry Kalodner, and Matthew Salganik at Princeton, Fernando Diaz and many others at Microsoft Research, Franziska Roesner at UW, Marc Juarez at KU Leuven, Nikolaos Laoutaris at Telefonica Research, Vincent Toubiana at CNIL, France, Lukasz Olejnik at INRIA, France, Tanvi Vyas at Mozilla, Chameleon developer Alexei Miagkov, Joel Reidenberg at Fordham, Andrea Matwyshyn at Northeastern, and the participants of the Princeton Web Privacy and Transparency workshop.

This work was supported by NSF Grant CNS 1526353, a grant from the Data Transparency Lab, and by Amazon AWS Cloud Credits for Research.

References

- [1] ACAR, G., EUBANK, C., ENGLEHARDT, S., JUAREZ, M., NARAYANAN, A., AND DIAZ, C. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS 2014)* (2014).
- [2] ACAR, G., JUAREZ, M., NIKIFORAKIS, N., DIAZ, C., GÜRSER, S., PIESSENS, F., AND PRENEEL, B. FPDetective: dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM.
- [3] ADAMIC, L. A., AND HUBERMAN, B. A. Zipfs law and the internet. *Glottometrics* 3, 1 (2002), 143–150.
- [4] ALTAWEEL I, GOOD N, H. C. Web privacy census. *Technology Science* (2015).
- [5] ANGWIN, J. What they know. The Wall Street Journal. <http://online.wsj.com/public/page/what-they-know-digital-privacy.html>, 2012.
- [6] AYENSON, M., WAMBACH, D. J., SOLTANI, A., GOOD, N., AND HOOFNAGLE, C. J. Flash cookies and privacy II: Now with HTML5 and ETag respawning. *World Wide Web Internet And Web Information Systems* (2011).
- [7] BLACK, P. E. Ratcliff/Obershelp pattern recognition. <http://xlinux.nist.gov/dads/HTML/ratcliff0bershelp.html>, December 2004.

- [8] DATTA, A., TSCHANTZ, M. C., AND DATTA, A. Automated experiments on ad privacy settings. *Proceedings on Privacy Enhancing Technologies 2015*, 1 (2015), 92–112.
- [9] DAVIS, W. KISSmetrics Finalizes Supercookies Settlement. <http://www.mediapost.com/publications/article/191409/kissmetrics-finalizes-supercookies-settlement.html>, 2013. [Online; accessed 12-May-2014].
- [10] ECKERSLEY, P. How unique is your web browser? In *Privacy Enhancing Technologies* (2010), Springer.
- [11] ELECTRONIC FRONTIER FOUNDATION. Encrypting the Web. <https://www.eff.org/encrypt-the-web>.
- [12] ENGLEHARDT, S., REISMAN, D., EUBANK, C., ZIMMERMAN, P., MAYER, J., NARAYANAN, A., AND FELTEN, E. W. Cookies that give you away: The surveillance implications of web tracking. In *Proceedings of the 24th International Conference on World Wide Web* (2015), International World Wide Web Conferences Steering Committee, pp. 289–299.
- [13] FEDERAL TRADE COMMISSION. Google will pay \$22.5 million to settle FTC charges it misrepresented privacy assurances to users of Apple’s Safari internet browser. <https://www.ftc.gov/news-events/press-releases/2012/08/google-will-pay-225-million-settle-ftc-charges-it-misrepresented>, 2012.
- [14] FIFIELD, D., AND EGELMAN, S. Fingerprinting web users through font metrics. In *Financial Cryptography and Data Security*. Springer, 2015, pp. 107–124.
- [15] FRUCHTER, N., MIAO, H., STEVENSON, S., AND BALEBAKO, R. Variations in tracking in relation to geographic location. In *Proceedings of W2SP* (2015).
- [16] GILL, P., ERRAMILI, V., CHAINTREAU, A., KRISHNAMURTHY, B., PAPAGIANNAKI, K., AND RODRIGUEZ, P. Follow the money: understanding economics of online aggregation and advertising. In *Proceedings of the 2013 conference on Internet measurement conference* (2013), ACM, pp. 141–148.
- [17] GORMAN, S., AND VALENTINO-DEVRIES, J. New Details Show Broader NSA Surveillance Reach. <http://online.wsj.com/news/articles/SB10001424127887324108204579022874091732470>, 2013.
- [18] HANNAK, A., SOELLER, G., LAZER, D., MISLOVE, A., AND WILSON, C. Measuring price discrimination and steering on e-commerce web sites. In *Proceedings of the 14th Internet Measurement Conference (IMC 2014)* (2014).
- [19] HOOFNAGLE, C. J., AND GOOD, N. Web privacy census. *Available at SSRN 2460547* (2012).
- [20] KRANCH, M., AND BONNEAU, J. Upgrading HTTPS in midair: HSTS and key pinning in practice. In *NDSS ’15: The 2015 Network and Distributed System Security Symposium* (February 2015).
- [21] KRASHAKOV, S. A., TESLYUK, A. B., AND SHCHUR, L. N. On the universality of rank distributions of website popularity. *Computer Networks* 50, 11 (2006), 1769–1780.
- [22] KRISHNAMURTHY, B., NARYSHKIN, K., AND WILLS, C. Privacy leakage vs. protection measures: the growing disconnect. In *Proceedings of the Web* (2011), vol. 2.
- [23] KRISHNAMURTHY, B., AND WILLS, C. Privacy diffusion on the web: a longitudinal perspective. In *Proceedings of the 18th international conference on World wide web* (2009), ACM.
- [24] KRISHNAMURTHY, B., AND WILLS, C. E. On the leakage of personally identifiable information via online social networks. In *Proceedings of the 2nd ACM workshop on Online social networks* (2009), ACM.
- [25] LAPERDRIX, P., RUDAMETKIN, W., AND BAUDRY, B. Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In *37th IEEE Symposium on Security and Privacy (S&P 2016)* (2016).
- [26] LÉCUYER, M., DUCCOFFE, G., LAN, F., PAPANCEA, A., PETSIOS, T., SPAHN, R., CHAINTREAU, A., AND GEAMBASU, R. Xray: Enhancing the webs transparency with differential correlation. In *USENIX Security Symposium* (2014).
- [27] LECUYER, M., SPAHN, R., SPILIOPOLOUS, Y., CHAINTREAU, A., GEAMBASU, R., AND HSU, D. Sunlight: Fine-grained targeting detection at scale with statistical confidence. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), ACM, pp. 554–566.
- [28] LEYDEN, J. Sites pulling sneaky flash cookie-snoop. http://www.theregister.co.uk/2009/08/19/flash_cookies/, 2009.
- [29] LIBERT, T. Exposing the invisible web: An analysis of third-party http requests on 1 million websites. *International Journal of Communication* 9, 0 (2015).
- [30] MARSHALL, J. Burst of M&A in Online Advertising as Shakeout Begins. <http://blogs.wsj.com/cmo/2015/01/07/burst-of-ma-in-online-advertising-as-shakeout-begins/>, 2015.
- [31] MATTIOLI, D. On Orbitz, Mac users steered to pricier hotels. <http://online.wsj.com/news/articles/SB10001424052702304458604577488822667325882>, 2012.
- [32] MAYER, J. R., AND MITCHELL, J. C. Third-party web tracking: Policy and technology. In *Security and Privacy (SP), 2012 IEEE Symposium on* (2012), IEEE.
- [33] McDONALD, A. M., AND CRANOR, L. F. Survey of the use of Adobe Flash Local Shared Objects to respawn HTTP cookies, a. *ISJLP* 7 (2011).
- [34] MIKIANS, J., GYARMATI, L., ERRAMILI, V., AND LAOUTARIS, N. Detecting price and search discrimination on the internet. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks* (2012), ACM.
- [35] MOHAMED, N. You deleted your cookies? think again. <http://www.wired.com/2009/08/you-deleted-your-cookies-think-again/>, 2009.
- [36] MOWERY, K., AND SHACHAM, H. Pixel perfect: Fingerprinting canvas in html5. *Proceedings of W2SP* (2012).
- [37] MOZILLA DEVELOPER NETWORK. Mixed content - Security. https://developer.mozilla.org/en-US/docs/Security/Mixed_content.
- [38] NIKIFORAKIS, N., INVERNIZZI, L., KAPRAVELOS, A., VAN ACKER, S., JOOSEN, W., KRUEGEL, C., PIESSENS, F., AND VIGNA, G. You are what you include: Large-scale evaluation of remote javascript inclusions. In *Proceedings of the 2012 ACM conference on Computer and communications security* (2012), ACM, pp. 736–747.

- [39] NIKIFORAKIS, N., KAPRAVELOS, A., JOOSEN, W., KRUEGEL, C., PIESSENS, F., AND VIGNA, G. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Security and Privacy (SP), 2013 IEEE Symposium on* (2013), IEEE.
- [40] OCARIZA, F., PATTABIRAMAN, K., AND ZORN, B. Javascript errors in the wild: An empirical study. In *Software Reliability Engineering (ISSRE), 2011 IEEE 22nd International Symposium on* (2011), IEEE, pp. 100–109.
- [41] OLEJNIK, L., ACAR, G., CASTELLUCCIA, C., AND DIAZ, C. The leaking battery. *Cryptology ePrint Archive Report 2015/616* (2015).
- [42] OLEJNIK, L., CASTELLUCCIA, C., ET AL. Selling off privacy at auction. In *NDSS '14: The 2014 Network and Distributed System Security Symposium* (2014).
- [43] PHANTOM JS. Supported web standards. <http://www.webcitation.org/6hI3iptm5>, 2016.
- [44] RAFIQUE, M. Z., VAN GOETHEM, T., JOOSEN, W., HUYGENS, C., AND NIKIFORAKIS, N. Its free for a reason: Exploring the ecosystem of free live streaming services. In *NDSS '16: The 2016 Network and Distributed System Security Symposium* (2016).
- [45] ROBINSON, N., AND BONNEAU, J. Cognitive disconnect: Understanding Facebook Connect login permissions. In *Proceedings of the second edition of the ACM conference on Online social networks* (2014), ACM, pp. 247–258.
- [46] ROESNER, F., KOHNO, T., AND WETHERALL, D. Detecting and defending against third-party tracking on the web. In *9th USENIX Symposium on Networked Systems Design and Implementation* (2012).
- [47] ROESNER, F., KOHNO, T., AND WETHERALL, D. Detecting and Defending Against Third-Party Tracking on the Web. In *Symposium on Networking Systems Design and Implementation* (2012), USENIX.
- [48] SELENIUM BROWSER AUTOMATION. Selenium faq. <https://code.google.com/p/selenium/wiki/FrequentlyAskedQuestions>, 2014.
- [49] SINGEL, R. Online Tracking Firm Settles Suit Over Undeletable Cookies. <http://www.wired.com/2010/12/zombie-cookie-settlement/>, 2010.
- [50] SOLTANI, A., CANTY, S., MAYO, Q., THOMAS, L., AND HOOFNAGLE, C. J. Flash cookies and privacy. In *AAAI Spring Symposium: Intelligent Information Privacy Management* (2010).
- [51] SOLTANI, A., PETERSON, A., AND GELLMAN, B. NSA uses Google cookies to pinpoint targets for hacking. <http://www.washingtonpost.com/blogs/the-switch/wp/2013/12/10/nsa-uses-google-cookies-to-pinpoint-targets-for-hacking>, December 2013.
- [52] STAROV, O., DAHSE, J., AHMAD, S. S., HOLZ, T., AND NIKIFORAKIS, N. No honor among thieves: A large-scale analysis of malicious web shells. In *Proceedings of the 25th International Conference on World Wide Web* (2016).
- [53] THE GUARDIAN. ‘Tor Stinks’ presentation - read the full document. <http://www.theguardian.com/world/interactive/2013/oct/04/tor-stinks-nsa-presentation-document>, October 2013.
- [54] TOLLMAN, Z. Were Going HTTPS: Heres How WIRED Is Tackling a Huge Security Upgrade. <https://www.wired.com/2016/04/wired-launching-https-security-upgrade/>, 2016.
- [55] UBERTI, J., AND WEI SHIEH, G. WebRTC IP Address Handling Recommendations. <https://datatracker.ietf.org/doc/draft-ietf-rtcweb-ip-handling/>.
- [56] VAN ACKER, S., NIKIFORAKIS, N., DESMET, L., JOOSEN, W., AND PIESSENS, F. Flashover: Automated discovery of cross-site scripting vulnerabilities in rich internet applications. In *Proceedings of the 7th ACM Symposium on Information, Computer and Communications Security* (2012), ACM, pp. 12–13.
- [57] VAN GOETHEM, T., PIESSENS, F., JOOSEN, W., AND NIKIFORAKIS, N. Clubbing seals: Exploring the ecosystem of third-party security seals. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014), ACM, pp. 918–929.
- [58] VISSERS, T., NIKIFORAKIS, N., BIELOVA, N., AND JOOSEN, W. Crying wolf? on the price discrimination of online airline tickets. HotPETS, 2014.
- [59] WAZER, W. V. Moving the Washington Post to HTTPS. <https://developer.washingtonpost.com/pb/blog/post/2015/12/10/moving-the-washington-post-to-https/>, 2015.
- [60] XING, X., MENG, W., DOOZAN, D., FEAMSTER, N., LEE, W., AND SNOEREN, A. C. Exposing inconsistent web search results with bobble. In *Passive and Active Measurement* (2014), Springer, pp. 131–140.
- [61] YUE, C., AND WANG, H. A measurement study of insecure javascript practices on the web. *ACM Transactions on the Web (TWEB)* 7, 2 (2013), 7.
- [62] ZARRAS, A., KAPRAVELOS, A., STRINGHINI, G., HOLZ, T., KRUEGEL, C., AND VIGNA, G. The dark alleys of madison avenue: Understanding malicious advertisements. In *Proceedings of the 2014 Conference on Internet Measurement Conference* (2014), ACM, pp. 373–380.

Appendices

A HTTPS Mixed Content Classification

To classify URLs in the HTTPS mixed content analysis, we used the block lists described in Section 4. Additionally, we include a list of CDNs from the WebPagetest Project¹⁶. The mixed content URL is then classified according to the first rule it satisfies in the following list:

1. If the requested domain matches the landing page domain, and the request URL ends with `favicon.ico` classify as a “favicon”.
2. If the requested domain matches the landing page domain, classify as the site’s “own content”.
3. If the requested domain is marked as “should block” by the blocklists, classify as “tracker”.

¹⁶<https://github.com/WPO-Foundation/webpagetest>



Figure 11: Sample canvas fingerprinting images. These three images represent the final canvas images created by fingerprinting scripts, which are subsequently hashed and used to identify the device.

4. If the requested domain is in the CDN list, classify as “CDN”.
5. Otherwise, classify as “non-tracking” third-party content.

B WebRTC ICE Candidate Generation

It is possible for a Javascript web application to access ICE candidates, and thus access a user’s local IP addresses and public IP address, without explicit user permission. Although a web application must request explicit user permission to access audio or video through WebRTC, the framework allows a web application to construct an `RTCDataChannel` without permission. By default, the data channel will launch the ICE protocol and thus enable the web application to access the IP address information without any explicit user permission. Both users behind a NAT and users behind a VPN/proxy can have additional identifying information exposed to websites without their knowledge or consent.

Several steps must be taken to have the browser generate ICE candidates. First, a `RTCDataChannel` must be created as discussed above. Next, the `RTCPeerConnection.createOffer()` must be called, which generates a `Promise` that will contain the session description once the offer has been created. This is passed to

Content-Type	Count
binary/octet-stream	8
image/jpeg	12664
image/svg+xml	177
image/x-icon	150
image/png	7697
image/vnd.microsoft.icon	41
text/xml	1
audio/wav	1
application/json	8
application/pdf	1
application/x-www-form-urlencoded	8
application/unknown	5
audio/ogg	4
image/gif	2905
video/webm	20
application/xml	30
image/bmp	2
audio/mpeg	1
application/x-javascript	1
application/octet-stream	225
image/webp	1
text/plain	91
text/javascript	3
text/html	7225
video/ogg	1
image/*	23
video/mp4	19
image/pjpeg	2
image/small	1
image/x-png	2

Table 10: Counts of responses with given Content-Type which cause mixed content errors. NOTE: Mixed content blocking occurs based on the tag of the initial request (e.g. `image src` tags are considered passive content), not the response Content-Type. Thus it is likely that the Javascript and other active content loads listed above are the result of misconfigurations and mistakes that will be dropped by the browser. For example, requesting a Javascript file with an image tag.

`RTCPeerConnection.setLocalDescription()`, which triggers the gathering of candidate addresses. The prepared offer will contain the supported configurations for the session, part of which includes the IP addresses gathered by the ICE Agent.¹⁷ A web application can retrieve these candidate IP addresses by using the event handler `RTCPeerConnection.onicecandidate()` and retrieving the candidate IP address from the `RTCPeerConnectionIceEvent.candidate` or, by parsing the resulting Session Description Protocol (SDP)¹⁸ string from

¹⁷<https://w3c.github.io/webrtc-pc/#widl-RTCPeerConnection-createOffer-Promise-RTCSessionDescription--RTCOfferOptions-options>
¹⁸<https://tools.ietf.org/html/rfc3264>

`RTCPeerConnection.localDescription` after the offer generation is complete. In our study we only found it necessary to instrument `RTCPeerConnection.onicecandidate()` to capture all current scripts.

C Additional methodological details

All measurements are run with Firefox version 41. The Ghostery measurements use version 5.4.10 set to block all possible bugs and cookies. The HTTPS Everywhere measurement uses version 5.1.0 with the default settings. The Block TP Cookies measurement sets the Firefox setting to “block all third-party cookies”

C.1 Classifying Third-party content

In order to determine if a request is a first-party or third-party request, we utilize the URL’s “public suffix + 1” (or PS+1). A public suffix is “is one under which Internet users can (or historically could) directly register names. [Examples include] .com, .co.uk and pvt.k12.ma.us.” A PS+1 is the public suffix with the section of the domain immediately preceding it (not including any additional subdomains). We use Mozilla’s Public Suffix List¹⁹ in our analysis. We consider a site to be a potential third-party if the PS+1 of the site does not match the landing page’s PS+1 (as determined by the algorithm in Appendix C.2). Throughout the paper we use the word “domain” to refer to a site’s PS+1.

C.2 Detection of landing pages from HTTP data

Upon visiting a site, the browser may either be redirected by a response header (with a 3XX HTTP response code or “Refresh” field), or by the page content (with javascript or a “Refresh” meta tag). Several redirects may occur before the site arrives at its final landing page and begins to load the remainder of the content. To capture all possible redirects we use the following recursive algorithm, starting with the initial request to the top-level site. For each request:

1. If HTTP redirect, following it preserving referrer details from previous request.
2. If the previous referrer is the same as the current we assume content has started to load and return the current referrer as the landing page.
3. If the current referrer is different from the previous referrer, and the previous referrer is seen in future requests, assume it is the actual landing page and return the previous referrer.

¹⁹<https://publicsuffix.org/>

4. Otherwise, continue to the next request, updating the current and previous referrer.

This algorithm has two failure states: (1) a site redirects, loads additional resources, then redirects again, or (2) the site has no additional requests with referrers. The first failure mode will not be detected, but the second will be. From manual inspection, the first failure mode happens very infrequently. For example, we find that only 0.05% of sites are incorrectly marked as having HTTPS as a result of this failure mode. For the second failure mode, we find that we can’t correctly label the landing pages of 2973 first-party sites (0.32%) on the top 1 million sites. For these sites we fall back to the requested top-level URL.

C.3 Detecting Cookie Syncing

We consider two parties to have cookie synced if a cookie ID appears in specific locations within the *referrer*, *request*, and *location* URLs extracted from HTTP request and response pairs. We determine cookie IDs using the algorithm described in Section 4. To determine the *sender* and *receiver* of a synced ID we use the following classification, in line with previous work [1, 42]:

- If the ID appears in the *request* URL: the requested domain is the recipient of a synced ID.
- If the ID appears in the *referrer* URL: the referring domain is the sender of the ID, and the requested domain is the receiver.
- If the ID appears in the *location* URL: the original requested domain is the sender of the ID, and the redirected location domain is the receiver.

This methodology does not require reverse engineering any domain’s cookie sync API or URL pattern. An important limitation of this generic approach is the lack of discrimination between intentional cookie syncing and accidental ID sharing. The latter can occur if a site includes a user’s ID within its URL query string, causing the ID to be shared with all third parties in the referring URL.

The results of this analysis thus provide an accurate representation of the privacy implications of ID sharing, as a third party has the technical capability to use an unintentionally shared ID for any purpose, including tracking the user or sharing data. However, the results should be interpreted only as an upper bound on cookie syncing as the practice is defined in the online advertising industry.

C.4 Detection of Fingerprinting

Javascript minification is used to reduce the size of a file for transit. Additionally, Javascript files can be obfuscated, such that the majority of the script is stored in

one or several obfuscated strings which are transformed and evaluated at run time using various string operations and the `eval` function. This makes static analysis difficult, if not impossible, for these scripts, as the script can't be re-constructed without executing the bundled parser. Anecdotally, we have found that obfuscation is not uncommon in fingerprinting and tracking scripts, motivating the use of a dynamic approach. With our detection methodology, we intercept and record access to specific Javascript objects, which is not hindered by minification or obfuscation of the source code.

The methodology builds on that used by Acar, et.al. [1] to detect canvas fingerprinting. Using the Javascript calls instrumentation described in Section 3.1, we record access to specific APIs which have been found to be used to fingerprint the browser. Each time an instrumented object is accessed, we record the full context of the access: the URL of the calling script, the top-level url of the site, the property and method being accessed, any provided arguments, and any properties set or returned. For each fingerprinting method, we design a detection algorithm which takes the context as input and returns a binary classification of whether or not a script uses that method of fingerprinting when embedded on that first-party site.

When manual verification is necessary, we have two approaches which depend on the level of script obfuscation. If the script is not obfuscated we manually inspect the copy which was archived according to the procedure discussed in Section 3.1. If the script is obfuscated beyond inspection, we embed a copy of the script in isolation on a dummy HTML page and inspect it using the Firefox Javascript Deobfuscator²⁰ extension. We also occasionally spot check live versions of sites and scripts, falling back to the archive when there are discrepancies.

C.5 Instrumenting JavaScript calls

Our instrumentation of JavaScript calls relies on overriding of getters and setters for all properties and methods of each instrumented object. The scripts being measured run at the same privilege level, so they might erase or override our instrumentation changes, preventing us from recording access to the object. Similarly, the script could first check if a custom getter is present before executing any fingerprinting code.

However, this would in turn be detectable since we would observe access to the `__define{G,S}etter__` or `__lookup{G,S}etter__` methods for the object in question and could investigate the cause. In our 1 million site measurement, we only observe script access to getters or setters for `HTMLCanvasElement` and `CanvasRenderingContext2D` interfaces. All of

²⁰<https://addons.mozilla.org/en-US/firefox/addon/javascript-deobfuscator/>

these are benign accesses from 47 scripts total, with the majority related to an HTML canvas graphics library.

D AudioContext Fingerprinting Configuration

Figure 8 in Section 6.4 summarizes the two audio fingerprinting configurations found in the wild.

The top configuration, used by a single script (`*.cdn-net.com/cc.js`), utilizes `AudioContext` to generate a fingerprint. First, the script generates a triangle wave using an `OscillatorNode`. This signal is passed through an `AnalyserNode` and a `ScriptProcessorNode` (omitted from Figure 8). Finally, the signal is passed into a through a `GainNode` with gain set to zero to mute any output before being connect to the `AudioContext`'s destination (e.g. the computer's speakers). The `AnalyserNode` provides access to a Fast Fourier Transform (FFT) of the audio signal, which is captured using the `onaudioprocess` event handler added by the `ScriptProcessorNode`. The resulting FFT is fed into a hash and used as a fingerprint.

The bottom configuration, used by two scripts (`client.a.pxi.pub/*/main.min.js` and `http://js.ad-score.com/score.min.js`), uses a similar technique as the previous script with two notable differences. The scripts use a `DynamicsCompressorNode`, possibly to increase differences in processed audio between machines. Rather than access the FFT of a muted stream, it uses `OfflineAudioContext`, which outputs the processed audio to a buffer available to the script. This removes the need for an `AnalyserNode`, `GainNode`, or `ScriptProcessorNode`. The script uses a hash of the sum of values from the buffer as the fingerprint.

Fingerprinting Script	Count
cdn.doubleverify.com/dvtp_src_internal24.js	4588
cdn.doubleverify.com/dvtp_src_internal23.js	2963
ap.lijit.com/sync	2653
cdn.doubleverify.com/dvbs_src.js	2093
rtbcdn.doubleverify.com/bsredirect5.js	1208
g.alicdn.com/alilog/mlog/aplus_v2.js	894
static.audienceinsights.net/t.js	498
static.boo-box.com/javascripts/embed.js	303
admicro1.vcmedia.vn/core/fipmin.js	180
c.imedia.cz/js/script.js	173
ap.lijit.com/www/delivery/fp	140
www.lijit.com/delivery/fp	127
s3-ap-southeast-1.amazonaws.com/af-bdaz/bquery.js	118
d38nbbai6u794i.cloudfront.net/*/platform.min.js	97
voken.eyereturn.com/	85
p8h7t6p2.map2.ssl.hwcdn.net/fp/Scripts/PixelBundle.js	72
static.fraudmetrix.cn/fm.js	71
e.e701.net/cpc/js/common.js	56
tags.bkrtx.com/js/bk-coretag.js	56
dt617kogtco.cloudfront.net/sauce.min.js	55
685 others	1853
	18283
TOTAL	14371 unique ¹

Table 11: Canvas fingerprinting scripts on the top Alexa 1 Million sites.
*: Some URLs are truncated for brevity.
1: Some sites include fingerprinting scripts from more than one domain.

Fingerprinting script	# of sites	Text drawn into the canvas
mathid.mathtag.com/device/id.js		
mathid.mathtag.com/d/i.js	2941	mmmmmmmmmmlli
admicro1.vcmedia.vn/core/fipmin.js	243	abcdefghijklmnopqr[snip]
*.online-metrix.net ¹	75	gMcdefghijklmnopqrstuvwxyz0123456789
pixel.infernotions.com/pixel/	2	mmmmmmmmmmMMMMMMMMM=llllllllll“.
api.twisto.cz/v2/proxy/test*	1	mmmmmmmmmmlli
go.lynxbroker.de/eat_session.js	1	mimimimimimi[snip]
TOTAL	3263 (3250 unique ²)	-

Table 12: Canvas font fingerprinting scripts on the top Alexa 1 Million sites.
*: Some URLs are truncated for brevity.
1: The majority of these inclusions were as subdomain of the first-party site, where the DNS record points to a subdomain of online-metrix.net.
2: Some sites include fingerprinting scripts from more than one domain.

Fingerprinting Script	First-party Count	Classification
cdn.augur.io/augur.min.js	147	Tracking
click.sabavision.com/*/jsEngine.js	115	Tracking
static.fraudmetrix.cn/fm.js	72	Tracking
*.hwcdn.net/fp/Scripts/PixelBundle.js	72	Tracking
www.cdn-net.com/cc.js	45	Tracking
scripts.poll-maker.com/3012/scpolls.js	45	Tracking
static-hw.xvideos.com/vote/displayFlash.js	31	Non-Tracking
g.alicdn.com/security/umscript/3.0.11/um.js	27	Tracking
load.instinctiveads.com/s/js/afp.js	16	Tracking
cdn4.forter.com/script.js	15	Tracking
socauth.privatbank.ua/cp/handler.html	14	Tracking
retailautomata.com/ralib/magento/raa.js	6	Unknown
live.activeconversion.com/ac.js	6	Tracking
olui2.fs.ml.com/publish/ClientLoginUI/HTML/cc.js	3	Tracking
cdn.geocomply.com/101/gc-html5.js	3	Tracking
retailautomata.com/ralib/shopifynew/raa.js	2	Unknown
2nyan.org/animal/	2	Unknown
pixel.infernotions.com/pixel/	2	Tracking
167.88.10.122/ralib/magento/raa.js	2	Unknown
80 others present on a single first-party	80	-
TOTAL	705	-

Table 13: WebRTC Local IP discovery on the Top Alexa 1 Million sites.

*: Some URLs are truncated for brevity.

Site	Prominence	# of FP	Rank Change
doubleclick.net	6.72	447,963	+2
google-analytics.com	6.20	609,640	-1
gstatic.com	5.70	461,215	-1
google.com	5.57	397,246	0
facebook.com	4.20	309,159	+1
googlesyndication.com	3.27	176,604	+3
facebook.net	3.02	233,435	0
googleadservices.com	2.76	133,391	+4
fonts.googleapis.com	2.68	370,385	-4
scorecardresearch.com	2.37	59,723	+13
adnxs.com	2.37	94,281	+2
twitter.com	2.11	143,095	-1
fbcdn.net	2.00	172,234	-3
ajax.googleapis.com	1.84	210,354	-6
yahoo.com	1.83	71,725	+5
rubiconproject.com	1.63	45,333	+17
openx.net	1.60	59,613	+7
googletagservices.com	1.52	39,673	+24
mathtag.com	1.45	81,118	-3
advertising.com	1.45	49,080	+9

Table 14: Top 20 third-parties on the Alexa top 1 million, sorted by prominence. The number of first-party sites each third-party is embedded on is included. Rank change denotes the change in rank between third-parties ordered by first-party count and third-parties ordered by prominence.