Online Tracking: A 1-million-site Measurement and Analysis

Steven Englehardt
Princeton University
ste@cs.princeton.edu

Arvind Narayanan
Princeton University
arvindn@cs.princeton.edu

This is an extended version of our paper that appeared at ACM CCS 2016.

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 open-source web privacy measurement tool, OpenWPM[1], 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. We demonstrate our platform's strength in enabling researchers to rapidly detect, quantify, and characterize emerging online tracking behaviors.

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, 15]. 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.

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 previous engineering efforts. (1) We achieve scale through 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 [31] (a limitation we explore in detail in Section 3.3). (2) We provide comprehensive instrumentation by expanding on the rich browser extension instrumentation of FourthParty [33], 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.

A large-scale view of web tracking and privacy.

In this paper we report results from a January 2016 measurement of the top 1 million sites (Section 4). 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 Audio 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.

[1] https://github.com/citp/OpenWPM

[2] http://www.seleniumhq.org/

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 of 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 "referer" 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 referer header.

Web privacy measurement platforms. 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 is 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, which the authors state 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 supports command re-use between studies.

Chameleon Crawler is a Chromium based crawler that utilizes the Chameleon[3] 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 instrumenting the browser and does not handle automation [33]. OpenWPM has incorporated and expanded upon nearly all of FourthParty's instrumentation (Section 3).

WebXray is a PhantomJS based tool for measuring HTTP traffic [31]. It has been used to study third-party inclusions on the top 1 million sites, but as we show in Section 3.3, measurements with a stripped-down browser have the potential to miss a large number of resource loads.

TrackingObserver is a Chrome extension that detects tracking and exposes APIs for extending its functionality such as measurement and blocking [48].

XRay [27] and *AdFisher* [9] are tools for running automated personalization detection experiments. AdFisher builds on similar technologies as OpenWPM (Selenium, xvfb), but is not intended for tracking measurements.

Common Crawl[4] uses an Apache Nutch based crawler. The Common Crawl dataset is the largest publicly available web crawl[5], with billions of page visits. However, the crawler used does not execute Javascript or other dynamic content during a page visit. Privacy studies which use the dataset [49] will miss dynamically loaded content, which includes many advertising resources.

Crowd-sourcing of web privacy and personalization measurement is an important alternative to automated browsing. $heriff and Bobble are two platforms for measuring personalization [35, 65]. Two major challenges are participant privacy and providing value to users to incentivize participation.

Previous findings. Krishnarmurthy and Wills [24] 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. In the following years, studies show a continual increase in third-party tracking and in the diversity of tracking techniques [33, 48, 20, 2, 1, 4]. Lerner et al. also find an increase in the prevalence and complexity of tracking, as well as an increase in the interconnectedness of the ecosystem by analyzing Internet Archive data from 1996 to 2016 [29]. Fruchter et al. studied geographic variations in tracking [17]. More recently, Libert studied third-party HTTP requests on the top 1 million sites [31], providing view of tracking across the web. In this study, Libert showed 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 reinstantiate HTTP cookies [53], 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 [36, 30] and legal settlements [51, 10] against the companies participating in the practice. However several follow up studies by other research groups confirmed that, despite a reduction in usage (particularly in the U.S.), the technique is still used for tracking [48, 34, 1].

Device fingerprinting is a persistent tracking technique which does not require a tracker to set any state in the user's

[3]https://github.com/ghostwords/chameleon

[4]https://commoncrawl.org

[5]https://aws.amazon.com/public-data-sets/common-crawl/

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 device fingerprints are unique [12, 26]. New fingerprinting techniques are continually discovered [37, 43, 16], and are subsequently used to track users on the web [41, 2, 1]. 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 [9]. Lécuyer et al. develop XRay, a system for inferring which pieces of user data are used for personalization [27]. Another system by some of the same authors is Sunlight which improves upon their previous methodology to provide statistical confidence of their targeting inferences [28].

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 [19, 63]; *steering*, a gentler form of price discrimination where a product search shows differently-priced results for different users [32]; 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 [65].

Web security measurement. Web security studies often 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 [67]. Headless browsers have been used in many web security measurements, for example: to measure the amount of third-party Javascript inclusions across many popular sites and the vulnerabilities that arise from how the script is embedded [40], to measure the presence of security seals on the top 1 million sites [62], and to study stand-alone password generators and meters on the web [60]. Several studies have used Selenium-based frameworks, including: to measure and categorize malicious advertisements displayed while browsing popular sites [68], to measure the presence of malware and other vulnerabilities on live streaming websites [46], to study HSTS deployment [21], to measure ad-injecting browser extensions [66], and to emulate users browsing malicious web shells with the goal of detecting client-side homephoning [55]. Other studies have analyzed Flash and Javascript elements of webpages to measure security vulnerabilities and error-prone implementations [42, 61].

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.

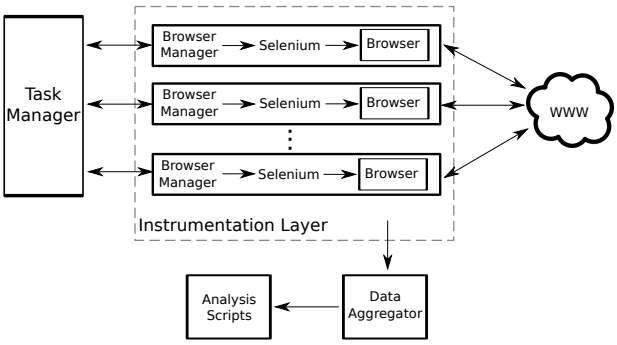

Figure 1: High-level overview of OpenWPM
The task manager monitors browser managers, which convert high-level commands into automated browser actions. The data aggregator receives and pre-processes data from instrumentation.

OpenWPM is open source and has already been used for measurement by several published studies. Section 3.4 in the supplementary materials examines the advanced features used by each study. In this paper we present, for the first time, the design and evaluation of the platform and highlight its strengths through several new measurements.

3.1 Design Motivations

OpenWPM builds on similar technologies as many previous platforms, but has several key design differences to supports modular, comprehensive, and maintainable measurement. Our platform supports stateful measurements while FPDetective [2] does not. Stateful measurements are important for studying the tracking ecosystem. Ad auctions may vary based on cookie data. A stateless browser always appears to be a new user, which skews cookie syncing measurements. In addition to cookie syncing studied in this paper, stateful measurements have allowed our platform to be used to study cookie respawning [1] and replicate realistic user profiles [14].

Many past platforms rely on native instrumentation code [39, 52, 2], which have a high maintenance cost and, in some cases a high cost-per-API monitored. In our platform, the cost of monitoring new APIs is minimal (Section 3.3) and APIs can be enabled or disabled in the add-on without recompiling the browser or rendering engine. This allows us to monitor a larger number of APIs. Native codebase changes in other platforms require constant merges as the upstream codebase evolves and complete rewrites to support alternative browsers.

3.2 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 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 PhantomJS. We currently use Selenium to drive Firefox, but Selenium's support for multiple browsers makes it easy to transition to others 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 [45]. 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.

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 (Section 3.3). 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 [50], 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 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 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, by using the pyvirtualdisplay library to interface with Xvfb, which draws the graphical interface of the browser to a virtual frame buffer.

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.

Data Aggregator: Providing repeatability. Repeatability can be achieved logging data in a standardized format, so research groups can easily share scripts and data. We aggregate data 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.

Instrumentation: Supporting comprehensive and reusable measurement. 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 domains are setting Flash cookies, and to record access to cookies in the absence of other instrumentation

HTTP Data — After examining several Python HTTP proxies, we chose to use Mitmproxy[6] 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. We use both `Content-Type` and file extension checking to detect scripts in the proxy. Once detected, a script is decompressed (if necessary) and hashed. The hash and content are sent 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 expands on the work of Fourthparty [33]. In particular, we utilize Fourthparty's Javascript instrumentation, which defines custom getters and setters on the `window.navigator` and `window.screen` interfaces[7]. We updated and extended this functionality to record access to the prototypes of the `Storage`, `HTMLCanvasElement`, `CanvasRenderingContext2D`, `RTCPeerConntection`, `AudioContext` objects, as well as the prototypes of several children of `AudioNode`. This records the setting and getting of all

[6]https://mitmproxy.org/

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

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. Everything is logged directly to the SQLite aggregator.

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 Extension[8].

In an adversarial situation, a script could disable our instrumentation before fingerprinting a user by overriding access to getters and setters for each instrumented object. However, this would 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 these are benign accesses from 47 scripts total, with the majority related to an HTML canvas graphics library.

Example workflow.

1. The researcher issues a command to the task manager and specifies that it should synchronously execute on all browser managers.
2. The task manager checks all of the command execution threads and blocks until all browsers are available to execute a new command.
3. The task manager creates new command execution threads for all browsers and sends the command and command parameters over a pipe to the browser manager process.
4. The browser manager interprets this command and runs the necessary Selenium code to execute the command in the browser.
5. If the command is a "Get" command, which causes the browser to visit a new URL, the browser manager distributes the browser ID and top-level page being visited to all enabled instrumentation modules (extension, proxy, or disk monitor).
6. Each instrumentation module uses this information to properly key data for the new page visit.
7. The browser manager can send returned data (e.g. the parsed contents of a page) to the SQLite aggregator.
8. Simultaneously, instrumentation modules send data to the respective aggregators from separate threads or processes.
9. Finally, the browser manager notifies the task manager that it is ready for a new command.

3.3 Evaluation

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 instrumented browser can visit around 3500 sites per day, requiring no manual interaction during that time. 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 used stripped-down browsers such as PhantomJS for studies, trading off fidelity for speed.

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, several sites (such as `dropbox.com`) seem to break when PhantomJS presents the incorrect user-agent string. This is because sites may 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. These findings support our view that OpenWPM enables significantly more complete and realistic web and tracking measurement than stripped-down browsers.

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 machine[9]. 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. 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 few additional lines of code. The instrumentation necessary to measure canvas fingerprinting (Section 6.1) is three additional lines of code, while the We-

[8]https://github.com/EFForg/privacybadgerfirefox

[9]https://aws.amazon.com/ec2/instance-types/

Study	Year	Browser automation	Stateful crawls	Persistent profiles	Fine-grained profiles	Advanced plugin support	Automated login	Detect tracking cookies	Monitor state changes	Javascript Instrumentation	Content extraction
Persistent tracking mechanisms [1]	2014	●	●	●	●	●		●	●		
FB Connect login permissions [47]	2014	●					●				○
Surveillance implications of web tracking [14]	2015	●	●			●		●			
HSTS and key pinning misconfigurations [21]	2015	●	●			●	○				●
The Web Privacy Census [4]	2015	●	●			●			●		
Geographic Variations in Tracking [17]	2015	●				●					
Analysis of Malicious Web Shells [55]	2016	●									
This study (Sections 5 & 6)	2016	●	●	●	●	●		●	●	●	

Table 1: Seven published studies which utilize our Platform.
An unfilled circle indicates that the feature was useful but application-specific programming or manual effort was still required.

bRTC measurement (Section 6.3) is just a single line of code. 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.

3.4 Applications of OpenWPM

Seven academic studies have been published in journals, conferences, and workshops, utilizing OpenWPM to perform a variety of web privacy and security measurements.[10] Table 1 summarizes the advanced features of the platform that 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, 14] 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, 17] 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 [47].

[10]We are aware of several other studies in progress.

4. WEB CENSUS 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 eash 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 3.2.

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

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.

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, 14]. 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:

$$(\text{name}_1 =)\text{value}_1|...|(\text{name}_N =)\text{value}_N$$

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) $8 \leq$ length(parameter-value) ≤ 100, (3) the parameter-value 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 two popular *tracking-protection lists* for this purpose: EasyList and EasyPrivacy. Including EasyList allows us to classify advertising related trackers, while EasyPrivacy detects non-advertising related trackers. The two lists consist of regular expressions and URL substrings which are matched against resource loads to determine if a request should be blocked.

Alternative tracking-protection lists exist, such as the list built into the Ghostery browser extension and the domain-based list provided by Disconnect[11]. Although we don't use these lists to classify trackers directly, we evaluate their performance in several sections.

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.

Limitations. The analysis presented in this paper has several methodological and measurement limitations. Our platform did not interact with sites in ways a real user might; we did not log into sites nor did we carry out actions such

[11] https://disconnect.me/trackerprotection

as scrolling or clicking links during our visit. While we have performed deeper crawls of sites (and plan to make this data publicly available), the analyses presented in the paper pertain only to homepages.

For comparison, we include a preliminary analysis of a crawl which visits 4 internal pages in addition to the homepage of the top 10,000 sites. The analyses presented in this paper should be considered a lower bound on the amount of tracking a user will experience in the wild. In particular, the average number of third parties per site increases from 22 to 34. The 20 most popular third parities embedded on the homepages of sites are found on 6% to 57% more sites when internal page loads are considered. Similarly, fingerprinting scripts found in Section 6 were observed on more sites. Canvas fingerprinting increased from 4% to 7% of the top sites while canvas-based font fingerprinting increased from 2% to 2.5%. An increase in trackers is expected as each additional page visit within a site will cycle through new dynamic content that may load a different set of third parties. Additionally, sites may not embed all third-party content into their homepages.

The measurements presented in this paper were collected from an EC2 instance in Amazon's US East region. It is possible that some sites would respond differently to our measurement instance than to a real user browsing from residential or commercial internet connection. That said, Fruchter, et al. [17] use OpenWPM to measure the variation in tracking due to geographic differences, and found no evidence of tracking differences caused by the origin of the measurement instance.

Although OpenWPM's instrumentation measures a diverse set of tracking techniques, we do not provide a complete analysis of all known techniques. Notably absent from our analysis are non-canvas-based font fingerprinting [2], navigator and plugin fingerprinting [12, 33], and cookie respawning [53, 6]. Several of these javascript-based techniques are currently supported by OpenWPM, have been measured with OpenWPM in past research [1], and others can be easily added (Section 3.3). Non-Javascript techniques, such as font fingerprinting with Adobe Flash, would require additional specialized instrumentation.

Finally, for readers interested in further details or in reproducing our work, we provide further methodological details in the Appendix: what constitutes distinct domains (13.1), how to detect the landing page of a site using the data collected by our Platform (13.2), how we detect cookie syncing (13.3), and why obfuscation of Javascript doesn't affect our ability to detect fingerprinting (13.4).

5. RESULTS OF 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, Twitter, and AdNexus are the only third-party entities present on more than 10% of sites.*

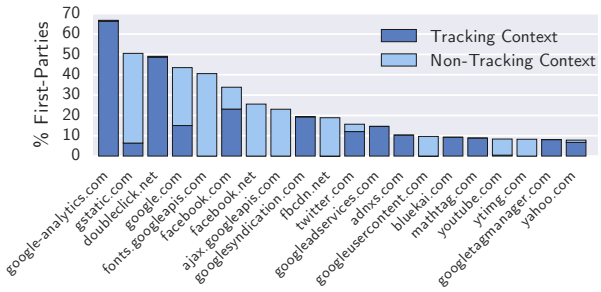

Figure 2: Top third parties on the top 1 million sites. Not all third parties are classified as trackers, and in fact the same third party can be classified differently depending on the context. (Section 4).

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.

We can expand on this by analyzing the top third-party *organizations*, many of which consist of multiple entities. As an example, Facebook and Liverail are separate entities but Liverail is owned by Facebook. We use the domain-to-organization mappings provided by Libert [31] and Disconnect[11]. As shown in Figure 3, Google, Facebook, Twitter, Amazon, AdNexus, and Oracle are the third-party organizations present on more than 10% of sites. In comparison to Libert's [31] 2014 findings, Akamai and ComScore fall significantly in market share to just 2.4% and 6.6% of sites. Oracle joins the top third parties by purchasing BlueKai and AddThis, showing that acquisitions can quickly change the tracking landscape.

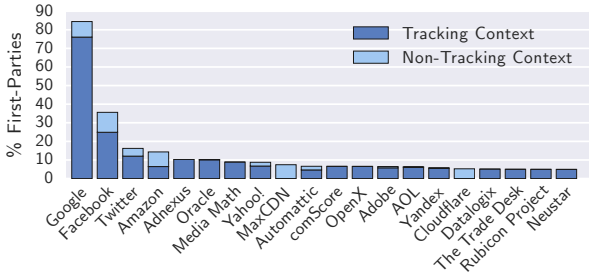

Figure 3: Organizations with the highest third-party presence on the top 1 million sites. Not all third parties are classified as trackers, and in fact the same third party can be classified differently depending on the context. (Section 4).

Larger entities may be easier to regulate by public-relations pressure and the possibility of legal or enforcement actions, an outcome we have seen in past studies [1, 6, 34].

5.2 Prominence: a third party ranking metric

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, 22]. The exact relationship is not important to us; any formula for traffic can be plugged into our prominence metric below.

Definition:.

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

where $edge(s,t)$ 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 4. 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 4) makes it ideally suited for these purposes.

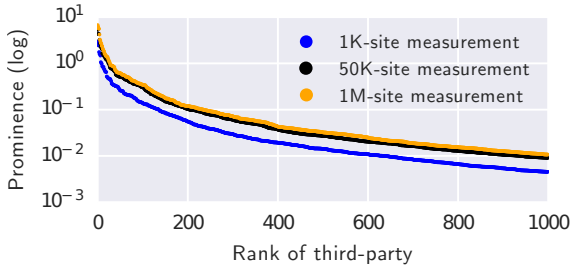

Figure 4: Prominence of third party as a function of prominence rank. We posit that the curve for the 1M-site measurement (which can be approximated by a 50k-site measurement) presents a useful aggregate picture of tracking.

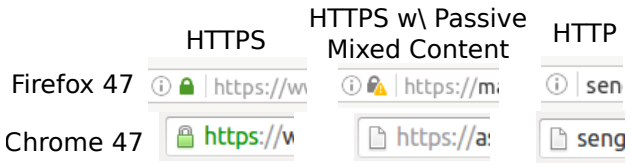

Figure 5: Secure connection UI for Firefox Nightly 47 and Chrome 47. Clicking on the lock icon in Firefox reveals the text "Connection is not secure" when mixed content is present.

	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.

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 [13]. 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 [57, 64].

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 [38]. *Passive* mixed content, that is, non-executable resources loaded over HTTP, cause the browser to display an insecure warning to the user but still load 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 sites, 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. 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 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 display-

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 55k % FP
Own	25.4%	24.9%
Favicon	2.1%	2.6%
Tracking	10.4%	20.1%
CDN	1.6%	2.6%
Non-tracking	44.9%	35.4%
Multiple causes	15.6%	6.3%

Table 5: A breakdown of causes of passive mixed-content warnings on the top 1M sites and on the top 55k sites. "Non-tracking" represents third-party content not classified as a tracker or a CDN.

ing mixed content errors to their users, the majority of which (92%) would contain active content which would be blocked.

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

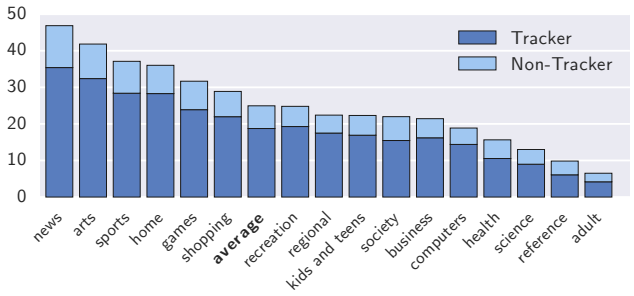

Figure 6: Average # of third parties in each Alexa category.

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 [14], 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).

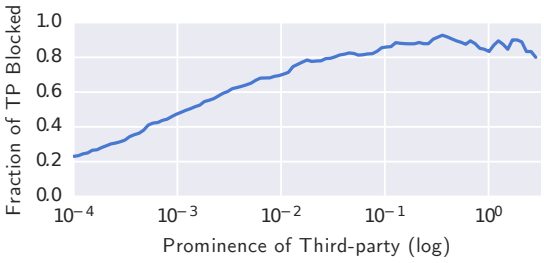

Figure 7: Fraction of third parties blocked by Ghostery as a function of the prominence of the third party. As defined earlier, a third party's prominence is the sum of the inverse ranks of the sites it appears on.

We also tested Ghostery, and found that it is effective at reducing the number of third parties and ID cookies (Figure 11 in the Appendix). 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 works well at achieving its stated objectives.

However, the tool is less effective for obscure trackers (prominence < 0.1). In Section 6.6, we show that less promi-

nent fingerprinting 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 the Appendix (Section 13.3), 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). We present details of the top cookie-syncing parties in Appendix 13.3.

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 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 [56, 54, 18]. 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 [14, 23, 25]). 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` 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 3.2. 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).

	% of First-parties		
Rank Interval	**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 [37]. 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.

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` interface

and of the CanvasRenderingContext2D 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.[12]
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 16px × 16px.

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 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 12 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 [12]. 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 [41, 2]. But can fonts be enumerated using the Canvas interface? The only public discussion of the technique seems to be a Tor Browser ticket from 2014[13]. To the best of our knowledge, we are the first to measure its usage in the wild.

[12]The default canvas size is 300px × 150px.

[13]https://trac.torproject.org/projects/tor/ticket/13400

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. See Table 12 in the Appendix for a full list of scripts.

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 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 [59]. 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 Section 11.

Detection methodology. To detect WebRTC local IP discovery, we instrument the `RTCPeerConnection` 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`[14]. 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. Otherwise, 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 a 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

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

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 Table 13 in the Appendix.

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. These results have implications for the ongoing debate on whether or not unsolicited WebRTC IP discovery should be private by default [59, 8, 58].

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.

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 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.

In the simplest case, a script from the company Liverail[15] 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 is conceptually similar to 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 audio fingerprinting configurations found in three scripts. The top configuration utilizes an `AnalyserNode` to extract an FFT to build the fingerprint. Both configurations process an audio signal from an `OscillatorNode` before reading the resulting signal and hashing it to create a device audio fingerprint. Full configuration details are in Appendix Section 12.

We created a demonstration page based on the scripts, which attracted visitors with 18,500 distinct cookies as of this submission. These 18,500 devices hashed to a total of 713 different fingerprints. We estimate the entropy of the fingerprint at 5.4 bits based on our sample. We leave a full evaluation of the effectiveness of the technique to future work.

We find that this technique is very infrequently used as of March 2016. The most popular script is from Liverail, present on 512 sites. Other scripts were present on as few as 6 sites. 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 API, which allows a site to query the browser

[15]https://www.liverail.com/

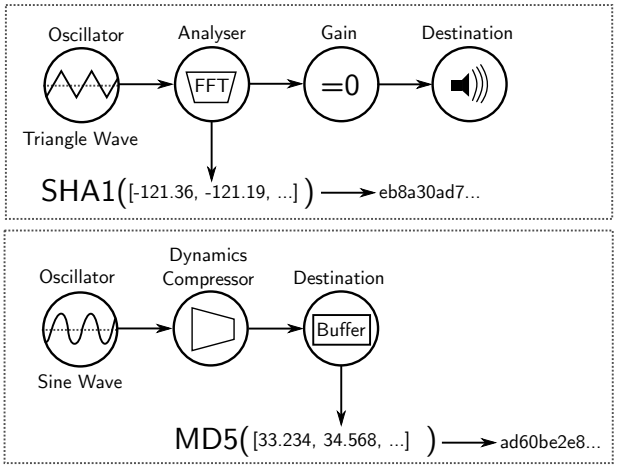

Figure 8: AudioContext node configuration used to generate a fingerprint. **Top:** Used by www.cdn-net.com/cc.js in an `AudioContext`. **Bottom:** Used by client.a.pxi.pub/*/main.min.js and js.ad-score.com/score.min.js in an `OfflineAudioContext`. Full details in Appendix 12.

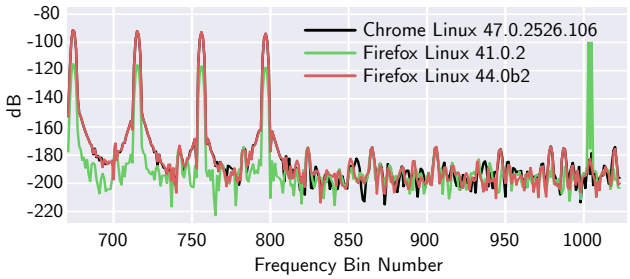

Figure 9: Visualization of processed `OscillatorNode` output from the fingerprinting script https://www.cdn-net.com/cc.js for three different browsers on the same machine. We found these values to remain constant for each browser after several checks.

for the current battery level or charging status of a host device. Olejnik et al. provide evidence that the Battery API can be used for tracking [43]. The authors show how the battery charge level and discharge time have a sufficient number of states and lifespan to be used as a short-term identifier. These status readouts can help identify users who take action to protect their privacy while already on a site. For example, the readout may remain constant when a user clears cookies, switches to private browsing mode, or opens a new browser before re-visiting the site. 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.

	Disconnect		EL + EP	
Technique	% Scripts	% Sites	% Scripts	% Sites
Canvas	17.6%	78.5%	25.1%	88.3%
Canvas Font	10.3%	97.6%	10.3%	90.6%
WebRTC	1.9%	21.3%	4.8%	5.6%
Audio	11.1%	53.1%	5.6%	1.6%

Table 9: Percentage of fingerprinting scripts blocked by Disconnect 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 may not change the rendering of a page or 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.

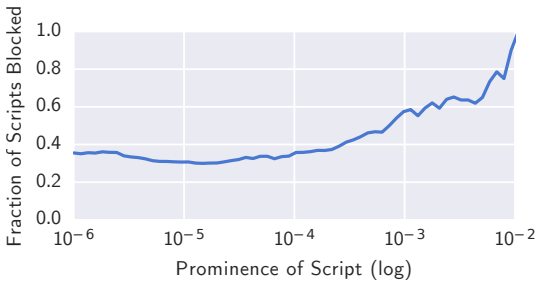

Figure 10: Fraction of fingerprinting scripts with prominence above a given level blocked by Disconnect, EasyList, or EasyPrivacy on the top 1M sites.

OpenWPM's active instrumentation (see Section 3.2) detects a large number of scripts not blocked by the current privacy tools. Disconnect and a combination of EasyList and EasyPrivacy both perform similarly in their block rate. The privacy tools block canvas fingerprinting on over 78% 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 10% 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.

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 Disconnect, EasyList, or Easyprivacy for all techniques analyzed in this section. 90% of scripts with a prominence above 0.01 are detected and blocked by one of the blocking lists, while only 35% 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 two components of the broader Web Transparency and Accountability Project at Princeton. 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 Telefonia Research, Vincent Toubiana at CNIL, France, Lukasz Olejnik at INRIA, France, Nick Nikiforakis at Stony Brook, 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. Finally, we'd like to thank the anonymous reviewers of this paper.

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

9. REFERENCES

[1] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of CCS*, 2014.

[2] G. Acar, M. Juarez, N. Nikiforakis, C. Diaz, S. Gürses, F. Piessens, and B. Preneel. FPDetective: dusting the web for fingerprinters. In *Proceedings of CCS*. ACM, 2013.

[3] L. A. Adamic and B. A. Huberman. Zipf's law and the internet. *Glottometrics*, 3(1):143–150, 2002.

[4] H. C. Altaweel I, Good N. Web privacy census. *Technology Science*, 2015.

[5] J. Angwin. What they know. The Wall Street Journal. http://online.wsj.com/public/page/what-they-know-digital-privacy.html, 2012.

[6] 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.

[7] P. E. Black. Ratcliff/Obershelp pattern recognition. http://xlinux.nist.gov/dads/HTML/ratcliffObershelp.html, Dec. 2004.

[8] Bugzilla. WebRTC Internal IP Address Leakage. https://bugzilla.mozilla.org/show_bug.cgi?id=959893.

[9] A. Datta, M. C. Tschantz, and A. Datta. Automated experiments on ad privacy settings. *Privacy Enhancing Technologies*, 2015.

[10] W. Davis. KISSmetrics Finalizes Supercookies Settlement. http://www.mediapost.com/publications/article/191409/kissmetrics-finalizes-supercookies-settlement.html, 2013. [Online; accessed 12-May-2014].

[11] Disconnect. Tracking Protection Lists. https://disconnect.me/trackerprotection.

[12] P. Eckersley. How unique is your web browser? In *Privacy Enhancing Technologies*. Springer, 2010.

[13] Electronic Frontier Foundation. Encrypting the Web. https://www.eff.org/encrypt-the-web.

[14] S. Englehardt, D. Reisman, C. Eubank, P. Zimmerman, J. Mayer, A. Narayanan, and E. W. Felten. Cookies that give you away: The surveillance implications of web tracking. In *24th International Conference on World Wide Web*, pages 289–299. International World Wide Web Conferences Steering Committee, 2015.

[15] 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.

[16] D. Fifield and S. Egelman. Fingerprinting web users through font metrics. In *Financial Cryptography and Data Security*, pages 107–124. Springer, 2015.

[17] N. Fruchter, H. Miao, S. Stevenson, and R. Balebako. Variations in tracking in relation to geographic location. In *Proceedings of W2SP*, 2015.

[18] S. Gorman and J. Valentino-Devries. New Details Show Broader NSA Surveillance Reach. http://on.wsj.com/1zcVv78, 2013.

[19] A. Hannak, G. Soeller, D. Lazer, A. Mislove, and C. Wilson. Measuring price discrimination and steering on e-commerce web sites. In *14th Internet Measurement Conference*, 2014.

[20] C. J. Hoofnagle and N. Good. Web privacy census. *Available at SSRN 2460547*, 2012.

[21] M. Kranch and J. Bonneau. Upgrading HTTPS in midair: HSTS and key pinning in practice. In *NDSS '15: The 2015 Network and Distributed System Security Symposium*, February 2015.

[22] S. A. Krashakov, A. B. Teslyuk, and L. N. Shchur. On the universality of rank distributions of website popularity. *Computer Networks*, 50(11):1769–1780, 2006.

[23] B. Krishnamurthy, K. Naryshkin, and C. Wills. Privacy leakage vs. protection measures: the growing disconnect. In *Proceedings of W2SP*, volume 2, 2011.

[24] B. Krishnamurthy and C. Wills. Privacy diffusion on the web: a longitudinal perspective. In *Conference on World Wide Web*. ACM, 2009.

[25] B. Krishnamurthy and C. E. Wills. On the leakage of personally identifiable information via online social networks. In *2nd ACM workshop on Online social networks*. ACM, 2009.

[26] P. Laperdrix, W. Rudametkin, and B. Baudry. 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.

[27] M. Lécuyer, G. Ducoffe, F. Lan, A. Papancea, T. Petsios, R. Spahn, A. Chaintreau, and R. Geambasu. Xray: Enhancing the web's transparency with differential correlation. In *USENIX Security Symposium*, 2014.

[28] M. Lecuyer, R. Spahn, Y. Spiliopolous, A. Chaintreau, R. Geambasu, and D. Hsu. Sunlight: Fine-grained targeting detection at scale with statistical confidence. In *Proceedings of CCS*. ACM, 2015.

[29] A. Lerner, A. K. Simpson, T. Kohno, and F. Roesner. Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016. In *Proceedings of USENIX Security)*, 2016.

[30] J. Leyden. Sites pulling sneaky flash cookie-snoop. http://www.theregister.co.uk/2009/08/19/flash_cookies/, 2009.

[31] T. Libert. Exposing the invisible web: An analysis of third-party http requests on 1 million websites. *International Journal of Communication*, 9(0), 2015.

[32] D. Mattioli. On Orbitz, Mac users steered to pricier hotels. http://online.wsj.com/news/articles/SB10001424052702304458604577488822667325882, 2012.

[33] J. R. Mayer and J. C. Mitchell. Third-party web tracking: Policy and technology. In *Security and Privacy (S&P)*. IEEE, 2012.

[34] A. M. McDonald and L. F. Cranor. Survey of the use of Adobe Flash Local Shared Objects to respawn HTTP cookies, a. *ISJLP*, 7, 2011.

[35] J. Mikians, L. Gyarmati, V. Erramilli, and N. Laoutaris. Detecting price and search discrimination on the internet. In *Workshop on Hot Topics in Networks*. ACM, 2012.

[36] N. Mohamed. You deleted your cookies? think again. http://www.wired.com/2009/08/you-deleted-your-cookies-think-again/, 2009.

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

[38] Mozilla Developer Network. Mixed content - Security. https://developer.mozilla.org/en-US/docs/Security/Mixed_content.

[39] C. Neasbitt, B. Li, R. Perdisci, L. Lu, K. Singh, and K. Li. Webcapsule: Towards a lightweight forensic engine for web browsers. In *Proceedings of CCS*. ACM, 2015.

[40] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. Van Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. You are what you include: Large-scale evaluation of remote javascript inclusions. In *Proceedings of CCS*. ACM, 2012.

[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] F. Ocariza, K. Pattabiraman, and B. Zorn. Javascript errors in the wild: An empirical study. In *Software Reliability Engineering (ISSRE)*. IEEE, 2011.

[43] L. Olejnik, G. Acar, C. Castelluccia, and C. Diaz. The leaking battery. *Cryptology ePrint Archive*, Report 2015/616, 2015.

[44] L. Olejnik, C. Castelluccia, et al. Selling off privacy at auction. In *NDSS '14: The 2014 Network and Distributed System Security Symposium*, 2014.

[45] Phantom JS. Supported web standards. http://www.webcitation.org/6hI3iptm5, 2016.

[46] M. Z. Rafique, T. Van Goethem, W. Joosen, C. Huygens, and N. Nikiforakis. It's free for a reason: Exploring the ecosystem of free live streaming services. In *Network and Distributed System Security (NDSS)*, 2016.

[47] N. Robinson and J. Bonneau. Cognitive disconnect: Understanding Facebook Connect login permissions. In *2nd ACM conference on Online social networks*. ACM, 2014.

[48] F. Roesner, T. Kohno, and D. Wetherall. Detecting and Defending Against Third-Party Tracking on the Web. In *Symposium on Networking Systems Design and Implementation*. USENIX, 2012.

[49] S. Schelter and J. Kunegis. On

the ubiquity of web tracking: Insights from a billion-page web crawl. *arXiv preprint arXiv:1607.07403*, 2016.

[50] Selenium
Browser Automation. Selenium faq. https://code.google.com/p/selenium/wiki/FrequentlyAskedQuestions, 2014.

[51] R. Singel. Online Tracking
Firm Settles Suit Over Undeletable Cookies. http://www.wired.com/2010/12/zombie-cookie-settlement/, 2010.

[52] K. Singh, A. Moshchuk, H. J.
Wang, and W. Lee. On the incoherencies in web browser access control policies. In *Proceedings of S&P*. IEEE, 2010.

[53] 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.

[54] 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.

[55] O. Starov, J. Dahse,
S. S. Ahmad, T. Holz, and N. Nikiforakis. No honor among thieves: A large-scale analysis of malicious web shells. In *International Conference on World Wide Web*, 2016.

[56] 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.

[57] Z. Tollman. We're Going HTTPS: Here's How WIRED Is Tackling a Huge Security Upgrade. https://www.wired.com/2016/04/wired-launching-https-security-upgrade/, 2016.

[58] J. Uberti. New proposal
for IP address handling in WebRTC. https://www.ietf.org/mail-archive/web/rtcweb/current/msg14494.html.

[59] J. Uberti and G. wei Shieh.
WebRTC IP Address Handling Recommendations. https://datatracker.ietf.org/doc/draft-ietf-rtcweb-ip-handling/.

[60] S. Van Acker, D. Hausknecht, W. Joosen, and A. Sabelfeld. Password meters and generators on the web: From large-scale empirical study to getting it right. In *Conference on Data and Application Security and Privacy*. ACM, 2015.

[61] S. Van Acker, N. Nikiforakis, L. Desmet,
W. Joosen, and F. Piessens. Flashover: Automated discovery of cross-site scripting vulnerabilities in rich internet applications. In *Proceedings of CCS*. ACM, 2012.

[62] T. Van Goethem, F. Piessens, W. Joosen, and N. Nikiforakis. Clubbing seals: Exploring the ecosystem of third-party security seals. In *Proceedings of CCS*. ACM, 2014.

[63] T. Vissers, N. Nikiforakis,
N. Bielova, and W. Joosen. Crying wolf? on the price discrimination of online airline tickets. HotPETS, 2014.

[64] W. V. Wazer. Moving the Washington Post to HTTPS. https://developer.washingtonpost.com/pb/blog/post/2015/12/10/moving-the-washington-post-to-https/, 2015.

[65] X. Xing, W. Meng, D. Doozan,
N. Feamster, W. Lee, and A. C. Snoeren. Exposing inconsistent web search results with bobble. In *Passive and Active Measurement*, pages 131–140. Springer, 2014.

[66] X. Xing, W. Meng, B. Lee, U. Weinsberg, A. Sheth,
R. Perdisci, and W. Lee. Understanding malvertising through ad-injecting browser extensions. In *24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2015.

[67] C. Yue and H. Wang. A measurement
study of insecure javascript practices on the web. *ACM Transactions on the Web (TWEB)*, 7(2):7, 2013.

[68] A. Zarras, A. Kapravelos, G. Stringhini,
T. Holz, C. Kruegel, and G. Vigna. The dark alleys of madison avenue: Understanding malicious advertisements. In *Internet Measurement Conference*. ACM, 2014.

APPENDIX

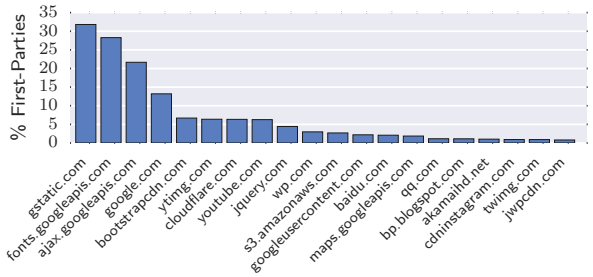

Figure 11: Third-party trackers on the top 55k sites with Ghostery enabled. The majority of the top third-party domains not blocked are CDNs or provide embedded content (such as Google Maps).

Figure 12: Three sample canvas fingerprinting images created by fingerprinting scripts, which are subsequently hashed and used to identify the device.

10. 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 Project[16]. The mixed content URL is then classfied 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".

[16]https://github.com/WPO-Foundation/webpagetest

3. If the requested domain is marked as "should block" by the blocklists, classify as "tracker".
4. If the requested domain is in the CDN list, classify as "CDN".
5. Otherwise, classify as "non-tracking" third-party content.

11. 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 `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 `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.

12. AUDIO FINGERPRINT CONFIGURATION

Figure 8 in Section 6.4 summarizes one of two audio fingerprinting configurations found in the wild. This configuration is used by two scripts, (client.a.pxi.pub/*/main.min.js and http://js.ad-score.com/score.min.js). These scripts use an `OscillatorNode` to generate a sine wave. The output signal is connected to a `DynamicsCompressorNode`, possibly to increase differences in processed audio between machines. The output of this compressor is passed to the buffer of an `OfflineAudioContext`. The script uses a hash of the sum of values from the buffer as the fingerprint.

A third 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`. 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.

[17] https://w3c.github.io/webrtc-pc/#widl-RTCPeerConnection-createOffer-Promise-RTCSessionDescription--RTCOfferOptions-options

[18] https://tools.ietf.org/html/rfc3264

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.

13. ADDITIONAL METHODOLOGY

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".

13.1 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 proceeding it (not including any additional subdomains). We use Mozilla's Public Suffix List[19] 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 the supplementaary materials Section 13.2). Throughout the paper we use the word "domain" to refer to a site's PS+1.

13.2 Landing page detection 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.
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.

13.3 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 [44, 1]:

- 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.

13.4 Detection of Fingerprinting

Javascript minification and obfuscation hinder static analysis. Minification is used to reduce the size of a file for transit. Obfuscation stores the script in one or more obfuscated strings, which are transformed and evaluated at run time using `eval` function. We find that fingerprinting and tracking scripts are frequently minified or obfuscated, hence our dynamic approach. With our detection methodology, we intercept and record access to specific Javascript objects, which is *not* affected 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 3.2, 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 3.2. 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 Deobfuscator[20] extension. We also occasionally spot check live versions of sites and scripts, falling back to the archive when there are discrepancies.

[19] https://publicsuffix.org/

[20] https://addons.mozilla.org/en-US/firefox/addon/javascript-deobfuscator/

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
dtt617kogtcso.cloudfront.net/sauce.min.js	55
685 others	*1853*
TOTAL	18283 *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	mmmmmmmmmMMMMMMMMM=llllIiiiiii‘’.
api.twisto.cz/v2/proxy/test*	1	mmmmmmmmmmlli
go.lynxbroker.de/eat_session.js	1	mimimimimimimi[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.