Automated discovery of privacy violations on the web

Steven Tyler Englehardt

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance by
the Department of
Computer Science

Adviser: Arvind Narayanan

September 2018

Abstract

Online tracking is increasingly invasive and ubiquitous. Tracking protection provided by browsers is often ineffective, while solutions based on voluntary cooperation, such as Do Not Track, haven't had meaningful adoption. Knowledgeable users may turn to anti-tracking tools, but even these more advanced solutions fail to fully protect against the techniques we study.

In this dissertation, we introduce OpenWPM, a platform we developed for flexible and modular web measurement. We've used OpenWPM to run large-scale studies leading to the discovery of numerous privacy violations across the web and in emails. These discoveries have curtailed the adoption of tracking techniques, and have informed policy debates and browser privacy decisions.

In particular, we present novel detection methods and results for persistent tracking techniques, including: device fingerprinting, cookie syncing, and cookie respawning. Our findings include sophisticated fingerprinting techniques never before measured in the wild. We've found that nearly every new API is misused by trackers for fingerprinting. The misuse is often invisible to users and publishers alike, and in many cases was not anticipated by API designers. We take a critical look at how the API design process can be changed to prevent such misuse in the future.

We also explore the industry of trackers which use PII-derived identifiers to track users across devices, and even into the offline world. To measure these techniques, we develop a novel bait technique, which allows us to spoof the presence of PII on a large number of sites. We show how trackers exfiltrate the spoofed PII through the abuse of browser features. We find that PII collection is not limited to the web—the act of viewing an email also leaks PII to trackers. Overall, about 30% of emails leak the recipient's email address to one or more third parties.

Finally, we study the ability of a passive eavesdropper to leverage tracking cookies for mass surveillance. If two web pages embed the same tracker, then the adversary

can link visits to those pages from the same user even if the user's IP address varies. We find that the adversary can reconstruct 62—73% of a typical user's browsing history.

Acknowledgements

I am grateful to my advisor, Arvind Narayanan, for always being willing to provide assistance and advice. Arvind has encouraged me to do the uncomfortable, to be confident in my work, and to not be afraid to do things differently. I owe a lot of my professional growth over the past five years to his guidance. I've learned the value of real world impact in research, and perhaps more importantly, an understanding of how to choose the right projects.

I am also grateful to my dissertation committee, Nick Feamster, Edward Felten, Jennifer Rexford, and Prateek Mittal for their assistance and advice, not only during the dissertation process, but throughout my time at Princeton.

I am incredibly lucky to have worked with such talented collaborators. I've benefited from their advice, critiques, and insights. In particular, I'd like to thank Güneş Acar, Claudia Diaz, Christian Eubank, Edward Felten, Jeffrey Han, Marc Juarez, Jonathan Mayer, Arvind Narayanan, Lukasz Olejnik, Dillion Reisman, and Peter Zimmerman. This research in this dissertation would not have been possible without their contributions.

I could always count on the members of the security group to keep life interesting! At any time of day you could jump in to an engaging conversation, whether it was working through research ideas or playing along with one of our countless nonsensical jokes. In particular, I'm thankful to Steven Goldfeder for the late-night comradery, to Harry Kalodner who was never afraid to jump in to help me struggle through a research problem, to Joe Bonneau for his assistance and encouragement, to Pete and Janee Zimmerman for never failing to bring us all together, and to Ben Burgess for teaching me that no amount of physical security is enough physical security.

My decision to pursue a career in research was the result of countless conversations and mentoring from friends and collaborators at Stevens Institute of Technology. My initial exposure to research was at the encouragement of Chris Merck, who introduced

me to the atmospheric research lab at Stevens. I owe a debt of gratitude to all of my mentors along the way, including Jeff Koskulics and Harris Kyriakou.

I simply wouldn't have made it to where I am today without my wife, Kim. She is always encouraging and optimistic, helping me to be the best version of myself. She has provided unwavering support and patience through the ups and downs of grad school, and has been a constant source of happiness.

Lastly, I am grateful to my family, who have always supported me in all my endeavours. My mother and father have taught me the value of hard and honest work. I couldn't ask for better role models in life.

The work in this dissertation was supported in part by a National Science Foundation Grant No. CNS 1526353, a grant from the Data Transparency Lab, a research grant from Mozilla, a Microsoft Excellence Scholarship for Internships and Summer Research given through Princeton's Center for Information Technology Policy, and by Amazon Web Services Cloud Credits for Research.

Contents

Bibliographic Notes

This dissertation is based on work presented in a number of academic papers and blog posts. We list these works here, along with the respective chapters that build upon them.

- **Cookies that give you Away: Evaluating the surveillance implications of web tracking [121].** Included in chapters 1, 2, 3, and 7.
- **The Web Never Forgets: Persistent tracking mechanisms in the wild [29].** Included in chapters 1, 2, 3, 4, and 5.
- **Online Tracking: A 1-million-site measurement and analysis [120].** Included in chapters 1, 2, 3, 4, 5, 7, and 8.
- **Battery Status Not Included: Assessing Privacy in Web Standards [260, 261].** Included in chapters 1, 2, and 5.
- **I never signed up for this! Privacy implications of email tracking [119].** Included in chapters 1, 2, 3, 6, and 8.
- **No boundaries: Exfiltration of personal data by session-replay scripts [116].** Included in Chapter 6.
- **No boundaries for credentials: New password leaks to Mixpanel and Session Replay Companies [117].** Included in Chapter 6.

- **No boundaries for user identities: Web trackers exploit browser login managers [28].** Included in Chapter 6.
- **Website operators are in the dark about privacy violations by third-party scripts [118].** Included in Chapter 6.

Chapter 1

Introduction

1.1 Overview

Data collection on individuals by is commonplace on the web. Nearly every website a user visits records, aggregates, and shares information about that visit with third-party entities. The ubiquitous presence of the web in modern life is due in large part to its "mash-up" design—any website can pull in and make use of content from any number of entities. Web developers can take advantage of this to easily build and monetize web applications that provide a rich user experience. However, this same "mash-up" nature has significantly reduced the cost of individual data collection and has greatly complicated efforts to protect user privacy.

A major funding mechanism for the web is behavioral advertising—that is, advertising which is targeted based on a user's personal information, interests, and past behaviors. Behavioral advertising requires a significant amount of data collection, ranging from data collected to better target an advertisement to the collection of purchase data to attribute the sale of a product to an advertiser. Another driving factor is the collection of detailed analytics data by third parties to help websites better understand and monetize their audience. There are a number of contributing

factors that enable and incentivize this type of data collection. We describe each in turn.

- *Data collection is largely invisible to users.* When a user visits a site, there is no visual indication of the amount of tracking the site contains. In fact, it's not only difficult for users to determine the how they're tracked, the websites themselves often don't know exactly which trackers they embed [118]. This is further evidenced by the existence of companies which specialize in helping websites discover unauthorized third-party embeds.[1]

- *Any third-party embedded on a site can collect user data.* While the techniques a third party can use to track users varies by the resource type and location of embedding (Section 2.1), any resource can be used to collect some user data. In the case of passive fingerprinting (Section 2.1.2), this tracking can be completely invisible—even to researchers. In some cases, such as when the embedded third-party resource is a script, the third party can invisibly pull in resources from other third parties, furthering the risk of surreptitious data collection.

- *The dynamic nature of the web makes this data collection difficult to monitor and prevent.* There may be large differences in the set of resources loaded between two visits to the same website, depending on the chain of third-party includes. In additional to privacy concerns, this has led to concerns of audience leakage and advertisement fraud.[2] In extreme cases, advertisements on the page are used to deliver malware to the user [298]. Indeed, we have had to provide

[1] OpenX provides such a service for websites (through its acquisition of Mezzobit). See `https://www.openx.com/publishers/openx-mezzobit/`.

[2] Audience leakage describes the process by which website audience data is shared with third parties without a reciprocal relationship between the third party and website [177]. As an example, consider an ad auction running on `example.com`. When the auction runs, a number of third parties may learn that the user is a subscriber of `example.com` without ever serving an ad. Advertisement fraud can take a variety of forms, but generally refers to the process by which a website falsifies performance metrics to receive undeserved advertising revenue. Examples include falsifying where an ad was displayed or to which type of user it was displayed [37].

additional information to several site owners to help them determine the origin of invasive tracking scripts exposed by our research [28].

- *Personalized advertising relies on a rich ecosystem of third parties.*[3] To display a personalized advertisement, a website will typically run an advertisement auction. This process involves a number of third parties on the supply-side of the transaction, all of which may collect information about the user and share it with a number of third parties on the buy-side of the transaction. The third parties purchasing the advertisement will use their own tracking data in combination with data provided by the supply-side parties to determine whether and how much they will pay for the advertisement slot. Once the advertisement is displayed, it may be served alongside a script from an unrelated party which performs impression verification; i.e., certifying that the advertisement was served to a human. Finally, additional parties may be embedded into the page to track "conversions", or advertisement clicks that result in a purchase. Any one of these parties has the technical capability to track the user, and in practice many will.

- *Abuse of tracking data is difficult to audit.* The open nature of the web platform provides unprecedented insight into how and by whom user data is collected. However, once tracking data is transferred to third-party servers it becomes much more difficult to audit its use. Data may be shared or aggregated between third parties without the knowledge of the user. Website privacy policies rarely provide details regarding which third parties are embedded on the page [202]. Academic projects have had some success in attributing advertisements to specific targeting criteria, but only in a narrow set of use cases [100, 194, 195]. Large scale auditing of data use in advertisements is not yet feasible. Similarly,

[3]For an overview of the parties involved in serving a programmatic advertisement, see: `https://pagefair.com/data-leakage-in-online-behavioural-advertising/`.

regulators and media pressure have succeeded in pressuring several large data collectors to provide some transparency tools [142, 330], but these tools only show a slice of all data collected and used for web tracking.

- *The increasing sophistication of the web platform helps trackers.* Browser vendors continue to refine the web platform by adding new HTML5 features; recent examples include support for virtual reality hardware [329] and complex video rendering [188]. In many cases, these new features lead to privacy vulnerabilities. Past research found trackers abusing stateful browser features to store tracking identifiers that users couldn't clear [47, 295, 304]. In our research, we show that trackers are often early adopters of these new features (Chapter 5). Until fairly recently, the standards community did not have a formal process or requirement to review the privacy impact of new web features (Section 2.3).

Ubiquitous data collection has societal risks. Web tracking and individual data collection on the web has a wide range of effects. Users often misunderstand the process by which their data is collected and used [344], and find personalized advertising creepy and invasive [222, 316]. Furthermore, when users better understand the extent to which data is collected, they feel less comfortable with targeted advertising [282, 312]. Some users report that this feeling of discomfort is enough to make them change their practices [222], showing that the data collected for targeted advertising has a chilling effect on users' actions on the web.

The data collected for targeted advertising can be abused in a number of ways. Tracking data can be used for price discrimination, the process of pricing a product or service based on the consumer's willingness to pay [249]. Price discrimination is often desirable for sellers to maximize revenue, but is disliked and viewed as unfair by the large majority of consumers [311]. While overt individual price discrimination has not been observed by past research [227, 324], researchers have seen *price steering*—

the process by which certain consumers are presented with more expensive product options as a result of their browsing history [151, 227].

In recent years personalized advertising has been used to manipulate users [27]. During the 2016 U.S. presidential election, divisive Facebook advertisements were purchased by Russian organizations and targeted to U.S. voters [165] The advertisers used Facebook's ad targeting categories to show ads to voters likely to be interested in hot-button issues, such as "gun rights". Some of these data categories are likely to have been built on top of web tracking data [43].

Web tracking data is often collected with the justification that it is "anonymous". In practice, there are a number of ways to identify the individuals from the clickstream data collected by trackers. Trackers can identify users through first-party logins (for those trackers that have a first-party presence), leaked identifiers in URLs, or through direct collection of personal information (e.g., through web surveys) [242]. Web browsing history itself is sufficiently unique and stable to serve as an identifier for a user [259]. It's often possible to identify a user's browsing history by observing just a subset of page visits—example attacks include discovering visits through a browser vulnerability [259] or recording the links a user shares on social media [305].

The ubiquitous presence of web tracking identifiers also has second-order impacts. In particular, the pervasive monitoring of internet traffic through network surveillance is strengthened by the presence of tracking identifiers [126]. As we show in Chapter 7, tracking identifiers sent in unencrypted traffic allow a network attacker, like the NSA, to cluster page visits and link those visits to real identities. This enabled a number of attacks. First, browsing history itself could be the information of interest, as in the NSA plan to discredit "radicals" based on browsing behavior, such as pornography [15]. Second, further sensitive information might be found in unencrypted web content such as preferences, purchase history, address, etc. Finally, it can enable active attacks such as delivering malware to a targeted user [135, 296].

Web Privacy Measurement—observing websites and services to detect, characterize and quantify privacy-impacting behaviors—has proved tremendously influential. The majority of browser vendors do little to prevent user tracking, and the protections that users can opt-in to address only part of the problem. Efforts by the community to self-regulate, including The Platform for Privacy Preferences (P3P) and Do Not Track (DNT), have been similarly ineffective [86,148,197]. Instead, web privacy measurement has repeatedly forced companies to improve their privacy practices due to public pressure, press coverage, and regulatory action [101, 127, 293, 317].

Web measurement is hard because the web is not designed for automation or instrumentation. Selenium,[4] 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 browser extension, in-page instrumentation, and a disk state monitor.

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 have made our data and measurement platform, OpenWPM, available publicly. We expect that measurement will be useful to developers of privacy tools, to regulators and policy makers, journalists, and many others.

[4] `http://www.seleniumhq.org/`

1.2 Contributions

The purpose of this dissertation is to advance the tools and methods of the web privacy measurement community, as well as to advance the understanding of the invasive techniques used to track users. To that end, we developed OpenWPM, an open source platform for running privacy studies across millions of websites (Chapter 3). Over the course of its development OpenWPM has been used in 22 academic studies, including several of our own (Section 3.1.4). In this dissertation, we make extensive use of OpenWPM to quantify, understand, and uncover the ways users are tracked.

We make contributions to several distinct areas of study: web tracking (Chapter 4, Chapter 5, and Chapter 6), government surveillance (Chapter 7), and the standardization process for the web platform (Section 5.2). In each area, we show how large-scale measurement can be used to discover new privacy vulnerabilities, quantify known problems, and evaluate defenses. We show that the techniques and tools we developed for measuring web tracking can be used to evaluate the threat of government surveillance. Similarly, we show the problems that arise when designing web APIs in the absense of usage data, and proposal that abuse measurement become a core component of the standardization process.

Our tracking measurements fall into three broad categories: (1) stateful tracking (Chapter 4), (2) stateless tracking through device fingerprinting (Chapter 5), and (3) tracking with identifiers which are based on the user's personally identifiable information (Chapter 6). We note several high-level observations:

- Tracking is ever-present, both in emails and on the web. In most cases—93% of websites and 85% of emails—a user will encounter at least one third party when reading a website or opening an email.

- The larger trackers by-and-large use simpler tracking techniques that offer users more control. The more invasive techniques, such as device fingerprinting and

PII-based tracking are used by trackers with a smaller presence. However, we show that the interconnected nature of tracking can amplify the practices of these bad actors.

- Privacy extensions which block resources using blacklists consistently offer users the best protection from tracking. However, even these extensions appear to miss the lesser known trackers or trackers using lesser known techniques. We show that automated web measurement can detect these missed trackers and complement current blacklists.

- Web tracking has second-order impacts beyond the immediate harms of data collection by trackers. We show how the widespread tracking of users helps network surveillance agencies record a user's browsing history.

1.2.1 OpenWPM: a web measurement platform

A core contribution of this dissertation is the development of OpenWPM, a generic web measurement platform that supports large-scale measurements with a realistic browser (Chapter 3). Realistic web measurement is hard; websites often deploy bot detection techniques that can detect measurement instances and treat them differently than real humans. By building upon the Firefox web browser, OpenWPM makes to possible to run web measurement studies on millions of websites with a consumer browser that supports all of the modern web platform features. This means it's more difficult to detect an OpenWPM instance as a bot. OpenWPM includes deep instrumentation to record much of the data a researcher might need for a privacy measurement study, including: all HTTP request and response data, records of when and by whom browser storage was accessed, and probes that monitor many of the Javascript APIs used for device fingerprinting[5] OpenWPM's design is robust

[5]For a description of device fingerprinting see Section 2.1.2.

to browser crashes and failures, and saves data in a standardized output format to facilitate data sharing and repeated analyses.

Alongside OpenWPM we introduce a set of methods that are fundamental to many measurement tasks: the classification of third-party content (Section 3.2.1) and trackers (Section 3.2.2) on the web and in emails, several methods of generating browsing models for measurements (Section 3.2.3), and analysis methods to analyze tracking scripts on the web (Section 3.2.6). In addition, we propose several advanced techniques that are useful for measuring web trackers, including: detecting unique identifiers stored in browser storage (Section 3.2.4), discovering leaks of personal data in HTTP traffic, even when that data is encoded (Section 3.2.5), and a "bait technique" to spoof that personal information is present on real websites (Section 6.2.2). These methods are robust to changes in the web platform or protocols, as they can easily be adapted for new storage techniques or communication channels.

1.2.2 The state of web tracking

We analyzed tracking on the top 1 million websites (Chapter 4). This represents one of the largest automated measurements of web tracking performed with a real consumer browser. In particular, we showed that web tracking has a "long tail" of trackers, but only 123 of which were present on more than 1% of sites. The only third-party organizations present on more than 10% of sites at the time were some of the largest companies in the world—Google, Facebook, Twitter, Amazon, AdNexus, and Oracle. This analysis alone doesn't fully capture a third party's ability to track a user, since users spend more time on the top sites. To address that, we also rank trackers by their *prominence*, a metric we propose for weighting a third party by the popularity of the sites which embed it. We show that the prominence metric works well when comparing tracker popularity across measurements. We also found that

tracking varied significantly across categories of sites, with news sites having the most trackers.

In addition, we performed the first large-scale measurements of cookie respawning and cookie synchronization. Cookie respawning is the process by which trackers store identifiers in multiple locations in the browser and use this information to "respawn" HTTP cookies if they are cleared.[6] We developed a technique to automate and parallelize the detection of respawned cookies, allowing us to measure the practice on 10,000 sites—a scale much larger than previous studies (100 [47] and 600 [223] sites). We found that cookie respawning was still present on many of the top sites in China and Russia. Cookie syncing is the process by which trackers can bypass the Same Origin Policy (SOP) to share identifiers with each other.[7] We show that cookie syncing was widespread on the top sites and that nearly all of the top third parties cookie sync. More importantly, we examine how data sharing enabled by cookie syncing could allow hundred of third parties to learn about a large portion of a user's browsing history.

Turning to tracking defenses, we evaluated the reduction in tracking caused by built-in browser defenses, tracker blockers available as browser extensions, and the advertising industry's own opt-outs. We show that third-party cookie blocking was very effective at blocking cookie-based tracking and reduced the number of third parties per site by nearly 29% (from an average of 17.7 third parties per site to 12.6). We found that the blocklist-based Ghostery browser extension had a much more significant impact, reducing the average number of third parties per site to just 3.3. We also found that industry opt-outs resulted in a modest reduction in tracking cookies, but had a relatively minor impact on cookie syncing.

[6] We define cookie respawning in more detail in Section 2.1.1.

[7] We define cookie syncing in more detail in Section 2.1.3.

1.2.3 Measuring device fingerprinting

In addition to tracking with cookies, trackers can attempt to identify users by recording detailed information about their devices. The process is known as device fingerprinting[8]; trackers will typically produce a fingerprint by abusing web platform features to extract identifying device information, such as the operating system version or graphics hardware stack.

We make three contributions in Chapter 5: we develop a set of semi-automated techniques for detecting fingerprinting, we measure the use of fingerprinting on the top 1 million sites, and perform a case study of the development of the Battery Status API and use it to distill recommendations for mitigating fingerprinting during the standardization process.

We performed the first automated measurements of several advanced fingerprinting techniques, including those which used the Canvas, WebRTC, and Battery APIs. During our investigations we discovered a new fingerprinting technique that uses the Audio API, and likewise performed the first measurement of its use in the wild. Overall we found that fingerprinting is more common on popular sites, but is more often performed by less popular trackers. Fingerprinting occurred on far fewer sites than stateful tracking; even on the top 1,000 sites, where we observed fingerprinting most frequently, it occurred on just 5% of sites.

Device fingerprinting is difficult to defend against—browser APIs that have already shipped can't be changed without web compatibility, and it's impractical to change the properties of a device. Instead, users can install tracking blocking tools to block fingerprinting scripts entirely. We measured how well common tracker blocklists were able to detect the fingerprinting scripts we discovered. We found that the block-

[8]We define device fingerprinting in more detail in Section 2.1.2

lists[9] detected better-known techniques (like canvas fingerprinting) more frequently, but missed lesser known techniques (like WebRTC) and less popular scripts.

While previous web standards rarely addressed fingerprinting, that has started to change in recent years following formal recommendations for privacy assessments by standards bodies [95, 102, 108]. We perform a case study of the standardization of the Battery Status API, which was ultimately removed from several browser engines following the discovery of privacy vulnerabilities. We show that the API was indeed primarily used by trackers and provide several recommendations for future standards work. These include: requiring a privacy review of implementations, auditing API use in the wild after deployment, and avoiding over-specification of mitigations.

1.2.4 PII collection and use by trackers

In Chapter 6 we study the collection of PII for the purposes of cross-site tracking or website analytics. Unlike stateful tracking (Chapter 4) or fingerprinting (Chapter 5), PII-based identifiers are not tied to a device. Instead, the information collected by trackers can be used to follow users across devices and even link data collected online to data collected from offline sources. Similarly, the personal data collected by third parties—even those which don't use it for cross-site tracking—further exposes users to potential misuse and data breaches.

We measure PII collection both in emails and on the web. We contribute new methods to detect PII-based tracking, as well as new measurement results. Overall, our results bring to light two new trends in web tracking: the collection of hashed email addresses for the purposes of tracking and the increasingly invasive practices of tracking and analytics scripts embedded directly into websites.

PII leakage in emails. Email tracking is pervasive. We found that 85% of emails in our corpus contain embedded third-party content, and 70% contained resources

[9] We tested the Disconnect list and a combination of EasyList and EasyPrivacy.

categorized as trackers by popular tracking-protection lists. There were an average of 5.2 and a median of 2 third parties per email which embedded any third-party content, and nearly 900 third parties contacted at least once. But the top ones are familiar: Google-owned third parties (Doubleclick, Google APIs, etc.) were present in one-third of emails.

We simulated users viewing emails in a full-fledged email client (Section 6.1.3). We found that about 29% of emails leaked the user's email address to at least one third party, and about 19% of senders sent at least one email that had such a leak. The majority of these leaks (62%) were intentional, based on our heuristics. Tracking protection was helpful, but not perfect: it reduced the number of email leaks by 87%. Interestingly, the top trackers that received these leaked emails were different from the top web trackers. We present a case study of the most-common tracker at the time of measurement, LiveIntent (Section 6.1.3).

PII leakage on the web. Websites commonly embed third-party scripts into the main context of the page. Scripts that are embedded in this way are not restricted by the Same Origin Policy (SOP) from accessing any content in the main frame. We examine two distinct vulnerabilities that highlight the risk of this practice: the exfiltration of content from the DOM and the abuse of browser login managers. To measure both vulnerabilities we introduce a *bait technique*, in which we inject sensitive information into different parts of the page or browser and check to see if any scripts exfiltrate that data.

We discovered 28 different third parties scraping information from the DOM to provide analytics, translation, advertising, and support services. Many of the services offered some combination of manual and automated redaction to prevent the collection of sensitive information. We conducted a case study of "session replay" scripts, which collect information from the DOM to produce recordings of how users interact with websites. We found that these redaction features were often incomplete or went

unused by first parties. We also discovered two parties abusing the browser login manager to extract user email addresses. These scripts injected invisible forms into the page, waited for the browser's autofill feature to fill them, and then extracted usernames from the autofilled forms.

1.2.5 The surveillance implications of web tracking

In Chapter 7 we examine the use of web tracking data in government surveillance. Our contributions are both conceptual and empirical. First, we identify and formalize a new privacy threat from packet sniffing. While the technique of utilizing cookies to target users is well understood (Chapter 4), we formulate the attack concretely in terms of the following steps: (1) automatically classifying cookies as unique identifiers (2) using multiple ID cookies to make *transitive* links between site visits that allow us to *cluster* HTTP traffic, (3) geographically tagging the flow of traffic, and (4) inferring real-world identity from HTTP request and response bodies. We believe this attack to be the strongest known way for a passive adversary to utilize web traffic for surveillance.

Next, we rigorously evaluate the above attack model using a novel methodology that combines web measurement, network measurement, a user model, and an adversary model (Section 7.2). We simulate realistic browsing behavior and measure the actual cookie ecosystem. This requires nuanced techniques along several fronts: (1) the use of OpenWPM (Chapter 3) to closely simulate real users (2) a model of browsing history derived from real user behavior, and (3) network measurements to help verify the robustness of geolocation data.

We build a browsing model based on search data from real users who made between 0 - 300 web searches over a 2 - 3 month period, and simulate that browsing in a June 2014 web crawl. We consider users located in several possible countries. For each such set of visits, we perform clustering using the method described above and find

the "giant connected component." For non-U.S. users, we consider an adversary with wiretaps in the target user's country as well as one with wiretaps in the U.S. On a high-level, our results are as follows:

- For a U.S. user, over 73% of visits were in the connected component. The clustering effect was extremely strong and robust to differences in the models of browsing behavior. Clustering even occurred when the adversary was restricted to a small, random subset of the user's requests.

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

- Of the Alexa Top 50 U.S. sites, 56% transmitted some form of identifying information in plaintext once a user logged in, whether first name, first and last name, username, or email address. The majority of these (42% of websites overall) transmitted *unique* identifiers (username or email address) in the clear.

- Built-in browser protections were able to reduce but not fully mitigate the attack. The most effective blocking solution, the Ghostery browser extension, still allowed 24.2% of a user's traffic to be clustered, while alternative solutions had far less of an impact.

1.3 Structure

This dissertation introduces automated web privacy measurement methods and tools, and uses them to study three classes of web tracking: stateful tracking (Chapter 4), stateless tracking (Chapter 5), and PII-based tracking (Chapter 6). The dissertation is

structured as follows: Chapter 2 provides the background and related work, Chapter 3 introduces OpenWPM and several measurement methods used throughout our work, Chapter 4, Chapter 5, and Chapter 6 present web tracking measurement results, and in Chapter 7 we examine the externalities of web tracking.

Chapter 2

Background and related work

Third-party web tracking has been studied since the early 2000s. A large body of research details the evolution of tracking techniques and records the growth of tracking on the web. Our work builds on this foundation, making contributions to several distinct areas of study: web tracking measurement (Section 2.1), government surveillance (Section 2.2), and the web standardization process (Section 2.3).

2.1 Third-party web 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`[1] 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, either directly or through the `Referer` header.

[1] The `Referer` header is a misspelling of referrer.

Third-party tracking has grown tremendously in prevalence and complexity since the introduction of HTTP cookies in the early 1990s [185]. Web tracking has expanded from simple HTTP cookies to include more persistent tracking techniques to "respawn" or re-instantiate HTTP cookies through Flash cookies [295], cache E-Tags, and other browser storage locations [47]. Overall, tracking is moving from stateful to stateless techniques: device fingerprinting attempts to identify users by a combination of the device's properties [110, 193], and trackers derive identifiers from a user's PII [93]. Such techniques have advanced quickly [130,232,256], and are now widespread on the web [30, 65, 246].

Our work provides an detailed view of web tracking, but it does not examine the many other ways in which trackers can collect information about users. Most closely related is tracking in mobile applications, which is technically similar and often performed by the same companies [320]. As with web tracking, mobile tracking is pervasive. Razaghpanah et al. find a consolidated mobile app tracking ecosystem: Google and Facebook are present in 73% and 31% of the 14,599 apps they study, and they find a long tail of hundreds of additional trackers [275]. These results are strikingly similar to the distribution of trackers we measure on the top websites (Section 4.1.3). Related research has found that mobile applications commonly leak user data [115, 348]—while free applications leak data more often, paid applications aren't much better [291]. Companies have also been found to collect tracking data through other consumer products, including smart TVs [132] and Internet of Things (IoT) devices [89]. More broadly, users data is collected through the use of store loyalty cards [79], online surveys [32], and public records [31].

Web tracking ostensibly exists to support behavioral advertisement on the web; advertisers can use a user's browsing history and other personal data to show them ads more relevant to their interests. **The use of behavioral targeting by advertisers**

has grown significantly in recent years [81, 203, 214]. A 2015 study found that 26-62% of ads shown to their measurement personas used behavioral targeting [81].

Despite strong growth in the use of behavioral targeting, there has been a lack of research showing that the returns from targeted advertising alone justify the increased data collection. Past research that suggests behaviorally targeted advertising is effective [56, 140, 342] has been questioned by later work for being unrepresentative [220] and for failing to correct for selection biases [125]. Lambrecht et al. find that retargeted ads which use individual data generally perform worse than those that use brand information, suggesting that detailed individual data collection may not be necessary [187]. Schmeiser finds that special interest websites—those which cater to a specific interest of the user—may actually lose revenue by participating in targeted advertising networks [286].

Web tracking data has many uses beyond behavioral advertising. It has been used to calculate credit scores [26, 89], to target political messaging [27], and to predict an individual's health risks for insurance purposes [88]. China has experimented with a "Social Credit System" system, which goes beyond typical financial or health risk assessment systems to rate an individual's overall "trustworthiness" [20]. Alibaba, one of eight companies chosen to implement a prototype of the system, uses behavioral data in their calculation of the score [25].

2.1.1 Stateful web tracking

Third-party trackers can track users across websites by storing a unique identifier on the user's device, a process known as "stateful tracking". Modern web browsers provide several APIs that can be used to store persistent information, including HTTP cookies [53], the Web Storage API (i.e., `localStorage` and `sessionStorage`) [224], the IndexedDB API [328], and Flash Local Shared Objects (LSOs) [35]. In addition to using APIs designed for storage, web sites can also use indirect means to store

an identifier — any API that persists data on the user's device is a candidate for such abuse. For example, trackers can encode identifiers in HTTP Strict Transport Security (HSTS) flags [304], or use the web browser's cache to store an identifier via an E-Tag [47]. A tracker's access to each of these APIs depends on several conditions:

1. *Whether the tracker has script access to the page.* Passive resources, such as images and HTML videos, are not able to access the Web Storage or IndexedDB APIs, as these APIs only have script interfaces. HTTP cookies and cache E-Tags are set and retrieved by HTTP requests and responses, and thus can still be used by passive resources.

2. *The frame in which the tracker is loaded.* Web pages consist of a series of nested rendering contexts, each with its own *origin*, often defined as the scheme, host, and port from which the main document of that context was loaded.[2] Each storage location has an access policy that may depend on the origin of the resource itself (e.g., E-Tags), the origin of the context in which the resource was loaded (e.g., the Web Storage API), or both (e.g., HTTP cookies) [162].

3. *The set of access policies applied by the browser.* Some browsers restrict access to certain APIs (such as the IndexedDB API) when the user is in a privacy-focused browsing mode [36]. Other browsers apply more restrictive policies to all browsing modes, such as Safari's third-party cookie blocking [339].

Most commonly, trackers will use HTTP cookies to store their identifiers. The HTTP cookie interface was the first stateful interface added to the browser [185], and is accessible through both an HTTP interface and a Javascript interface. A web server can use the `Set-Cookie` header in an HTTP response to store arbitrary

[2]The modern web's origin concept [54] is more complex than a simple scheme, host, and port definition. As an example, a sandboxed iframe is given a unique origin, regardless of the origin of the resource [337]. Some browsers have extended the web's origin to include additional keys, such as the host of the top-level frame [326].

data in a cookie keyed by its domain, and will receive the contents of that cookie in a `Cookie` header for each subsequent HTTP request to its domain. Scripts can also use the `document.cookie` interface to access the cookies associated with the rendering context in which they're embedded. As of January 2018, the majority of modern browsers allow both third-party and first-party content to set and retrieve cookies during each page visit—Safari being a notable exception [339]. In response to privacy concerns of third-party tracking via cookies, browser vendors have given users additional controls over HTTP cookies: most built-in private browsing modes use a temporary set of cookies distinct from those used in regular modes [36], several browsers provide options to clear cookies on shutdown [268], and users can block third-party cookies [306].

Web tracking has expanded from simple HTTP cookies to include techniques that provide less user control. Trackers can also store identifiers in other locations within the browser by using the browsers own storage APIs, or by abusing other stateful features. Browser vendors have historically failed to provide the same level of user control over other stateful APIs as they have provided for HTTP cookies [55,66,68,69], which causes data stored in these alternative locations to persist after a user tries to clear their cookies. Identifiers stored elsewhere in the browser can be used to "respawn" HTTP cookies, i.e., recreate HTTP tracking cookies after they have been cleared. An example of HTTP cookie respawning is given in Figure 2.1.

As an example of cookie respawning, Figure 2.1 depicts the stages of respawning by Local Shared Objects (LSOs), also known as Flash cookies. Whenever a user visits a site that respawns cookies, the site issues an ID and stores it in multiple storage mechanisms, including cookies, LSOs, and localStorage. In Figure 2.1a, the value *123* is stored in both HTTP and Flash cookies. When the user removes her HTTP cookie (Figure 2.1b), the website places a cookie with the same value (123) by reading the ID value from a Flash cookie that the user may fail to remove (Figure 2.1c).

(a) (b) (c)

Figure 2.1: Respawning HTTP cookies by Flash evercookies: (a) the webpage stores an HTTP and a Flash cookie (LSO), (b) the user removes the HTTP cookie, (c) the webpage respawns the HTTP cookie by copying the value from the Flash cookie.

In 2010, Samy Kamkar demonstrated the "Evercookie," a resilient tracking mechanism that utilizes multiple storage vectors including Flash cookies, localStorage, sessionStorage and ETags [163]. Kamkar employed a variety of novel techniques, such as printing ID strings into a canvas image which is then force-cached and read from the cached image on subsequent visits. In addition to those supported by the evercookie library, a number of other APIs were also found to be vulnerable to use in cookie respawning. These include HTTP Strict Transport Security (HSTS) flags [157], HTTP Public Key Pinning (HPKP) flags [123], cached intermediate certificate authorities [73], and DNS cache [128]. Bujlow et al. [75] and Mayer and Mitchell [220] provide a detailed survey of web tracking methods.

Measurement studies. Krishnarmurthy and Wills [181] 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 [30, 39, 122, 159, 220, 280]. 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 [198]. Fruchter et al. studied geographic variations in tracking [131]. More recently, Libert studied third-party HTTP requests on the top 1 million sites [201], 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.

In Chapter 4 we study stateful web tracking on the top 1 million websites. We find a long but thin tail of web trackers, showing that only 4 companies are present on 10% or more of sites (Section 4.1.3), show that news sites have the most trackers (Section 4.1.5), and find that blocking tools miss less popular trackers (Section 4.1.6).

A 2009 study by Soltani et al. showed the abuse of Flash cookies for regenerating previously removed HTTP cookies [295]. They found that 54 of the 100 most popular sites (rated by Quantcast) stored Flash cookies, of which 41 had matching content with regular cookies. Soltani et al. then analyzed respawning and found that several sites, including `aol.com`, `about.com` and `hulu.com`, regenerated previously removed HTTP cookies using Flash cookies. A follow up study in 2011 found that sites use ETags and HTML5 localStorage API to respawn cookies [47]. These discoveries led to media backlash [199, 229] and legal settlements [101, 293] 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 [223,280]. Our follow-up measurement in Section 4.2 agrees with these findings. In a recent study, Sorensen analyzed the use of cache as a persistent storage mechanism and found several instances of HTTP cookies respawned from cached page content [299].

There are various client-side tools to block, limit or visualize third-party tracking. These are too numerous to list exhaustively, but a sampling include Adblock Plus, Ghostery, ShareMeNot [8], Lightbeam, and TrackingObserver [9]. Studies that have quantified the privacy effect of these tools include [49, 137, 215, 220, 225]. We examine the performance of several popular tracking protection tools in Section 4.1.6.

2.1.2 Stateless tracking

Stateless tracking is a persistent tracking technique that does not require a tracker to set any state in the user's browser. Instead, trackers attempt to identify users by a combination of the device's properties, both those available from network requests (passive fingerprinting) and those which can be queried through Javascript or Flash (active fingerprinting). Unlike the stateful tracking techniques discussed in Section 2.1.1, users have few options to control stateless tracking identifiers.

Tracking with network identifiers

Every network request to a third-party server will include information that can identify the user – namely IP address and the `User-Agent` string. The stability and uniqueness of an IP address depends on the network configuration; multiple users may share an IP address due to residential NATs [209] or carrier-grade NATs [278], and short address renewal policies may lead to a churn in addresses [266].

Yen et al. show how IP, cookies and usernames can be combined to track devices reliably even when any one of these identifiers may be individually unreliable [345]. While Yen et al. [345] identify NATs as a possible source of error in tracking users, more recent work has shown that NATs can be useful in linking multiple devices belonging to the same user [351]. We further discuss cross-device tracking in Section 2.1.5.

Trackers may also be able to take advantage of "enriched" headers that contain tracking IDs injected by ISPs [217]. Header injection is especially common on mobile networks, where carriers have been found to inject device identifiers (such as an IMEI), operator-generated advertising identifiers, and the internal IP address of the subscriber [319].

Device fingerprinting

In 2009 Mayer studied several active fingerprinting sources[3], including device properties available from the `navigator`, `screen`, `Plugin`, and `MimeType` APIs. He found a mostly unique set of fingerprints across about 1,300 clients [219]. In 2010 Eckersley examined 470,161 device fingerprints collected on the EFF's Panopticlick test page [7]. In additional to the techniques used by Mayer, the Panopticlick device fingerprints included system fonts, HTTP `Accept` headers, and the device's `User-Agent` header. Eckersley found that about 84% of the devices tested had a unique fingerprint. More recently, a study by Laperdrix et al. [193] of over 100,000 fingerprints found that 90% of desktops and 81% of mobile devices tested had unique fingerprints.

New fingerprinting vectors are continually discovered, including: the Canvas and WebGL APIs [232], the Battery Status API [256], font metrics [130], performance metrics [231], the JavaScript engine [239], the rendering engine [315], clock skew [168], HTTP/2 features [289], device motion sensors [63, 98, 99, 104], protocol handlers [6], and mouse scroll wheel events [22]. Boda et al. [62] and Cao et al. [78] showed how a combination of these features can be used to track users across browsers on the same device. Alaca and van Oorschot studied the use of device fingerprinting for web authentication [38]. Finally, Unger et al. [315], studied the potential use of device fingerprinting as a defense mechanism against HTTP session hijacking.

Defenses. Device fingerprinting has proven difficult to mitigate; users can't reasonably be expected to change their own device's fingerprint. By measuring repeat visits to Panopticlick, Eckersley found that device fingerprints change over time, with 37.4% of repeat visitors exhibiting more than one fingerprint [7]. However, he was able to correctly re-identify 65% of devices using a fairly simple string similarity heuristic. The Tor Browser provides broad fingerprinting resistance by disabling or normalizing APIs which can be used for fingerprinting, but can also lead to site breakage [4].

[3]Mayer refers to browser fingerprinting as device "quirkiness".

Several researchers have explored alternative approaches to mitigate fingerprinting. Besson et al. [58] examined the theoretical boundaries of fingerprinting defenses using Quantified Information Flow. Boda et al. developed FireGloves, a Firefox extension that fakes several common navigator and screen features and randomizes element properties used for font discovery [61, 62]. Nikiforakis et al. refine this idea, developing Privaricator, which uses a set of randomization policies to minimize the linkability of a device across sites while also minimizing site breakage [245]. By randomizing device features the user will have a different fingerprinting for each page visit, making it harder to use their fingerprint to track them across sites. Using an Chromium implementation of Privaricator, Nikiforakis et al. show that randomization is an effective fingerprinting defense with relatively low site breakage. Laperdrix et al. follow a similar approach in FPRandom, a modified version of Firefox that adds randomization to the Canvas and AudioContext APIs [191]. Laperdrix et al. also propose randomizing device features at the platform level and introduce Blink, a prototype system to automatically provides this randomization through virtual machines [192].

Measurement studies. Two studies measured the prevalence of different fingerprinting mechanisms and evaluated existing countermeasures [30, 246]. Nikiforakis et al. studied three previously known fingerprinting companies and found 40 such sites among the top 10K sites employing practices such as font probing and the use of Flash to circumvent proxy servers [246]. Acar et al. found that 404 sites in the top million deployed JavaScript-based fingerprinting and 145 sites of the top 10,000 sites leveraged Flash-based fingerprinting [30]. In Chapter 5 we examine fingerprinting on the top websites. We perform the first large scale measurements of the use of several HTML5 APIs for fingerprinting, including: Canvas (Section 5.1.2), WebRTC (Section 5.1.4), Audio (Section 5.1.5), and Battery (Section 5.1.6).

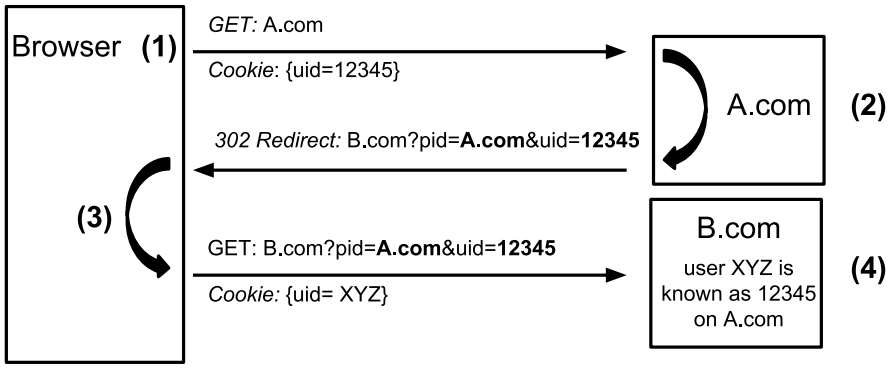


Figure 2.2: A diagram of the cookie syncing process.
(1) The user's browser makes a request for embedded content from `A.com`. This request includes the user's cookie for `A.com`, with value `user_id=12345`.
(2) A.com returns a 302 redirect to `B.com`, embedding the cookie value it uses to track the user.
(3) The user's browser makes a new request to `B.com`, which includes both `A.com`'s name and cookie value embedded in the query string. Since this request is to a new server, the user's browser sends a different cookie.
(4) `B.com` can link the information in the query string to the user's cookie. Thus, `B.com` learns that the user it refers to as `XYZ` is known as `12345` by `A.com`.

2.1.3 Cookie syncing

Cookie synchronization or cookie syncing is the practice of tracker domains passing pseudonymous IDs associated with a given user, typically stored in cookies, amongst each other. Figure 2.2 shows one possible implementation of cookie syncing. Domain A passes an ID to domain B by making a request to a URL hosted by domain B which contains the ID as a parameter string. According to Google's developer guide to cookie syncing (which they call cookie matching), cookie syncing provides a means for domains to share cookie values, given the restriction that sites can't read each other cookies, in order to better facilitate targeting and real-time bidding [17].

Olejnik et al. studied cookie syncing, and found that over 100 cookie syncing events happen on the top 100 sites [262]. The authors consider cookie synchronization both as a means of detecting business relationships between different third-parties but also as a means of determining to what degree user data may flow between parties, primarily through real-time bidding. We provide further insight into the use

of cookie syncing in the wild, presenting large-scale usage measurements (Section 4.3) and examining the implications of ubiquitous cookie syncing (Section 4.3.4).

2.1.4 Personally Identifiable Information (PII) leakage

The ability of trackers to compile information about users is further aided by PII leaks from first parties to third parties. PII leakage is often inadvertent: a user's email address or user ID is be embedded in a publisher URL (e.g., `example.com/profile?user=userID`) and ends up leaking to a number of third-party servers via the `Referer` header. `Referer` header leaks are amplified by the availability of the leaking URI via the `document.referrer` API on subsequent page load or in embedded iframes. PII leakage can also be intentional, where a first party shares information with a third party for business purposes. PII leaks can allow trackers to attach identities to pseudonymous browsing histories.

In the past, PII leakage was mostly in the hands of first parties; accidental leakage could only be prevented if a first party took care to ensure no user information was included in their site's URLs. However, the recent `Referrer-Policy` standard has made it possible for browser vendors, extension authors, and sites to control the scope of the `Referer` header sent to third-party resources [112]. Policies can be specified to strip `Referer` headers down to the URL's origin or remove it entirely for cross-origin resources. In January 2018 Mozilla announced that it will apply a default origin-only `Referrer-Policy` starting in the Private Browsing mode of Firefox 59 [97].

Measurement studies. Early work by Krishnamurthy and Wills found that both intentional and unintentional PII leakage were common in social networks [183, 184]. In follow-up studies, Krishnamurthy et al. [180] and Mayer [216] showed that leakage is common across the web—around half of the websites studied leaked user identifiers to a third party. Mayer found PII leaks to as many as 31 separate third

parties on a single site, and discovered an instance of dating profile intentionally leaking profile data to two major data brokers [216].

More recent work includes detection of PII leakage to third parties in smartphone apps [277, 318], PII leakage in contact forms [302], and data leakage due to browser extensions [303]. In our work we discover PII exfiltration within emails (Section 6.1), and by tracking (Section 6.2) and analytics (Section 6.2.4) scripts on the web.

Obfuscated PII. The common problem faced by authors of past work (and by us) is that PII may be obfuscated before collection. When the data collection is crowdsourced [277, 318] rather than automated, there is the further complication that the strings that constitute PII are not specified by the researcher and thus not known in advance. On the other hand, crowdsourced data collection allows obtaining numerous instances of each type of leak, which might make detection easier.

Early work recognizes the possibility of obfuscated PII, but accepts it as a limitation [183, 184, 216]. Various approaches are taken in the work that follows. Ren et al. employ heuristics for splitting fields in network traffic and detecting likely keys; they then apply machine learning to discriminate between PII and other fields [277]. Starov et al. apply differential testing, that is, varying the PII entered into the system and detecting the resulting changes in information flows [302]. Brookman et al. [65] and Starov et al. [303] test combinations of encodings and/or hashes, which is most similar to the approach we take in Chapter 6.

2.1.5 Cross-device tracking

Stateful tracking (Section 2.1.1) and device fingerprinting (Section 2.1.2) are limited to tracking a user on a single device. In practice, a user's browsing can be spread across multiple desktops, laptops, and/or mobile devices. According to a January 2018 Pew study, 77% of American adults own a smartphone, which surpassed the percentage of desktop and laptop users (at 73%) [83]. Trackers wishing to follow

users across multiple devices have developed deterministic and probabilistic techniques collectively referred to as "cross-device tracking".

A user's identity is a major deterministic cross-device tracking signal. Devices are tied to a persistent user-derived identifier—a name, an email address, or a username—that can be used to link devices as belonging to the same user. Trackers with a first-party presence, such as Facebook or Google, can easily track a user across devices by linking new devices to a user's profile once the user authenticates to one of the tracker's sites or applications. Trackers without a first-party presence must rely on indirect methods of obtaining identifying information. We present several examples of surreptitious exfiltration of personal data in Chapter 6. Brookman et al. found data sharing of this type on 16 of the top 100 sites [65].

Side channels can also be used to link devices in close physical proximity. The prototypical example involves ultrasonic beacons. One of the user's devices emits a unique ultrasonic beacon, and another device in the immediate area records the beacon and uses the encoded identifier to link the two together. Arp et al. showed that modern electronic devices can reliably transmit and receive near-ultrasonic signals, and found that humans subjects were unable to reliably detect the beacons in the presence of background audio [46]. In addition, Mavroudis et al. [213] and Arp [46] show that ultrasonic tracking has been deployed on the Android app ecosystem via the *SilverPush* library.

In the absence of deterministic signals, trackers can use a number of probabilistic features to link devices. Zimmeck et al. examine relevant patent and industry documents and find that IP address and browsing history are often used to link devices [351]. The long tail of user's browsing history is often enough to uniquely identify them [259], and devices connected on residential networks often share the same IP address [209]. By creating their own cross-device tracking dataset, Zimmeck

et al. show that even these basic industry approaches can achieve over 80% accuracy, with high precision and recall [351].

If a tracker lacks the data necessary to track a user across devices, they can instead cookie sync (Section 2.1.3) with another tracker who provides cross-device tracking as a service. In Chapter 6 we examine and measure the tracking techniques used by several companies which offer such services.

2.1.6 Tracking in emails

Email tracking is possible because modern graphical email clients allow rendering a subset of HTML. JavaScript is invariably stripped, but embedded images and stylesheets are allowed. Compared to web tracking, email tracking does not use fingerprinting (Section 2.1.2) because of the absence of JavaScript. On the other hand, email readily provides a unique, persistent, real-world identifier, namely the email address.

An email's external resources are downloaded and rendered by the email client when the user views the email (unless they are proxied by the user's email server). Crucially, many email clients, and almost all web browsers, in the case of webmail, send third-party cookies with these requests, allowing linking to web profiles. The email address is leaked by being encoded as a parameter into these third-party URLs.

When link which point to the sender's website are clicked, the resulting leaks are outside the control of the email client or the email server. Even if the link doesn't contain any identifier, the web browser that opens the link will send the user's cookie with the request. The website can then link the cookie to the user's email address; this link may have been established when the user provided her email address to the sender via a web form. Finally, the sender can pass on the email address—and other personally identifiable information (PII), if available—to embedded third parties using methods such as redirects and referrer headers.

The literature on email security and privacy has focused on authentication of emails and the privacy of email *contents*. For example, Durumeric et al. found that the long tail of SMTP servers largely fail to deploy encryption and authentication, leaving users vulnerable to downgrade attacks, which are widespread in the wild [109]. Holz et al. also found that email is poorly secured in transit, often due to configuration errors [158]. In Section 6.1 we study an orthogonal problem. Securing email in transit will not defend against email tracking, nor will email tracking defenses provide security for emails in transit.

Closest to our work are two papers by Fabian et al. [124] and Bender et al. [57]. Fabian et al. create a detection criteria for email tracking pixels and links based on URL structure and HTML attributes. They find that email tracking is present across nearly all of the 600 German newsletters studied [124]. In a follow-up study, Bender et al. make the observation that many emails contain tracking pixels from companies unrelated to the newsletter or mailing list manager—an effect which is most pronounced in the United States [57]. In Section 6.1 we perform an in-depth study of email trackers and examine their close integration with web tracking.

2.2 The role of web tracking in government surveillance

The body of research described in Section 2.1 helps us understand what trackers themselves can learn, it does not address what an *eavesdropper* can by watching identifier leaks on the web. The latter is influenced by many additional factors including geographic location of the trackers and the adoption of HTTPS by websites.

In recent years, a combination of technical research, leaks, and declassifications has provided unprecedented transparency into Internet surveillance by governments. Some nations, such as Iran and Bahrain [114], practice near-total Internet monitoring.

Others, including the United States and Britain, have large-scale technical capacity—but subject to legal limits. This section explains how the National Security Agency (NSA) and Government Communication Headquarters (GCHQ) have used third-party cookies in their respective surveillance programs, as well as briefly discusses the laws that govern surveillance of Internet traffic within the United States.

2.2.1 NSA and GCHQ use of third-party cookies

Leaked documents reflect at least three ways in which the NSA has used third-party cookies obtained from its Internet intercepts. First, the agency has investigated passively identifying Tor users by associating cookies with non-Tor sessions. Specifically, the NSA attempted to link a Google third-party advertising cookie between Tor and non-Tor sessions [308].

Second, the agency has an active, man-in-the-middle system ("QUANTUM-COOKIE") that induces cookie disclosure [308]. Applications include identifying Tor users and targeting malware delivery.

Third, the agency has used passively obtained cookies to target active man-in-the-middle exploitation. On at least one occasion, the NSA offered a Google cookie to single out a user for exploitation [296].

In addition to these specific applications, HTTP analytical tools (such as "XKEYSCORE") incorporate cookie data. An analyst could easily take advantage of third-party cookies when querying intercepted data [144].

Several leaked documents also reveal two GCHQ programs for surveilling and targeting users via third-party tracking data, both from web browsers and mobile applications. One such program is "MUTANT BROTH", a repository of tracking cookies linked with additional metadata such as IP addresses and User-Agent strings. This repository is reported to have been used for targeted malware delivery [135].

The other program, "BADASS", offers a similar repository and search interface for querying information leakage from mobile apps. The system collects and extracts leaked identifiers, device and operating system details, and additional information transmitted in plaintext [196].

2.2.2 United States Internet monitoring

The law surrounding NSA authority derives from a complex mixture of constitutional doctrine, statutory restrictions, and executive regulation. One emergent property is that, when at least one party to a communication is a non-US person (i.e., not a permanent resident or citizen of the US), and that party is located outside the United States, that party is eligible for warrantless surveillance.[4] "Upstream" interception devices, controlled by the NSA and foreign partners, are exposed to large volumes of this "one-end foreign" Internet traffic. While the details remain classified, it also appears that a substantial quantity of one-end foreign traffic is temporarily retained. Leaks indicate that at least some installations of "XKEYSCORE," a distributed analysis system, maintain a multi-day buffer of Internet traffic [50].

2.2.3 Surveillance: attacks, defenses, and measurement

While there is a large body of work on what a passive adversary can infer about users, virtually all of it concerns attacks arising from side-channels in encrypted traffic, particularly Tor. While Tor is insecure against a global passive adversary, traffic analysis attacks have been studied with respect to passive and active adversaries with less comprehensive access to the network [240, 241]. *Website fingerprinting* allows a local eavesdropper to determine which of a set of web pages the user is visiting,

[4] One-end foreign wireline interceptions inside the United States are generally governed by Section 702 of the FISA Amendments Act [251, 273]. Two-end foreign interceptions and one-end foreign wireless interceptions inside the United States are generally governed by Executive Order 12333. Interceptions outside the United States are also generally governed by Executive Order 12333 [12].

even if the connection is encrypted, by observing packet lengths and other features [154, 156, 265, 267]. Other side-channel attacks include timing attacks on SSH [297], leaks in web forms, [85] and inferring spoken phrases from VoIP [338].

By contrast, in we study users who do *not* use network-level anonymity tools. The adversary's main challenge is not attacking encrypted traffic, but rather linking different unencrypted traffic flows to the same (real-world) identity.

Closer to our work, Arnbak and Goldberg studied how the NSA could actively redirect U.S. traffic abroad, so as to bring it within broader surveillance authorities [45].[5] The IXMaps tool allows users to interactively view the routes taken by their traffic and intersection with known NSA wiretapping sites [91].

In Chapter 7 we study how well a passive network adversary can link and de-anonymize a user's HTTP web traffic under several technical and legal models. We find that an adversary can typically link between 62 and 75% of a user's traffic, although they are much less successful when legal restrictions are applied. Vanrykel et al. apply a similar analysis to traffic originating from mobile applications and find that a global passive adversary can link 57% of a user's traffic [323].

2.3 The state of privacy review in web standards

Modern web browsers provide many features that can be abused by web trackers (Section 2.1). The additional tracking surface exposed by a new feature is often unintended. In some cases when abuse is discovered, the feature specification is updated or additional controls are added to the browser to allow users to mitigate the privacy risk. Examples include the addition of user preferences to restrict WebRTC after it was discovered that scripts were using it to harvest local IP addresses [21],

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

or the additional state management controls added to Flash after trackers started respawning HTTP cookies from Flash storage (Section 2.1.1).

We examine the standardization process and the extent to which privacy review occurs during that process. In Section 5.2 we dig deeper, presenting a case study of the Battery Status API, which was removed from multiple browsers following the discovery of privacy and fingerprinting risks—an unprecedented reaction by browser vendors. We use the lessons learned from this case study to make concrete recommendations for improving the process (Section 5.2.3).

2.3.1 The W3C standardization process

The W3C employs a *maturity level* model in the standardization process [11]. Specifications start as a community group Working Draft and may undergo several revisions while the scope and content is refined. Once the specification is ready for a final review by a wide audience, it will progress to a Candidate Recommendation. The W3C formally calls for implementations at this stage, although in practice they may already exist. Feedback from the Candidate Recommendation and experience gathered from implementations is used to refine the specification further. If the specification requires no substantive changes it will progress to a Proposed Recommendation. After a final set of endorsements a specification will progress to a full W3C Recommendation.

The lengthy standardization process is consensus-driven. The stakeholders of a standard are generally organized into Working Groups, typically comprised of employees of browser vendors and other technology companies. To reach consensus, all members must agree on a decision. Other Working Groups, such as those specializing in privacy, accessibility, or web architecture may give their input on aspects of the specification relevant to their mission. Additionally, the specification Working Group must provide evidence of wide review, which includes reviews by a number of external parties: the public (i.e. researchers) and NGOs (some of whom are W3C members).

Privacy reviews often happen during the draft stage, although the depth of reviews can vary. As of 2018, the official W3C Process Document [11] does not require a privacy review. In practice, reviews are often performed prior to a draft entering the Candidate Recommendation level. A privacy consideration section can be *normative*, in which the statements included are requirements that an implementation must follow to be compliant with the specification. Alternatively, the section can be *non-normative*, which is used to provide extra information, context, or recommendations that an implementation is not required to adhere to.

The W3C's Technical Architecture Group (TAG), which aims to build consensus on principles of web architecture, published the Security and Privacy Self-Review Questionnaire [334]. The questionnaire exists to help authors, TAG, and others assess the privacy and security impacts of a specification. It recommends that authors review their specifications under several different threat models and asks a series of questions related to data access and quality. The W3C also has the Privacy Interest Group (PING), which provides guidance and advice for addressing privacy in standards [14].

2.3.2 W3C privacy assessment practices and requirements

Privacy reviews in specifications often focus on how the proposed design impacts web tracking. Past studies have shown that trackers frequently use many browser technologies to track users: by using stateful mechanisms like cookies, localStorage, and Flash storage to respawn cleared identifiers (Section 2.1.1), and by identifying a device solely by its properties (Section 2.1.2). The W3C's TAG has identified these advanced tracking behaviours as "actively harmful to the Web, because [they are] not under the control of users and not transparent" [248]. In response to fingerprinting concerns, the W3C's PING released a Working Group Note to provide guidance to specification authors on how to address and mitigate fingerprinting in their specifications [107].

When data is identified as potentially sensitive, such as that which relates to the user's device, behavior, location, or environment, various W3C specifications have applied different restrictions on access to that data. Some specifications have made the data available only in the top level browsing context (i.e. where access from third-party scripts is limited) [24], and others provide data only in a secure context (i.e. among other restrictions, requiring TLS) [350]. This type of data access also frequently requires user permission before any potentially sensitive information is made available. The Web Permissions API is a draft specification of a mechanism that allows users to manage these types of permissions in a user-friendly way [189].

2.3.3 Past privacy assessment research

Several studies have examined how privacy assessments are conducted as part of the specification process. Nick Doty identifies and addressees the challenges of privacy reviews in standardization bodies [108]. Doty describes the history of security and privacy consideration sections in Request for Comments (RFC), IETF specifications, and W3C specifications. RFC 6973 describes how design choices in internet protocols may impact privacy, and provides guidelines for drafting of Privacy Considerations sections in RFC documents [95]. Similarly, Frank Dawson describes a methodology for drafting the privacy considerations sections of W3C standards [102]. Dawson highlights the importance of privacy assessments during each stage of a draft specification and the need for an open process to incorporate the findings of external research.

Chapter 3

OpenWPM: A web measurement platform

Web privacy measurement has proven effective in controlling the actions of companies [127, 317]. 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 3.1) 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 [30], while still supporting stateful

measurements. We're able to scale to 1 million sites, without having to resort to a stripped-down browser [201] (a limitation we explore in detail in Section 3.1.3). (2) We provide comprehensive instrumentation by expanding on the rich browser extension instrumentation of FourthParty [220], 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.

Alongside OpenWPM we introduce a set of core measurement methods (Section 3.2) which solve many common web measurement tasks. We describe how OpenWPM data can be used to classify third-party content (Section 3.2.1) and trackers (Section 3.2.2) across the web and in emails. We present several options for browsing models and make recommendations on when to use them (Section 3.2.3). We describe how we detect tracking cookies (Section 3.2.4) and PII leakage (Section 3.2.5). Lastly, we further explore how we monitor Javascript calls (Section 3.2.6).

3.1 The design of OpenWPM

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.

OpenWPM is open source and has already been used for measurement in 22 academic studies. Section 3.1.4 examines the advanced features used by several of these studies. Nearly all of the measurements analyzed in this dissertation were collected using OpenWPM.

3.1.1 Previous web tracking measurement platforms

Web tracking researchers have created a number of tools for detecting and measuring tracking and privacy, such as FPDetective [30] and FourthParty [220]. OpenWPM builds on similar technologies as many of these platforms, but has several key design differences to support modular, comprehensive, and maintainable measurement. In some cases, we built directly 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 [30], with both native browser code and a proxy for instrumentation. FPDetective was used for the detection and analysis of fingerprinters, and much of the included instrumentation was built to support that [30]. 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[1] (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[2] browser extension for detecting browser fingerprinting. Chameleon Crawler uses

[1]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 this work, we've used stateful measurements to study cookie syncing (Section 4.3), cookie respawning (Section 4.2), and to replicate realistic user profiles (Section 7.2.1).

[2]`https://github.com/ghostwords/chameleon`

similar automation components, but supports a subset of OpenWPM's instrumentation. OpenWPM's Javascript monitoring uses techniques originally developed for Chameleon Crawler.

FourthParty is a Firefox plug-in for instrumenting the browser and does not handle automation [220]. OpenWPM has incorporated and expanded upon nearly all of FourthParty's instrumentation (Section 3.1).

WebXray is a PhantomJS based tool for measuring HTTP traffic [201]. It has been used to study third-party inclusions on the top 1 million sites, but as we show in Section 3.1.3, 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 [280].

XRay [194] and *AdFisher* [100] 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[3] uses an Apache Nutch based crawler. The Common Crawl dataset is the largest publicly available web crawl[4], 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 [285] 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 [227, 340]. Two major challenges are participant privacy and providing value to users to incentivize participation.

Many past platforms rely on native instrumentation code [30,244,294], which have a high maintenance cost and, in some cases a high cost-per-API monitored. In our

[3] `https://commoncrawl.org`

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

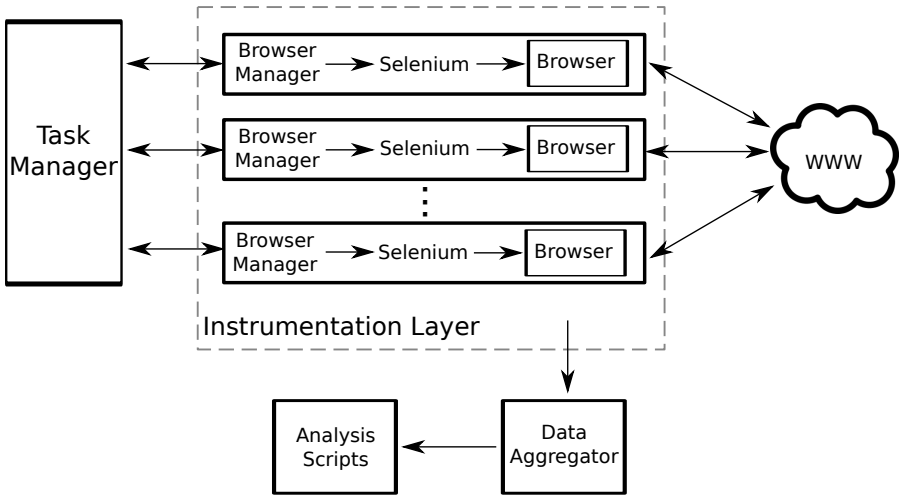


Figure 3.1: High-level overview of OpenWPM
The task manager monitors browser managers, which convert high-level commands into automated browser actions. The data aggregator receives and pre-processes data from instrumentation.

platform, the cost of monitoring new APIs is minimal (Section 3.1.3) 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.

3.1.2 Design and Implementation

We divided our browser automation and data collection infrastructure into four main modules: *browser managers* which act as an abstraction layer for automating individual *browser drivers*, 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 extend support 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 [272]. 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 5.1.5.

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 as third-party cookie blocking and the Tracking Protection feature. New extensions can easily be supported with only a few extra lines of code (Section 3.1.3). See Section 4.1.6, Section 5.1.7, Section 7.4, and Section 7.3.4 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 [290], 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 by 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.

OpenWPM provides several hooks for data access: (1) raw data on disk, (2) at the network level with an HTTP probes in a Firefox extension, and (3) at the Javascript level with page script probes injected by our extension. This provides nearly full coverage of a browser's interaction with the web and the system. Each level of instrumentation keys data with a unique visit id 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 — HTTP data is collected in a Firefox extension[5] through the browser's internal HTTP events[6]. We used Fourthparty [220] as a starting point and extend their probes in a number of ways:

- *Save request and response bodies in addition to headers.* We parse and save HTTP request POST bodies. This instrumentation has proven useful in detecting PII exfiltration (Section 6.2.4). We also provide the option to save HTTP response content, either for all responses or only for scripts. The former can be used to fully audit a site visit, while the later is useful in verifying script activity.

- *Explicitly link HTTP redirects.* HTTP redirects are common in advertising; trackers use HTTP redirects to run ad auctions, to cookie sync, and to exfiltrate and share PII (Section 6.2). HTTP header fields can be used to trace redirects, but the process is error prone.[7] To link HTTP redirects explicitly, we use Firefox's internal `channelId`, which is normally used to identify channels across process boundaries. We overwrite `asyncOnChannelRedirect` to log a mapping between `channelId`s during an HTTP redirect.

[5]Earlier versions of OpenWPM used Mitmproxy (`https://mitmproxy.org/`) to record all HTTP request and response headers. To capture HTTPS traffic we generated a self-signed certificate and loaded it into Firefox. Several of the measurements included in this dissertation were collected with the proxy instrumentation (e.g., Section 4.1 and Chapter 7). While proxy-based HTTP instrumentation is more modular, extension-based instrumentation provides much deeper context for each request and response.

[6]The internal Observer Notifications (`https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM/Observer_Notifications`) monitored by our extension are: `http-on-modify-request` which provides access to request headers and POST bodies, `http-on-examine-response` which provides access to response headers and content, and two additional response events for cached responses (`http-on-examine-cached-response` and `http-on-examine-merged-response`).

[7]Linking requests across redirects using only HTTP data has several sources of ambiguity and errors. The browser may upgrade the scheme, expand relative URLs, or drop a malformed header entirely. A URL may be requested and subsequently redirected in both the main frame and in nested iframes, leaving only the `Referer` header to disambiguate the traffic. `Referer` headers may be stripped or truncated depending on the page's Referrer policy.

- *Gather additional internal context for each request.* Browser request and response objects contain additional context beyond what is exposed to the network, including: whether the request is an `XHR`, the document in which the resource is loaded as well as the document which triggered the load, the element type responsible for the load, and whether the browser considers the load third-party. We record all of this context alongside each request.

- *Record the call context for requests triggered by Javascript.* The full call stack associated with a request is usually available through the browser's developer tools. We use the same interfaces to retrieve and parse the callstack for each HTTP request. The call stack allows the researcher to discover which script or scripts were involved in triggering a resource request. Example uses include tracking down the source of a PII leak (Section 6.2) or determining the script responsible for a cookie sync.

Javascript Calls — OpenWPM's Javascript instrumentation makes it possible to monitor property accesses or script calls for any browser API. OpenWPM can monitor any built-in object, and will record all property accesses and method calls on a monitored object. In addition, any arguments passed to a method or values set on a property will be saved. This works even for Javascript files which have been minified or obfuscated with `eval`. Everything is logged directly to the SQLite aggregator.

We used Fourthparty's [220] Javascript call monitoring as a starting point. Fourthparty uses an ES6 Proxy object to define custom getters and setters on the `window.navigator` and `window.screen` interfaces.[8] We take a more direct approach—we directly replace the getter and setter functions for all `accessor` properties on an object or object prototype of interest. Object `data` properties, which don't have `get` and `set` descriptor properties to overwrite, are converted to

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

`accessor` properties. The new `get` and `set` functions first log the access and the return the origin value (in the case of `data` properties) or call the original method (in the case of `accessor` properties). We make several additional extensions beyond Fourthparty:

- *Recursively instrument objects.* OpenWPM can monitor access to nested object by instrumenting return values on-the-fly. Instrumented method calls which return objects will also have those objects instrumented before passing them to the calling script. Any further interaction with the returned object will be logged by OpenWPM.

- *Detect and prevent tampering.* 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.[9]

- *Record access to new APIs.* In addition to `window.screen` and `window.navigator` we monitor access to the following interfaces: `Storage`, `document.cookie`, `HTMLCanvasElement`, `CanvasRenderingContext2D`, `RTCPeerConntection`, `BatteryManager`, and `AudioContext`. We also monitor the prototype objects for several children of `AudioNode`.

- *Record the call and frame context for each access.* In addition to recording access to instrumented objects, we record the full call stack at the time of access. To do so, we throw an Error and parse the stack trace after each call or property intercept. This method is adapted from the Privacy Badger Firefox extension[10]. We also record both the current and top-level `document` URLs at

[9] 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.

[10] `https://github.com/EFForg/privacybadgerfirefox`

the time of access, which can let the researcher determine in which frame the access occurred.

Example workflow.

When a researcher uses OpenWPM the following workflow walks through the steps the platform takes when a command is issued.

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 visit ID and top-level page being visited to all enabled instrumentation modules (extension and 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.

3.1.3 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 4.1), 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[11]. This virtual machine costs around $300 per month using price estimates from March 2018. 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 5.1.2) is three additional lines of code, while the WebRTC measurement (Section 5.1.4) 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

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

Study	Year	Browser automation	Stateful measurements	Extension support	Plugin support	Automated login	Content saving	Monitor state changes	Javascript Instrumentation	Content extraction
FB Connect login permissions [279]	2014	●			●					○
HSTS and key pinning misconfigurations [178]	2015	●	●		●	○				●
The Web Privacy Census [39]	2015	●	●		●		●			
Geographic variations in tracking [131]	2015	●			●					
Analysis of malicious web shells [301]	2016	●								
Web trackers can deanonymize cryptocurrencies [141]	2017		●					●		○
Effects of tracking protection [221]	2017	●	●	●		●		●		

Table 3.1: Several studies which use OpenWPM for measurements.
An unfilled circle indicates that the feature was useful but application-specific programming or manual effort was still required.

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 4.1 requires between 70 and 108 lines of code. By comparison, the core infrastructure code and included instrumentation is over 5100 lines of code plus an additional 2300 lines of code in tests, showing that the platform saves a significant amount of engineering effort.

3.1.4 Applications of OpenWPM

Twenty one academic studies have used OpenWPM to perform a variety of web privacy and security measurements [29,39,40,59,65,119,120,121,141,178,200,204,207, 221,228,260,276,279,287,301] and two have made use of our public datasets [59,305].[12] Table 3.1 summarizes the advanced features of the platform used by several research papers.

Several research groups take advantage of the OpenWPM's use of a real browser, including to run measurement instances with Flash enabled [39] and to measure the effects of privacy browser extensions on tracking [221]. Goldfeder et al. modify

[12]This count includes the 5 published papers this dissertation is based on [29, 119, 120, 121, 260]

OpenWPM to run interactive sessions where a researcher can go through a check-out procedure with all of the platform's instrumentation enabled [141]. Others use OpenWPM solely for its robust automation [204, 301]. Maass et al. additionally use OpenWPM to run PrivacyScore[13], their on-demand scanning service which benchmarks a website's privacy practices [207].

3.2 Core web privacy measurement methods

OpenWPM serves as a common measurement platform for all of our studies. Complementary to OpenWPM, we've developed a set of core measurement methods which we've used across many of the studies incldued in this dissertation. Study-specific measurement methods are described in their respective chapters.

3.2.1 Distinguishing third-party from first-party content

Nearly all of our measurements require us to distinguish between first-party and third-party content, as only third-party content can be used to track cross-site or cross-email. This distinction is not always clear; many websites use several distinct domains for security and organizational purposes. The distinction is even less clear for email content, where the rendered document is not retrieved from a web server.

On the web. 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 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[14] in our analysis. We consider a site to be a poten-

[13] `https://privacyscore.org/`
[14] `https://publicsuffix.org/`

tial third-party if the PS+1 of the site does not match the landing page's PS+1.[15] Throughout this dissertation we use the word "domain" to refer to a site's PS+1. We use this method throughout the dissertation, in Chapter 4, Chapter 5, Chapter 6, and Chapter 7.

A first party may choose to use multiple domains to serve their content. As an example, `example.com` may serve static resources from `example-static.com`. In order to avoid over counting third-party domains when presenting aggregate statistics, we sometimes require a domain classified as third-party to appear on at least 2 separate first parties.

In emails. Many email clients load embedded content directly from remote servers. Thus, remote content present in multiple emails can track users in the same way third-party content can track users across sites on the web. However, unlike the web there isn't always a clear distinction of which requests are "third-party" and which are "first-party". For example, all resources loaded by webmail clients are considered third-party by the browser. We consider any request to a domain which is different than both the domain on which we registered for the mailing list and the domain of the sender's email address to be a third-party request. We use this method to classify third-party requests in our email tracking measurements in Section 6.1.

3.2.2 Identifying trackers

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

[15] We use a heuristic to determine the landing page URL in HTTP proxy data, which is given in Appendix A.1. HTTP data collected by our extension includes the loading tab's URL with every record, and thus does not require this heuristic.

consist of regular expressions and URL sub-strings 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[16]. 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 directly blocked that resource.[17] Resource loads which wouldn't have been directly blocked by these extensions are considered non-tracking. We classify tracking domains around the web (Section 4.1) and in emails (Section 6.1).

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.

3.2.3 Browsing Models

A number of factors must be taken into consideration when choosing a browsing model for measurements. Do the crawls need to approximate the profile of a real user? If so, a browsing model which takes into account the probability of a user visiting a

[16] `https://disconnect.me/trackerprotection`

[17] We only consider domains which directly match a filter to be trackers. In practice, a privacy tool may end up preventing a non-matching resource from loading by blocking the matching resource that embeds it or redirects to it.

site is appropriate. Is a tracking technique expected to be present on all pages of a site? If so, more sites can be reached by only visiting the homepage of each site. If not, the researcher may need to spider within sites at the expense of visiting fewer sites overall. We identify several trade offs that must be considered when choosing a browsing model and describe the models used in our own measurements.

Simulating real users versus sampling popular sites. First, a researcher should consider whether it's necessary to simulate real user browsing patterns, or if a simple sample of the popular sites is sufficient. Past researchers have taken numerous approaches to simulating real user browsing patterns. A study of price discrimination modeled histories of "affluent" and "budget" shoppers based on Alexa categories [227]. Several other studies used Alexa category lists to simulate users with topical interests [227, 262]. Other choices include the Quantcast list of top sites with demographic breakdowns [150] and Google search results for topical keywords [227]. Another study simulated users issuing search queries (to Bing) of pseudonymous users in the leaked AOL search log dataset, followed by visiting the top 5 results returned by Bing [203].

The majority of our measurements provide a cross-section of tracking on the web. As such, we primarily use a browsing model which samples the most popular sites. We use the Alexa top 1 million site list, which ranks sites based on their global popularity with Alexa Toolbar users. Before each measurement, OpenWPM retrieves an updated copy of the list. Depending on the measurement we may use the entire list directly (Section 4.1.1), we may truncate the list at a certain rank (Section 6.1.1), or we may sample across several distinct ranges of the list (Chapter 6). In Chapter 7 we measure the surveillance implications of web tracking we need to understand the risks for an average web user—necessitating the use of a browsing model that better approximates real users. For that measurement we use a modified AOL search query approach, inspired by Liu et al. [203] (see Section 7.2.1 for a full description).

Homepage versus internal pages. In some measurements we visit only the homepage of a site (Section 4.1), while for others we spider within pages (Chapter 6). To understand the differences between the two, we compare aggregate statistics for a crawl which visits only the homepage of the top 10,000 sites to one which visits 4 internal pages in addition to the homepage. We note several differences. The average number of third parties per site increases from 22 to 34. The 20 most popular third parities embedded on the homepages are found on 6% to 57% more sites when internal page loads are considered. Similarly, fingerprinting scripts found in Section 5.1 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%.

These differences are 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. In general, the analyses presented in this dissertation should be considered a lower bound on the amount of tracking present in the wild.

Measurement location. The majority of the measurements presented in this dissertation were collected from Princeton, New Jersey or from Amazon's US East region. Fruchter, et al. [131] used 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. However, they did not measure the use of the more invasive tracking techniques studied in this dissertation, such as scripts which exfiltrate PII from the page (see Chapter 6). These practices may be subject to different regulatory restrictions in different locations.

Connection type. Many of our measurements are performed using Amazon's EC2 service. Sites may respond differently to requests which originate from a cloud IP address rather than a consumer or commercial address. In fact, industry research

suggests that pages loaded on cloud machines receive 20-25% fewer tracking tags than those loaded on consumer devices [23].

3.2.4 Detecting User IDs

In several of our measurements (Section 4.2, Section 4.3, and Section 7.2) we need to detect cookies which store unique identifiers. Identifiers can be stored in many locations (e.g. HTTP cookies, Flash cookies), but to be sent back to trackers the identifiers must be included in HTTP cookies or query parameters of the request. We choose to focus on HTTP cookies as they are included in every request and thus provide a generic approach that does not necessitate the parsing of URL parameters. However, the general method can be applied to other storage locations of a similar format.

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

Browsers store cookies in a structured key-value format, allowing sites to provide both a *name string* and *value string*. In practice a site may store an identifier alongside other, non-identifying data in the same 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_-=.[18] This provides us with a set of parameter strings for each cookie.

We first filter all cookies which are not *long-lived*, i.e., those which have an expiration time less than three months.[19] We then define a cookie to be an identifier cookie if the value of any parameter string meets the following criteria:

- Is *stable*, and remains constant through all page visits. Dynamic strings may be timestamps or other non-identifiers.
- Is *constant-length* across all our datasets, and has a length between 8 and 100 characters.
- Is *user-specific*, so the values are unique across different browser instances in our dataset.
- Is *high-entropy*, with values sufficiently different between machines to enable unique identification. To test for this, we used the Ratcliff-Obershelp [60] algorithm to compute similarity scores between value strings. We then filter out all cookies with a similarity between measurements greater than or equal to 66%.[20]

We compare cookie value strings across synchronized measurement data. By using synchronized measurements, we avoid the problem of sites changing their cookie interaction behavior depending on a user's browsing time. For instance, in relation to the entropy heuristic, cookies with values that depend on time stamps will be easier to detect and ignore if the crawls have nearly the same timestamps for all actions. A list of identifying cookies generated from synchronized data can then be used to

[18]This configuration is used in Section 4.3.7. The variations of this algorithm used in Section 4.3.2 and Chapter 7 use a simpler parameter parsing which simply splits the cookie value on a number of common delimiters (e.g. : and &)

[19]Section 4.3.2 uses a variation of this algorithm in which cookies with an expiration time less than 1 month are filtered.

[20]The similarity cutoff was refined throughout our measurements. Section 4.3.2 uses 33%, Section 4.3.7 uses 66%, and Chapter 7 uses 55%.

detect identifying cookies in non-synchronized measurements by searching for cookies with matching domain, name, and parameter name.

3.2.5 Detecting PII Leakage

PII may leak to remote servers through resource requests. Detecting these leaks is not as simple as searching for PII in requests, since the information may be hashed or encoded, sometimes iteratively. To detect such leakage we develop a methodology that, given a set of encodings and hashes, a PII string, and a token from a resource request, is able to determine if the token is a transformation of the PII. Starting with the plaintext PII string we pre-compute a candidate set of tokens by applying all supported encodings and hashes iteratively, stopping once we reach three nested encodings or hashes. We then take the resource request token and apply all supported decodings to the value, checking if the result is present in the candidate set. If not, we iteratively apply decodings to the token until we reach a level of three nested decodings.

In a preliminary measurement we found no examples of a value that was encoded before being hashed. This is unsurprising, as hashed PII is used to sync data between parties and adding a transformation before the hash would prevent that use case. Thus, when analyzing requests, we restrict ourselves to at most three nested hashes for a set of 24 supported hashes, including `md5`, `sha1`, `sha256`. For encodings, we apply all possible combinations of 10 encodings, including `base64`, `urlencoding`, and `gzip`. The full list of supported hashes and encodings is given in Appendix A.2.

Resource requests must be split into tokens for efficient processing. We developed a set of parsing rules based on common patterns observed on the web.

- *URL tokens:* A URL is split into its components: `host`, `path`, `query`, and `fragment` strings. We first remove any file extensions from the `path`, then split it on the path separator (a forward slash), and finally tokenize each portion by

a set of common delimiters.[21] The `query` and `fragment` strings are also split on the same delimiters. We run the detection process on the request URL, the `Location` header, and the `Referer` header.

- *HTTP cookies:* We parse `Cookie` headers using the Python cookies[22] library. We then use the same URL delimiters to split the cookie `name` and `value` components.

- *HTTP POST bodies:* We found sites using a wide variety of formats for HTTP POST request bodies, such that parsing on common formats was impossible. Instead, we perform a substring search within the POST body content for all nested encodings and hashes of the target PII. Although this process is significantly slower than checking tokens against a candidate set, the relatively low frequency of POST requests makes this an acceptable cost in our analyses.

When detecting PII leaks on the web (Section 6.2) we check for leaks in all transmission vectors. Leaks during email rendering (Section 6.1.3) are more constrained; all modern email clients will not execute Javascript and generally don't support interactive forms. When detecting PII leaks during email rendering, we only examine the request URL.

3.2.6 Measuring Javascript calls

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

[21] The set of delimiters used to split strings from all storage locations is `&`, `|`, \, and `,`. We chose these delimiters by examining common structures on the web.

[22] `https://github.com/sashahart/cookies`

intercept and record access to specific Javascript objects, which is *not* affected by minification or obfuscation of the source code.

Using the Javascript calls instrumentation described in Section 3.1.2, we record access to specific APIs which have been found to be used to be used by tracking and fingerprinting scripts.[23] Each time an instrumented object is accessed, we record the full context of the access: the entire call stack, the top-level url of the site, the property and method being accessed, any provided arguments, and any properties set or returned. For each tracking 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 tracking when embedded on that first-party site.

When manual verification is necessary, we have two approaches which depend on the level of script obfuscation. If the script is not obfuscated we manually inspect the copy which was archived according to the procedure discussed in Section 3.1.2. 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[24] extension. We also occasionally spot check live versions of sites and scripts, falling back to the archive when there are discrepancies.

[23] As an example, we monitor access to `CanvasRenderingContext2D`, which is used by canvas fingerprinting scripts (see Section 5.1.2). The instrumentation can also be used to detect non-fingerprinting related tracking; we monitor the `window.body.innerHTML` and `window.body.outerHTML` to detect when a tracking script stringifies the DOM (see Section 6.2.4).

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

Chapter 4

Web tracking is ubiquitous

In this chapter we report results from a January 2016 measurement of the top 1 million sites (Section 4.1.1). 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 4.1.3). Using a new metric for quantifying tracking (Section 4.1.4), we find that the tracking-protection tool Ghostery (`https://www.ghostery.com/`) is effective, with some caveats (Section 4.1.6). We perform a targeted measurement of cookie respawning, a tracking technique used to subvert efforts by users to protect their privacy (Section 4.2). Lastly, we study cookie syncing, a workaround to the Same-Origin Policy, which allows trackers to share identifiers with each other (Section 4.3). We find that cookie syncing is pervasive (Section 4.3.7) and explore the ways it which it can amplify tracking techniques which work against user privacy (Section 4.3.5).

4.1 A 1-million-site census of online tracking

We run the Princeton Web Census, a monthly measurement 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 presence of tracking, allowing us to shine a light onto the practices of third parties and trackers across a large portion of the web. Throughout this dissertation we reference data collected by the census measurements. In this section, we examine the results of our January 2016 measurement, and explore the effectiveness of consumer privacy tools at controlling stateful web tracking.

4.1.1 Measurement configuration

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 4.1: 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.

We ran our measurements on a "c4.2xlarge" Amazon EC2 instance, which allocated 8 vCPUs and 15 GiB of memory per machine at the time of measurement (January 2016). With this configuration we were able to run 20 browser instances in parallel. OpenWPM was configured to collect HTTP Request and Response data (via an HTTP proxy), Javascript calls, and Javascript files using the instrumentation detailed in Section 3. Table 4.1 summarizes the measurement instance configurations.

For each site, the browser visited the homepage and waited until the site finished loading or until a 90 second timeout was reached. The browser did not interact with the site or visit any other pages within the site. In the event of a timeout, we

killed the process and restarted the browser for the next page visit, as described in Section 3.1.2. At the completion of a page visit the browser was closed and all state was cleared before starting the next page visit (with the exception of our stateful measurement—see Section 4.1.2).

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".

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).

4.1.2 Measuring stateful tracking at scale

To obtain a complete picture of tracking we must carry out stateful measurements in addition to stateless ones. Stateful measurements do not clear the browser's profile between page visits, meaning cookie and other browser storage persist from site to site. For some measurements the difference is not material, but for others, such as cookie syncing (Section 4.3), 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 (see Table 4.1).

This method is not without limitations. For example third parties which don't appear on the top sites used for 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 of 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. While this naive approach provides reasonable coverage, the coverage can likely be improved for future measurements by choosing the set of sites to visit during the seed crawl based on the cookies observed in during previous measurements.

4.1.3 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 at the time of measurement was over 81,000.

What is more surprising is that the prevalence of third parties quickly drops off: only 123 of these 81,000 were 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, were Google-owned domains. In fact, *Google, Facebook, Twitter, and AdNexus were the only third-party entities present on more than 10% of sites.*

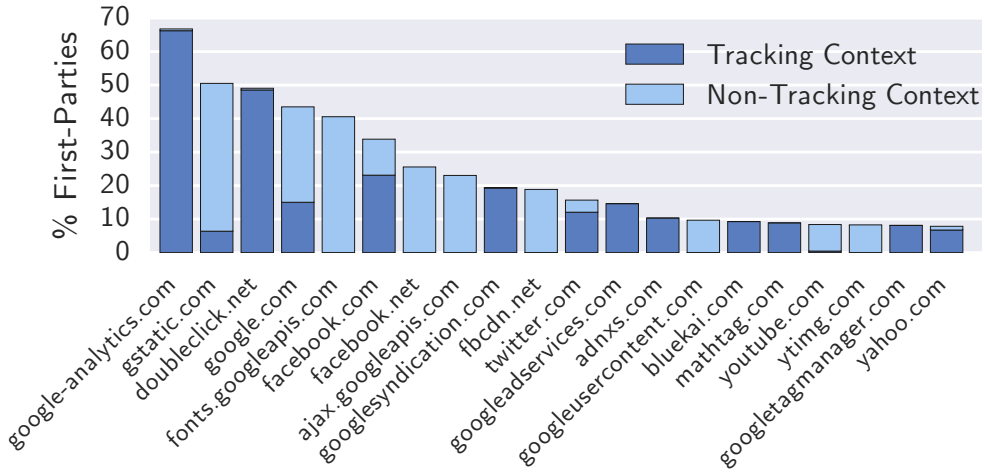


Figure 4.1: Top third parties on the top 1 million sites in January 2016. Not all third parties are classified as trackers, and in fact the same third party can be classified differently depending on the context (see Section 3.2.2).

Further, if we use the definition of tracking based on tracking-protection lists, as defined in Section 3.2.2, then trackers are even less prevalent. This is clear from Figure 4.1, 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 [201] and Disconnect [105]. As shown in Figure 4.2, Google, Facebook, Twitter, Amazon, AdNexus, and Oracle were the third-party organizations present on more than 10% of sites. In comparison to Libert's [201] 2014 findings, Akamai and ComScore fell significantly in market share to just 2.4% and 6.6% of sites. Oracle joined the top third parties by purchasing BlueKai and AddThis, showing that acquisitions can quickly change the tracking landscape.

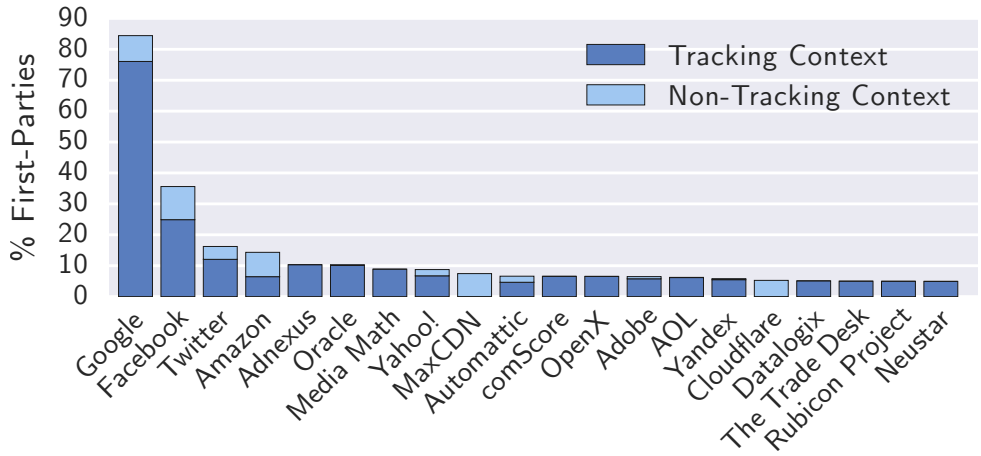


Figure 4.2: Organizations with the highest third-party presence on the top 1 million sites in January 2016. Not all third parties are classified as trackers, and in fact the same third party can be classified differently depending on the context (see Section 3.2.2).

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 [29, 47, 223].

4.1.4 Prominence: a third party ranking metric

In Section 4.1.3 we ranked third parties by the number of first party sites on which they appear. 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.

The measurement community 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 [39], but we argue that this is a misleading metric, since cookies may not have anything to do with tracking.

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 4.2: Top 20 third-parties on the Alexa top 1 million sites in January 2016, sorted by prominence. The number of first-party sites each third-party was 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.

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 $\frac{1}{N}$ [33, 179]. The exact relationship is not important to us; any formula for traffic can be plugged into our prominence metric below.

Definition:.

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

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 4.3. We propose that prominence is the right metric for:

1. Comparing third parties and identifying the top third parties. We present the list of top third parties by prominence in Table 4.2. 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 4.1.6.

3. Analyzing the evolution of the tracking ecosystem over time and comparing between studies. The robustness of the *rank-prominence curve* (Figure 4.3) makes it ideally suited for these purposes.

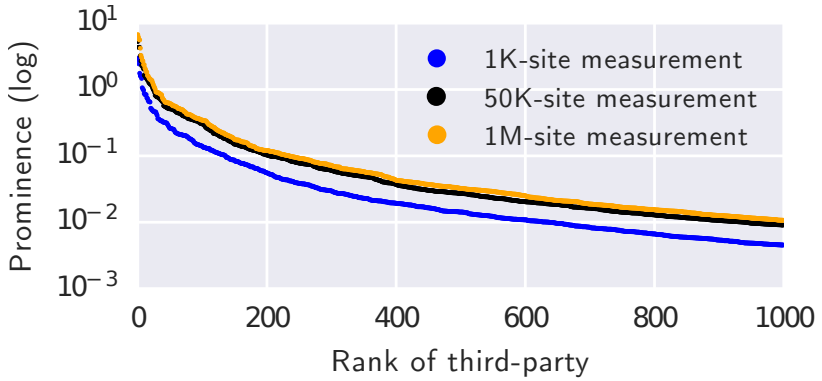


Figure 4.3: Prominence of third party as a function of prominence rank. We posit that the curve for the 1M-site measurement (which can be approximated by a 50k-site measurement) presents a useful aggregate picture of tracking.

4.1.5 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 4.4 we show the average number of third parties loaded across 100 of the top sites in each Alexa category in January 2016. Third parties are classified as trackers if they would have been blocked by one of the tracking protection lists (see Section 3.2.2).

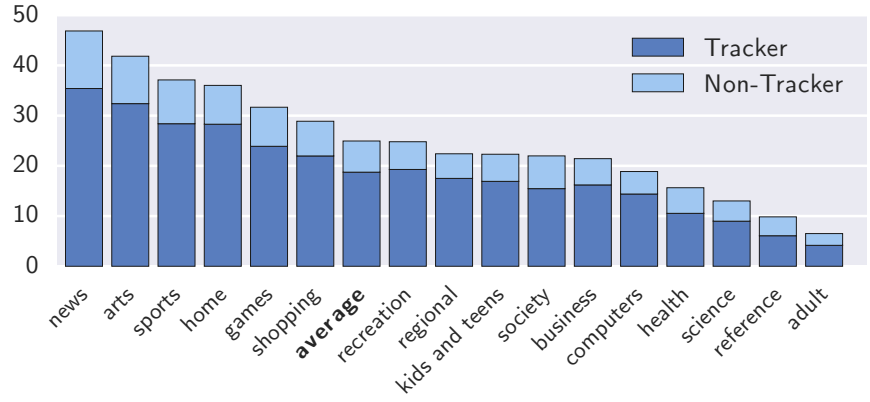


Figure 4.4: Average # of third parties in each Alexa category in January 2016.

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. In some cases there may be ethical or legal reasons for decreased tracking; for example, US government websites are restricted by law in how they can track users [250]. 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 may be pressured to monetize page views with significantly more advertising. In the case of shopping sites, trackers may be present to verify "conversions", or purchases that are the result of advertising displayed or clicked on another site.

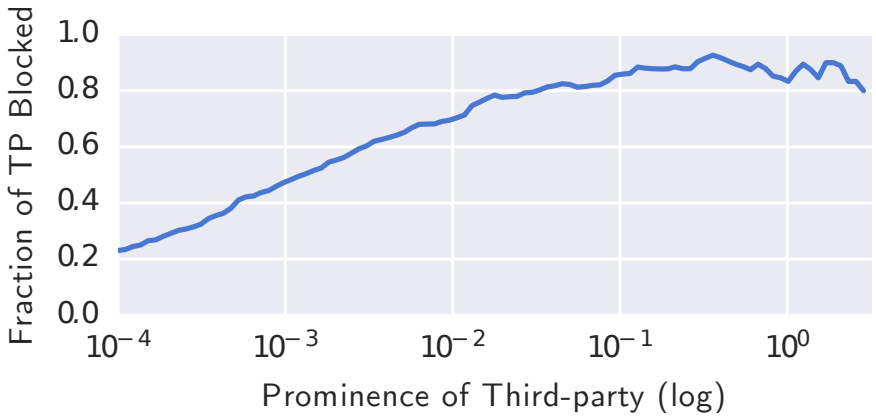


Figure 4.5: Fraction of third parties blocked by Ghostery in January 2016 as a function of the prominence of the third party. As defined earlier, a third party's prominence is the sum of the inverse ranks of the sites it appears on.

4.1.6 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.

We found Firefox's third-party cookie blocking to be effective. Specifically, only 237 sites (0.4%) had any third-party cookies set during our measurement set to block all third-party cookies ("Block TP Cookies" in Table 4.1). Most of these were 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 were six Australian news sites that first redirected to `news.com.au` before re-directing back to the initial domain, which seemed to be for tracking purposes.

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 4.3).

We also tested the Ghostery browser extension, and found that it was effective at reducing the number of third parties and ID cookies (Figure 4.6). The average number of third-party includes went down from 17.7 to 3.3, of which just 0.3 had

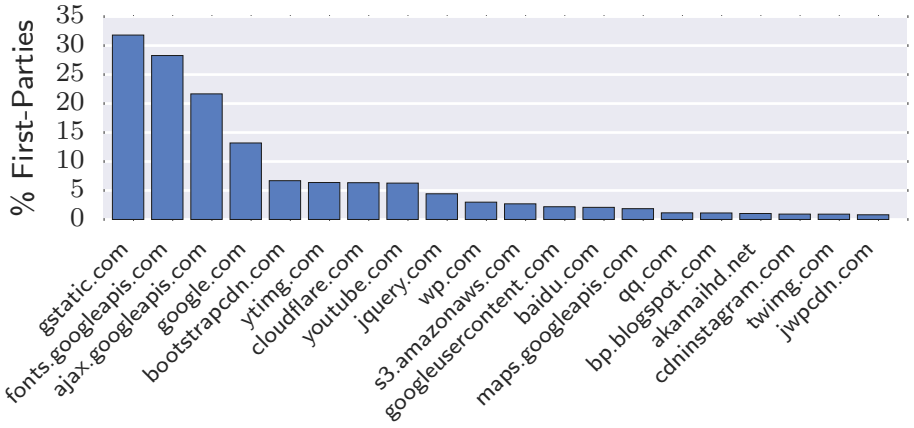


Figure 4.6: Third parties on the top 55k sites with Ghostery enabled in January 2016. The majority of the top third-party domains not blocked are CDNs or provide embedded content (such as Google Maps).

third-party cookies (0.1 with IDs). We examined the prominent third parties that were not blocked and found almost all of them to be content-delivery networks like `cloudflare.com` or widgets like `maps.google.com`, which Ghostery did not try to block. So Ghostery worked well at achieving its stated objectives.

However, the tool was less effective for obscure trackers (prominence < 0.1). In Chapter 5 we examine device fingerprinting, and show that less prominent fingerprinting scripts were not blocked as frequently by blocking tools (Section 5.1.7). 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.

4.2 Measuring Cookie Respawning

Evercookies are designed to overcome the "shortcomings" of the traditional tracking mechanisms. By utilizing multiple storage vectors that are less transparent to users and may be more difficult to clear, evercookies provide an extremely resilient tracking

mechanism, and have been found to be used by many popular sites to circumvent deliberate user actions [47, 101, 295]. In this section, we first provide a set of criteria that we used to automatically detect identifier strings, present detailed results of an automated analysis of respawning by Flash evercookies, and show the existence of respawning by HTTP cookies.

4.2.1 Flash cookies respawning HTTP cookies

Although there are many "exotic" storage vectors that can be used to store tracking identifiers, Flash cookies have a clear advantage of being shared between different browsers that make use of the Adobe Flash plugin[1]. We developed a procedure to automate the detection of respawning by Flash cookies employing the method discussed in Section 3.2.4 to detect IDs and using GNU/Linux's *strace* [166] debugging tool to log access to Flash cookies.

Compared to earlier respawning studies [47, 223, 295], the method employed in this section is different in terms of automation and scale. In prior studies, most of the work, including the matching of HTTP and Flash cookie identifiers was carried out manually. By automating the analysis and parallelizing the crawls, we were able to analyze 10,000 websites, which is substantially more than the previous studies (100 sites, 600 sites). Note that, similar to [223], we only visited the home pages, whereas [47, 295] visited 10 internal links on each website. Another methodological difference is that we maintained the Flash cookies when visiting different websites, whereas [47, 295] used a virtual machine to prevent contamination. Last, [223] also used the moving and contrasting Flash cookies from different computers to determine ID and non-ID strings, which is one of the main ideas of the analysis described below.

For this analysis we used data from four different crawls. First, we sequentially crawled the Alexa top 10,000 sites using OpenWPM and saved the accumulated

[1] iOS based devices and Chrome/Chromium bundled with the Pepper API are exceptions

HTTP and Flash cookies ($Crawl_1$). We then made three 10,000 site crawls using modCrawler[2], two of which were run with the Flash cookies loaded from the sequential crawl ($Crawl_{2,3}$). The third crawler ran on a different machine, without any data loaded from the previous crawl ($Crawl_4$). Note that, except for the sequential crawl ($Crawl_1$), we ran multiple browsers in parallel to extend the reach of the study at the cost of not keeping a profile state (cookies, localStorage) between visits. During each visit, we ran an *strace* instance that logs all open, read and write system calls of Firefox and all of its child processes. Trace logs were parsed to get a list of Flash cookies accessed during the visit, which were then parsed and inserted into a crawl database.

For the analysis, we first split the Flash cookie contents from the three crawls ($Crawl_{2,3,4}$) by using a common set of separators (e.g. "=:&;). We then took the common strings between crawls made with the same LSOs ($Crawl_{2,3}$) and subtracted the strings found in LSO contents from the unrelated crawl ($Crawl_4$). We then checked the cookie contents from the original profile ($Crawl_1$) and cookies collected during the visits made with the same LSO set ($Crawl_{2,3}$). Finally, we subtracted strings that are found in an unrelated visit's cookies ($Crawl_4$) to minimize the false positives. Note that, in order to further eliminate false positives, one can use cookies and LSOs from other unrelated crawls since an ID-string cannot be present in unrelated crawls.

For clarity, we express a simplified form of the operation in set notation:

$$\bigcup_{i=1}^{MaxRank} ((((F_{2_i} \cap F_{3_i}) \setminus F_4) \cap C_{2_i} \cap C_{3_i}) \setminus C_4),$$

where F_{n_i} denotes *Flash cookies* from $Crawl_n$ for the site with the Alexa rank i, C_{n_i} denotes *cookies* from $Crawl_n$ for the site with the Alexa rank i and F_4, and C_4 denotes all Flash cookies and HTTP cookies collected during $Crawl_4$.

[2]modCrawler is a crawler under the FPDetective project [30], available at `https://github.com/fpdetective/modCrawler`

Global rank	Site	CC	Respawning (Flash) domain	1st/3rd Party
16	sina.com.cn	CN	simg.sinajs.cn	3rd*
17	yandex.ru	RU	kiks.yandex.ru	1st
27	weibo.com	CN	simg.sinajs.cn	3rd*
41	hao123.com	CN	ar.hao123.com	1st
52	sohu.com	CN	tv.sohu.com	1st
64	ifeng.com	HK	y3.ifengimg.com	3rd*
69	youku.com	CN	irs01.net	3rd
178	56.com	CN	irs01.net	3rd
196	letv.com	CN	irs01.net	3rd
197	tudou.com	CN	irs01.net	3rd

Table 4.3: Top-ranked websites found to include respawning based on Flash cookies in May 2014. CC: ISO 3166-1 code of the country where the website is based. 3rd*: The domains that are different from the first-party but registered for the same company in the WHOIS database.

We applied the method described above to four crawls run in May 2014 and found that 33 different Flash cookies from 30 different domains respawned a total of 355 cookies on 107 first party domains during the two crawls ($Crawl_{2,3}$). Table 4.3 shows that on six of the top 100 sites, Flash cookies were used to respawn HTTP cookies. Nine of top ten sites on which we observed respawning belong to Chinese companies (one from Hong Kong) whereas the other site belongs to the top Russian search engine Yandex.

The Flash cookie that respawned the most cookies (69 cookies on 24 websites) was *bbcookie.sol* from the `bbcdn-bbnaut.ibillboard.com` domain which belongs to a company that was found to use Flash based fingerprinting [30]. Note that this Flash cookie respawned almost three HTTP cookies per site which belong to different third party domains (`bbelements.com`, `.ibillboard.com` and the first-party domain). Table 4.4 summarizes the most frequently respawned Flash cookies at the time of measurement.

Flash domain	# respawned cookies	
	Pass 1	Pass 2
bbcdn-bbnaut.ibillboard.com	63	69
irs01.net	21	18
embed.wistia.com	14	13
source.mmi.bemobile.ua	13	14
kiks.yandex.ru	11	11
static.baifendian.com	10	10
tv.sohu.com	7	7
ar.hao123.com	3	2
embed-ssl.wistia.com	3	3
img5.uloz.to	3	3

Table 4.4: The Flash cookies that respawned the most cookies on the Alexa top 10,000 sites in May 2014. The rightmost two columns represent the number of cookies respawned in two crawls made with the same set of Flash cookies ($Crawl_{2,3}$).

4.2.2 HTTP cookies respawning Flash cookies

In May 2014, we ran a sequential crawl of the Top 3,000 Alexa sites using OpenWPM and saved the accumulated HTTP and Flash cookies. We extracted IDs from this crawl's HTTP cookies using the method described in Section 3.2.4. We then made an additional sequential crawl of the Top 3,000 Alexa sites on a separate machine loading only the HTTP cookies from the initial crawl.

Our method of detecting HTTP respawning from Flash cookies is as follows: (i) take the intersection of the initial crawl's flash objects with the final crawl's flash objects (ii) subtract common strings from the intersection using an unrelated crawl's flash objects and (iii) search the resulting strings for the first crawl's extracted HTTP cookie IDs as described in Section 3.2.4. This enabled us to ensure that the IDs were indeed found in the Flash objects of both crawls, weren't common to unrelated crawls, and existed as IDs on the original machine. Using this method, we detected 11 different unique IDs common between the three storage locations.

These 11 IDs corresponded to 14 first-party domains, a summary of which is provided by Table A.1 in the Appendix. We primarily observed respawning from

JavaScript originating from two third-parties: `www.iovation.com`, a fraud detection company that was specialized in device fingerprinting, and `www.postaffiliatepro.com`, creators of affiliate tracking software (that runs in the first-party context). We also observed three instances at the time of measurement of what appeared to be in-house respawning scripts from three brands: Twitch Interactive (`twitch.tv` and `justin.tv`), `casino.com`, and `xlovecam.com`.

4.3 Measuring Cookie Syncing

Cookie synchronization—the practice of third-party domains sharing pseudonymous user IDs typically stored in cookies—provides the potential for more effective tracking, especially when coupled with technologies such as cookie respawning (Section 4.2) and device fingerprinting (Chapter 5). First, pairs of domains who both know the same IDs via synchronization can use these IDs to share data server-to-server. Second, cookies that are respawned through fingerprinting or supercookies may contain IDs that are widely shared due to prior sync events, enabling trackers to link a user's browsing history across cookie clears.

4.3.1 Detecting cookie synchronization

Using the ID detection technique outlined in Section 3.2.4, we identified cookies containing values likely to be user IDs. 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` header that will be sent along with the request. We therefore look for instances of IDs in `Referer` headers, request URLs, and response `Location` headers as described below.

We consider two parties to have cookie synced if a cookie ID appears in specific locations within the `Referer` header, *request URL*, and `Location` headers extracted

from HTTP request and response pairs. To determine the *sender* and *receiver* of a synced ID we use the following classification, in line with previous work [258]:

- If the ID appears in the *request URL*: the requested domain is the recipient of a synced ID.
- If the ID appears in the `Referer` header: the referring domain is the sender of the ID, and the requested domain is the receiver.
- If the ID appears in the `Location` header: 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.

4.3.2 Measurement configuration

We used version 0.2.0 of OpenWPM to run multiple stateful crawls of the top 3,000 Alexa domains on Amazon EC2[3] in 2014. The browsers were configured to load each site with a timeout of 60 seconds before continuing to the next site. The instances were configured to use three different Firefox privacy settings: allowing all cookies

[3] `http://aws.amazon.com/ec2/`

(i.e. no privacy-protective measures), allowing all cookies but enabling Do Not Track (DNT), and blocking third-party cookies. We found DNT to have a negligible impact on tracking and therefore omit it from the discussion of our results.[4]

4.3.3 Cookie syncing is widespread on the top sites

Table 4.5 shows the prevalence of cookie syncing compared to the overall usage of ID cookies. This analysis aggregates both third-party and first-party data, as many domains (e.g. `doubleclick.com`, `facebook.com`) exist in both the Alexa Top 3000 and as third-parties on other sites.

Statistic	**Third party cookie policy**	
	Allow	**Block**
# IDs	1308	938
# ID cookies	1482	953
# IDs in sync	435	347
# ID cookies in sync	596	353
# (First*) Parties in sync	(407) 730	(321) 450
# IDs known per party	1/2.0/1/33	1/1.8/1/36
# Parties knowing an ID	2/3.4/2/43	2/2.3/2/22

Table 4.5: Comparison of high-level cookie syncing statistics when allowing and disallowing third-party cookies (top 3,000 Alexa domains) as of a 2014 measurement. The format of the bottom two rows is minimum/mean/median/maximum. *Here we define a first-party as a site which was visited in the first-party context at any point in the crawl.

Table 4.6 shows a summary of the top 10 parties involved in cookie synchronization at the time of measurement under two cookie policies. Observe that although some parties were involved in less syncing under the stricter cookie policy, many of the top parties received the same number of IDs. Overall, disabling third-party cookies reduced the number of synced IDs and parties involved in syncing by nearly a factor

[4]With all cookies allowed, the impact of Do Not Track on the aggregate statistics we measure was negligible. In particular, enabling Do Not Track only reduced the number of domains involved in synchronization by 2.9% and the number of IDs being synced by 2.6%. This finding was consistent with studies such as Balebako et al.—they found that, due to lack of industry enforcement, Do Not Track provided little practical protection against trackers [49].

All Cookies Allowed		No 3P Cookies	
Domain	**# IDs**	**Domain**	**# IDs**
gemius.pl	33	gemius.pl	36
doubleclick.net	32	2o7.net	27
2o7.net	27	omtrdc.net	27
rubiconproject.com	25	cbsi.com	26
omtrdc.net	24	parsely.com	16
cbsi.com	24	marinsm.com	14
adnxs.com	22	gravity.com	14
openx.net	19	cxense.com	13
cloudfront.net	18	cloudfront.net	10
rlcdn.com	17	doubleclick.net	10

Table 4.6: Number of IDs known by the Top 10 parties involved in cookie sync at the time of our 2014 measurement under both the policy of allowing all cookies and blocking third-party cookies.

All Cookies Allowed		No 3P Cookies	
ID Creator	**# Domains**	**ID Creator**	**# Domains**
turn.com	43	sociomantic.com	22
adsrvr.org	30	mybuys.com	11
mookie1.com	29	mybuys.com	11
Unknown*	24	mercadolibre.com	9
media6degrees.com	23	shinobi.jp	7
parsely.com	22	newsanalytics.com.au	6
Unknown*	19	microsoft.com	6
titaltv.com	18	mercadolibre.cl	5
crwdcntrl.net	18	mercadolibre.com.ar	5
uservoice.com	15	rackspace.com	5

Table 4.7: Number of domains which had knowledge of unique IDs created by each listed domain at the time of our 2014 measurements. The ID creator was determined manually by first placement of cookie (* the relationship was unclear from HTTP/cookie logs).

of two. While this reduction may appear promising from a privacy standpoint, in Section 4.3.4 we see that even with this much sparser amount of data, database merges could have enabled domains to reconstruct a large portion of a user's browsing history.

Included in Table 4.7 is a summary of the top 10 most shared IDs at the time of measurement in 2014 under both cookie policies. As an example, consider the most shared ID when all third party cookies were allowed, which was originally created by `turn.com`. This ID was created and placed in a cookie on the first page visit that included Turn as a third party. On the next page visit, Turn made a `GET` requests to 25 unique hostnames with a referrer of the form `http://cdn.turn.com/server/ddc.htm?uid=<unique_id>...` that contained its ID. These 25 parties gained knowledge of Turn's ID, as well as their own tracking cookies, in the process. Similar sharing occurred as the user continued to browse, eventually leading to 43 total domains. With third-party cookies disabled, the top shared IDs came from a disjoint set of parties, largely composed of syncs which share a first party cookie with several third-party sites.

4.3.4 Back-end database synchronization

We now turn to quantifying how much trackers can learn about users' browsing histories by merging databases on the back-end based on synced IDs. Cookie syncing allows trackers to associate a given user both with their own pseudonymous ID and with IDs received through syncs, facilitating later back-end merges. We cannot observe these merges directly, so we do not know if such merges occur with any frequency. That said, there is a natural incentive in the tracking ecosystem to aggregate data in order to learn a much larger fraction of a user's history.

First, if we assume there is no collaboration among third-party trackers, only a handful of trackers were in position to track a sizeable portion of a user's browsing

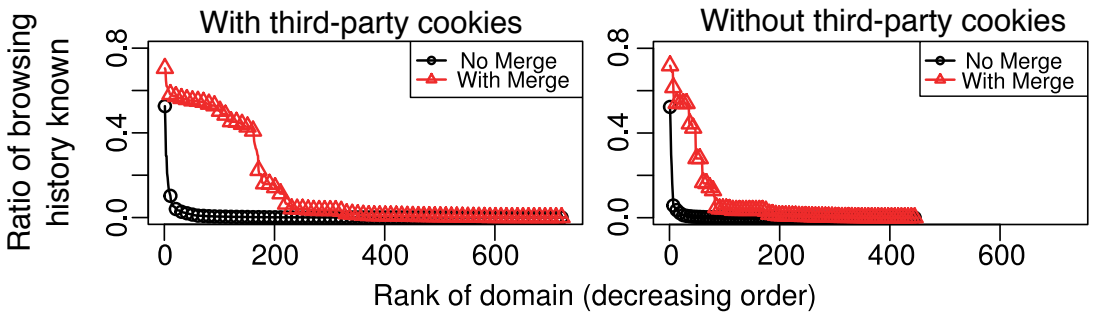


Figure 4.7: Proportions of user history known when allowing and blocking third-party cookies under the two different merging schemes at the time of our 2014 measurement. Note that since the x-axis is sorted by the proportion of a user's history that a domain could have recovered, and thus the domains may appear in different orders for the different models.

history. Only two of the 730 trackers using ID cookies could have recovered more than 40% of a user's history and only 11 could have recovered more than 10%. When third-party cookies were disabled, the respective counts were two and six. These results are consistent with earlier findings [280].

We consider the following model of back-end database merges: a domain can merge its records with a single other domain that mutually knows some ID. We assume that when two domains merge their records for a particular user, they will share their full records. Our model assumes some collaboration within the tracking ecosystem—among domains already known to share IDs—but is much weaker than assuming full cooperation.

Figure 4.7 shows the proportion of a user's 3000-site browsing history a tracker could have recovered, both with and without third-party cookies. When third-party cookies were blocked there were only roughly 60% as many parties. Observe that after introducing the ability for a tracker to merge records directly with one other tracker, the known proportion of a user's history that each tracker has access to grew significantly. When third-party cookies were allowed, 101 trackers could have reconstructed over 50% of a user's history and 161 could have recovered over 40%. Even when third-party cookies were blocked, 44 domains could have recover over 40%.

While we couldn't determine how prevalent back-end database merges were, they are incentivized by the industry. Merging histories will increase each tracker's respective coverage of users' browsing histories. Alternatively, a large tracker may act as a data broker and sell user browsing histories for a fee.

4.3.5 Cookie syncing amplifies bad actors

At a given point in time, cookie synchronization provides a mechanism for trackers to link a user's history together. Represented as a graph, sites in an individual's history can be represented as nodes with edges between sites if a user tagged with some pseudonymous ID visited both sites. When a user clears his cookies and restarts browsing, the third parties will place and sync a new set of IDs and eventually reconstruct a new history graph.

Since these history graphs correspond to browsing periods with completely different tracking IDs, they will be disjoint—in other words, trackers can not associate the individual's history before and after they clear cookies. However, if one of the trackers respawns a particular cookie, parts of the two history graphs can be connected by an edge, thereby linking an individual's history over time. This inference becomes stronger if this respawned ID is synced to a party present on a large number of the sites that a user visits.

To test this possibility, we ran a second measurement in 2014 in which we set up two 3,000-site crawls on Amazon EC2, crawl A and crawl B. We cleared the cookies, Flash storage, cache, and local storage from machine B and loaded the Flash files from A to seed respawning from Flash. Finally, we ran another 3,000 site crawl on site B.

We discovered a total of 26 domains that respawned IDs during the measurement between the two crawls on machine B either through Flash or through other means[5].

[5]The exact method here is not important, as we are concerned with the fact that an ID which has been respawned is later involved in sync.

Three of these IDs were later observed in sync flows. After conducting manual analysis, we were unable to determine the exact mechanism through which 18 of these IDs were respawned since we cleared all the storage vectors previously discussed, nor did we detect device fingerprinting. We conjecture that these IDs were respawned through some form of passive, server-side fingerprinting[6].

One of these IDs provides a useful case study. After respawning this ID, its owner, `merchenta.com`, passed it to `adnxs.com` through an HTTP redirect sync call. Now, `merchenta.com` by itself was not in a position to observe a large fraction of a user's history—it only appearred on a single first party domain in our crawl (`casino.com`). In fact, the largest percentage of a user's history observable by a cookie-respawning domain acting alone was 1.4%. However, by passing its ID to `adnxs.com`, `merchenta.com` enabled a much larger proportion of a user's history to be linked across state clears, if the two companies were to have shared information on the back end.

In particular, we observed `adnxs.com` on approximately 11% of first party sites across the two crawls. Thus `adnxs.com` had the ability to merge its records for a particular user before and after an attempt to clear cookies, although of course we have no insight into whether or not they actually did. This scenario enabled at least 11% of a user's history to be tracked over time.

4.3.6 Opt-out doesn't help

In order to study the effect of ad-industry opt-out tools on cookie syncing, we opted-out from all the listed companies on the Network Advertising Initiative (NAI)[7] and European Interactive Digital Advertising Alliance (EDAA)[8] opt-out pages.

[6]Note that marketing material from one of these respawning domains, `merchenta.com` mentioned tracking by fingerprinting: "Merchenta's unique fingerprint tracking enables consumers to be engaged playfully, over an extended period of time, long after solely cookie-based tracking loses its effectiveness", `http://www.merchenta.com/wp-content/files/Merchenta%20Case%20Study%20-%20Virgin.pdf`.

[7]`http://www.networkadvertising.org/choices/`

[8]`http://www.youronlinechoices.com/uk/your-ad-choices`

The use of opt-out cookies reduced the number of IDs involved in cookie synchronization by 30%. However, we saw only a 5% reduction in the number of parties involved in synchronization. This reduction was comparatively smaller than the reduction we saw when the browser was set to block third-party cookies. The composition of the top parties involved in synchronization was nearly the same as in the first-party cookie only case seen in Table 4.6. In Section 4.3.4 we show how, even under the larger reduction in sync activity afforded by blocking all third-party cookies, it was possible to recover a large portion of a user's browsing history using just a small number of the parties involved.

Note that most companies offering or honoring the opt-outs we evaluated did not promise to stop tracking when a user opts out, but only stop displaying behavioral advertising. While we observed tiny or nonexistent reductions in various forms of tracking due to opt-out, we make no claims about how opt-outs affect behavioral advertising.

4.3.7 Nearly all of the top third parties cookie sync

In 2016 we re-ran some of our analyses on a measurement of the top 100,000 sites to provide an updated and expanded analysis of cookie syncing.[9] In this updated view, the most prolific cookie-syncing third party was `doubleclick.net`—it shared 108 different IDs with 118 other third parties (this includes both events as a sender and receiver).

More interestingly, we found that the vast majority of the top third parties synced 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.

[9]For measurement details see the description in Section 4.1.1 and the "Default Stateful" configuration in Table 4.1.

We also found that third parties were highly connected by synced cookies. Specifically, of the top 50 third parties that were involved in cookie syncing, the probability that a random pair would have had at least one cookie in common is 85%. The corresponding probability for the top 100 was 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 (Section 2.2). How effective can this technique be? We explore the question in depth in Chapter 7, but for now 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 Section 7.3.5). The adversary uses this identifier to coerce or compromise a third party into enabling surveillance or targeted exploitation.

We found 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 belonged to the domain `adverticum.net`; it was synced or leaked to 82 other parties which were collectively present on 752 of the top 1,000 websites at the time of measurement! In fact, each of the top 10 most promiscuous cookies was shared with enough third parties to cover 60% or more of the top 1,000 sites.

4.4 Summary

In this chapter we showed that stateful web tracking is ubiquitous. We examined the long tail of online trackers, finding that tracking is concentrated in a few large organizations (Section 4.1.3). We showed that cookie syncing can change that by allowing

trackers to merge their databases, which dramatically increases the percentage of a user's browsing history each individual tracker can observe (Section 4.3.4). We also show that some trackers attempt to work around user controls by respawning cookies (Section 4.2) and examine how cookie syncing amplifies that practice (Section 4.3.5). These findings represent a snapshot of tracking on the top sites in January 2016, but the web is likely to have changed since then. Past research [182, 198] has shown a steady increase in the amount and variety of tracking over the past decade. Future research will be necessary to determine the effects of new regulations [92] and browser improvements [339].

Despite ubiquitous tracking, users do have options to protect themselves. Industry opt-outs and DNT are largely ineffective (Section 4.3.6), but tracking protection extensions block a majority of stateful trackers (Section 4.1.6). Users also have the option of clearing their browser's storage or blocking third-party cookies. This isn't the case for the tracking techniques we examine in the next two chapters. In Chapter 5 we show how third parties use device fingerprinting to track users, which is much harder to control. Fingerprinting offers no equivalent control to stateful tracking—users can't easily change their device's fingerprint. In Chapter 6 we examine an even more persistent tracking practice: the use of a user's PII as a tracking identifier.

Chapter 5

Persistent tracking with device fingerprinting

OpenWPM significantly reduces the engineering requirement of measuring device fingerprinting, making it easy to update old measurements and discover new techniques. In this Chapter, we demonstrate this by measuring several new fingerprinting techniques, three of which have never been measured at scale before. 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 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 perform a case study of the Battery API, which we find in use by a number of fingerprinting scripts. We examine the standardization process that led to the introduction of the API, the discovery of privacy vulnerabilities, and the ultimate removal of the API from several browser engines.

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 find that the public pressure brought on by measurement work can be effective in reducing the deployment of device fingerprinting by these large trackers. The standardization process is also headed in a promising direction: privacy reviews are more common and browser vendors have shown a willingness to remove APIs following the discovery of abuse.

5.1 Fingerprinting: a 1-Million site view

Our measurements provide an updated and comprehensive view of fingerprinting on the top websites. We find several high-level trends: fingerprinting scripts are more common on popular sites, more trackers are fingerprinting users, and fingerprinting scripts have increased in sophistication. We measured canvas fingerprinting in 2014 and 2016 (Section 5.1.2); canvas fingerprinting was quickly adopted by trackers, and its use increased during the measurement period. The relatively quick adoption of an API for fingerprinting is not unique to canvas—we show that trackers are early adopters of many of the recent HTML5 APIs, in one case comprising 88% of the use measured in the wild (Section 5.1.4). Theoretical attacks [21, 232, 256] have been quickly implemented by trackers and deployed in the wild (Section 5.1.2, Section 5.1.4, and Section 5.1.6). We also discover trackers developing new techniques; we found two independent implementations of Audio API fingerprinting, despite a lack of previously discovered attacks (Section 5.1.5). Finally, we find that tracking protection tools catch popular scripts and techniques, but frequently miss scripts in the long tail (Section 5.1.7).

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

Table 5.1: Prevalence of fingerprinting scripts on different slices of the top sites in January 2016. More popular sites are more likely to have fingerprinting scripts.

5.1.1 Measurement configuration

To detect fingerprinting, we analyze the Javascript calls made during the January 2016 measurements described in Section 4.1.1, specifically the "Default Stateless" 1-million-site measurement listed in Table 4.1.[1] Our fingerprinting detection methodology utilizes data collected by the Javascript instrumentation described in Section 3.1.2. 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 future work (Chapter 8).

[1] We measured canvas fingerprinting twice, once in 2014 and once in 2016. The 2014 measurement used was performed using modCrawler (part of the FPDetective project [30]), and consisted of a crawl of the top 100,000 Alexa sites.

5.1.2 Canvas Fingerprinting

Privacy threat. The HTML Canvas allows web applications 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 [232]. 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.

The same text can be rendered in different ways on different computers depending on the operating system, font library, graphics card, graphics driver and the browser. This may be due to the differences in font rasterization such as anti-aliasing, hinting or sub-pixel smoothing, differences in system fonts, API implementations or even the physical display [232]. In order to maximize the diversity of outcomes, the adversary may draw as many different letters as possible to the canvas. Mowery and Shacham, for instance, used the pangram *How quickly daft jumping zebras vex* in their experiments.

Figure 5.1 shows the basic flow of operations to fingerprint canvas. When a user visits a page, the fingerprinting script first draws text with the font and size of its choice and adds background colors (1). Next, the script calls Canvas API's `ToDataURL` method to get the canvas pixel data in *dataURL* format (2), which is a Base64 encoded representation of the binary pixel data. Finally, the script takes the hash of the text-encoded pixel data (3), which serves as the fingerprint and may be combined with other high-entropy browser properties such as the list of plugins, the list of fonts, or the user agent string [110].

Detection methodology. We performed two measurements, one in 2014 and one in 2016. In the time between the two measurements, the Canvas API received broader adoption for non-fingerprinting purposes, which necessitated several changes to the detection methodology to reduce false positives. In our 2016 measurements

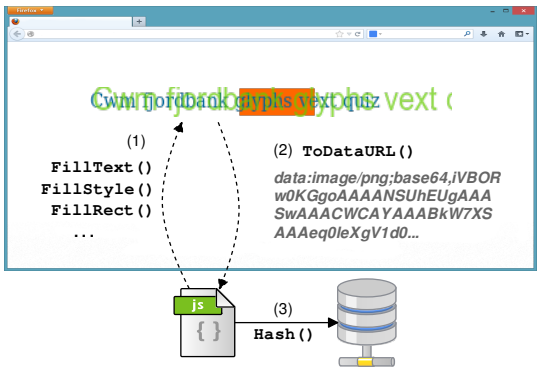


Figure 5.1: Sample canvas fingerprinting implementation

we record access to nearly all of properties and methods of the `HTMLCanvasElement` interface and of the `CanvasRenderingContext2D` interface. We present the filtering criteria for that measurement, and note the differences from the 2014 measurement where necessary:

1. The canvas element's `height` and `width` properties must not be set below 16 px.[2]

2. Text must be written to canvas with least two colors or at least 10 distinct characters.[3]

3. The script should not call the `save`, `restore`, or `addEventListener` methods of the rendering context.[4]

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.[5]

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

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

[3]A minimum color or character requirement for the written text was not applied during the 2014 measurement.

[4]Filtering on the `save`, `restore`, or `addEventListener` methods calls was not applied during the 2014 measurement

[5]The 2014 measurement did not consider scripts which called `getImageData`

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 5.2: Canvas fingerprinting on the Alexa Top 1 Million sites in January 2016. For a more complete list of scripts, see Table A.3 in the Appendix.

detection methodology by inspecting the images drawn and the source code. We found a mere 4 false positives out of 3493 scripts identified in our January 2016 1-million-site measurement. Each of the 4 is only present on a single first-party.

Results. In 2014 we found canvas fingerprinting on 5,542 (5.5%) of the top 100,000 sites. The overwhelming majority of these instances (95%) were from a single provider (addthis.com), but we also found an additional 19 domains performing canvas fingerprinting. A list of the most popular scripts at the time is provided in Appendix Table A.2. In 2016 we discovered canvas fingerprinting on 14,371 (1.6%) of the top 1 million sites. The vast majority (98.2%) were from third-party scripts. These scripts were loaded from about 3,500 URLs hosted on about 400 domains. Table 5.2 shows the top 5 domains which served canvas fingerprinting scripts ordered by the number of first-parties they were present on.

Comparing the two measurements we find three important trends. First, the most prominent trackers have by-and-large stopped using it, suggesting the the public backlash [41, 186] that followed the initial 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 may be less concerned about public perception. As the technique evolves, the images used have increased in variety and complexity, as we detail in Figure 5.2. Third, we observe an apparent shift

Figure 5.2: Example canvas renderings used in fingerprinting scripts found during our 2014 and 2016 measurements.

in usage from behavioral tracking to fraud detection, in line with the ad industry's self-regulatory norm regarding acceptable uses of fingerprinting.

5.1.3 Canvas Font Fingerprinting

Privacy threat. The browser's font list is very useful for device fingerprinting [7]. 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 [30, 246]. But can fonts be enumerated using the Canvas interface? The only public discussion of the technique prior to our analysis seems to be a Tor Browser ticket from 2014[6]. To the best of our knowledge, we are the first to measure its usage in the wild.

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

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

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[2]	-

Table 5.3: Canvas font fingerprinting scripts on the top Alexa 1 Million sites in January 2016.
*: 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: We found canvas font fingerprinting on 3250 unique sites. Some sites include fingerprinting scripts from more than one domain.

Results. We found canvas-based font fingerprinting present on 3,250 first-party sites. This represents less than 1% of sites, but as Table 5.1 shows, the technique was more heavily used on the top sites, reaching 2.5% of the top 1000. The vast majority of cases (90%) were served by a single third party, `mathtag.com`. The number of sites with font fingerprinting represents a seven-fold increase over a 2013 study [30], although they did not consider Canvas. See Table 5.3 for a full list of scripts.

5.1.4 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 [314]. 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 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.[7] Both users behind a NAT and users behind a VPN/proxy can have additional identifying information exposed to websites without their knowledge or consent.

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`[8]. 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

[7]Several steps must be taken to have the browser generate ICE candidates. First, a `RTCDataChannel` must be created. 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. 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) [281] string from `RTCPeerConnection.localDescription` after the offer generation is complete. In our measurement we only found it necessary to instrument `RTCPeerConnection.onicecandidate()` to capture all current scripts.

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

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

Table 5.4: Summary of WebRTC local IP discovery on the top 1 million Alexa sites in January 2016.

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. We provide a summary in Table 5.4. The top 10 scripts accounted for 83% of usage, in line with our other observations about the small number of third parties responsible for most tracking. We provide a list of scripts in Table A.4 in the Appendix.

The number of confirmed non-tracking uses of unsolicited IP candidate discovery was small, and based on our analysis, none of them were 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 [74, 313, 314].

5.1.5 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.

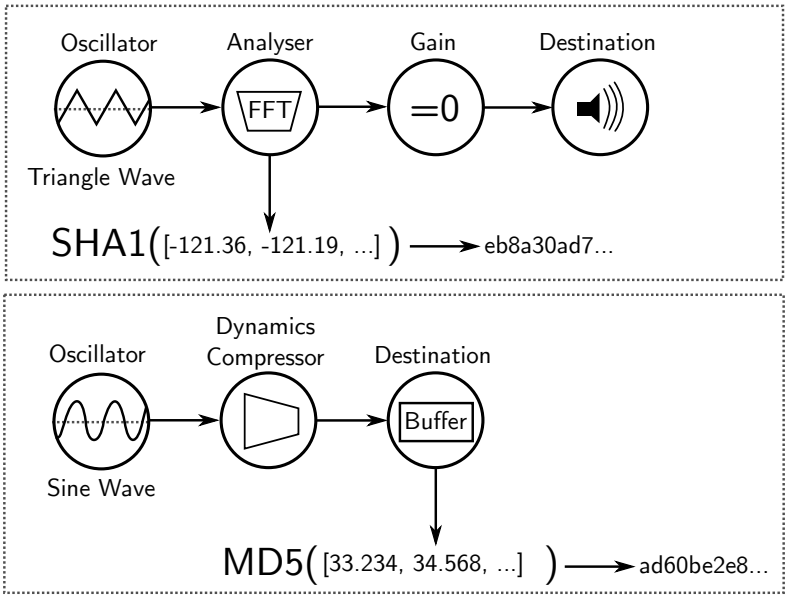


Figure 5.3: AudioContext node configuration used to generate a fingerprint as found in our January and March 2016 measurements. **Top:** Used by `www.cdn-net.com/cc.js` in an `AudioContext`. **Bottom:** Used by `client.a.pxi.pub/*/main.min.js` and `js.ad-score.com/score.min.js` in an `OfflineAudioContext`.

In the simplest case, a script from the company Liverail[9] checked for the existence of an `AudioContext` and `OscillatorNode` to add a single bit of information to a broader fingerprint. More sophisticated scripts processed 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 differences between the machines, while the same combination of machine and browser will produce the same output.

Figure 5.3 summarizes the two audio fingerprinting configurations we found in the wild. The top configuration was used by `*.cdn-net.com/cc.js`. First, the script creates an `AudioContext` and 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 (i.e., the com-

[9] `https://www.liverail.com/`

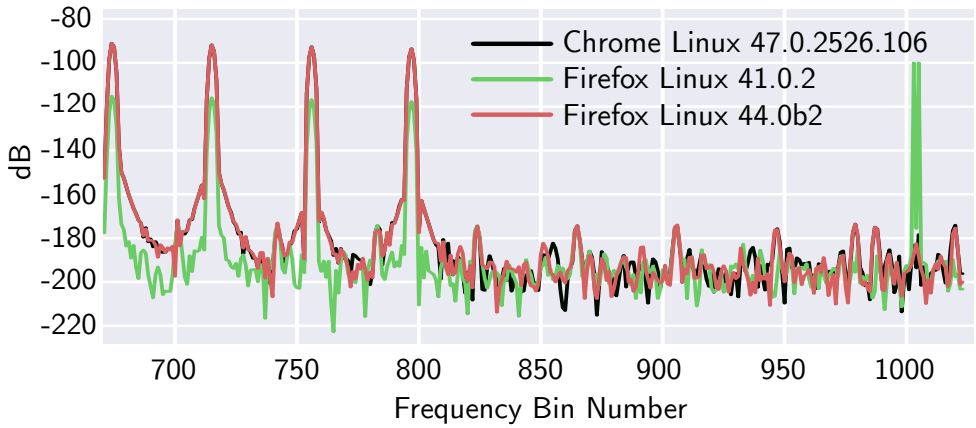


Figure 5.4: Visualization of processed `OscillatorNode` output from fingerprinting configuration found in `https://www.cdn-net.com/cc.js` in January 2016 for three different browsers on the same machine. We found these values to remain constant for each browser after several checks.

puter'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 was hashed and used as a fingerprint. The bottom configuration was used by two scripts, (`client.a.pxi.pub/*/main.min.js` and `http://js.ad-score.com/score.min.js`). These scripts used 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 scripts used a hash of the sum of values from the buffer as the fingerprint.

Effectiveness of audio fingerprinting We created a fingerprinting test page based on the scripts, which attracted visitors with 18,500 distinct cookies.[10] 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.

[10]The AudioContext fingerprinting test page is available at `https://audiofingerprint.openwpm.com`.

We found that this technique is very infrequently used as of March 2016. The most popular script was 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.

5.1.6 Battery Status API Fingerprinting

As a second example of bootstrapping, we analyze the Battery Status API, which allows a site to query the browser 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 [256]. 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 revisiting the site. More detail on this attack is given in Section 5.2.1. We discovered two fingerprinting scripts utilizing the API during our manual analysis of our January 2016 measurement data for other fingerprinting techniques.

One script, `https://go.lynxbroker.de/eat_heartbeat.js`, retrieved the current charge level of the host device and combined it with several other identifying features. These features included the canvas fingerprint and the user's local IP address retrieved with WebRTC as described in Section 5.1.2 and Section 5.1.4. The second script, `http://js.ad-score.com/score.min.js`, queried 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 were combined with other identifying features used to fingerprint a device.

In Section 5.2.2 we describe the findings of a larger, automated measurement of the use of Battery Status API in the wild.

As a result of our research and related findings of misuse, the Battery Status API was eventually removed from several major browsers [87]. We examine the events that led up to the removal and perform a retrospective analysis of the decision in Section 5.2.

5.1.7 The wild west of fingerprinting scripts

In Section 4.1.6 we found the various tracking protection measures to be very effective at reducing third-party tracking. In Table 5.5 we show how blocking tools miss many of the scripts we detected throughout Section 5.1, 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 5.5: Percentage of fingerprinting scripts blocked by Disconnect or the combination of EasyList and EasyPrivacy for all techniques described in Section 5.1 as of January 2016. 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.

OpenWPM's Javascript call monitoring (see Section 3.1.2) detects a large number of scripts not blocked by privacy tools at the time of measurement. The Disconnect

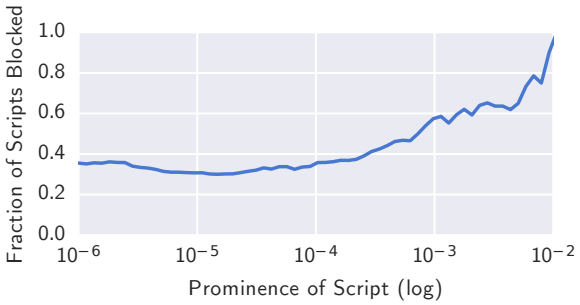


Figure 5.5: Fraction of fingerprinting scripts with prominence above a given level blocked by Disconnect, EasyList, or EasyPrivacy on the top 1M sites in January 2016.

list and a combination of EasyList and EasyPrivacy both performed similarly in their block rate. The privacy tools blocked canvas fingerprinting on over 78% of sites, and blocked canvas font fingerprinting on over 90%. However, only a fraction of the total number of scripts utilizing the techniques were blocked (between 10% and 25%) showing that less popular third parties were missed. Lesser-known techniques, like WebRTC IP discovery and Audio fingerprinting had even lower rates of detection.

In fact, fingerprinting scripts with a low prominence were blocked much less frequently than those with high prominence. Figure 5.5 shows the fraction of scripts which were blocked by Disconnect, EasyList, or Easyprivacy for all techniques analyzed in this section. 90% of scripts with a prominence above 0.01 were detected and blocked by one of the blocking lists, while only 35% of those with a prominence above 0.0001 were. The long tail of fingerprinting scripts were largely unblocked by the privacy tools.

5.2 Case study: the Battery Status API

The Battery Status API offers an interesting and unusual case study of privacy assessment in the web standardization process. The specification started with a typical progression: it went through a few iterations as a draft, was quickly implemented

by multiple browser vendors, and then progressed to a candidate recommendation — one which characterized the privacy impact as "minimal". Several years later, after it was implemented in all major browser engines and was nearing finalization, several privacy vulnerabilities were discovered [260] and actively exploited in the wild (Section 5.1.6). In an unprecedented move, two of the three major browser engines removed support for the API and another browser moved to an opt-in model.

In this section we analyze the privacy engineering aspects [147] of the Battery Status API, with a focus on the standardization and implementation process. Specifically, we perform a systematic review of the deployment of the Battery Status API (Section 5.2.1), we present the first large-scale measurement of Battery Status API use on the web (Section 5.2.2), and we extract useful privacy engineering practices and provide recommendations to improve the design process (Section 5.2.3).

5.2.1 The timeline of specification and adoption

Figure 5.6 summarizes the major developments during the design and implementation of the Battery Status API, from the initial specification to the eventual removal from several browsers. We discuss each in detail below.

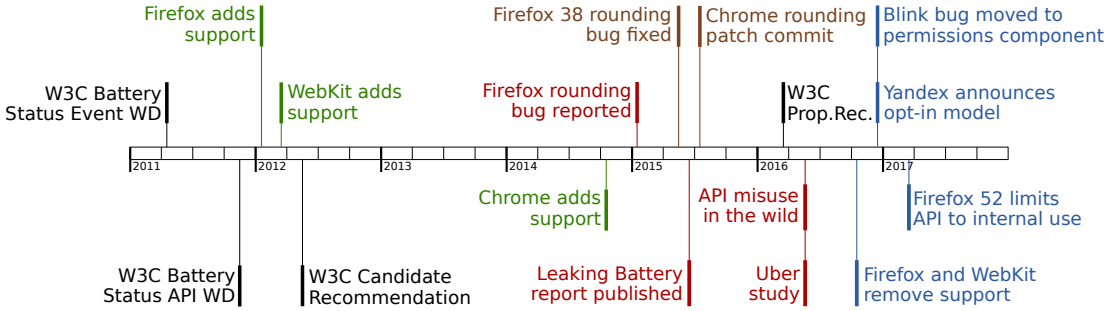


Figure 5.6: Timeline of events.

API specification

The Battery Status API is a browser mechanism that provides access to the power management information of a device [175]. An example use case listed in the W3C specification is responding to low battery levels. For instance, an online word processor might auto-save more frequently when the battery is low. The API provides the following information: the battery charge `level` (e.g. 0.43 when the battery has 43% remaining power), the `charging` status (whether or not the device is charging), the time in seconds to a full discharge (`dischargingTime`; when discharging) or a full charge (`chargingTime`; when charging).

In April 2011, the Battery Status Event Working Draft first described how a website can access battery information [171]. The specification was reworked into the Battery Status API in November 2011 [172] and progressed to a Candidate Recommendation in May 2012 [176]. This version identified a "minimal impact on privacy or fingerprinting", but suggested "the user agent can obfuscate the exposed value in a way that [websites] cannot directly know if a hosting device has no battery, is charging or is exposing fake values." [173]. This shows that although the specification authors felt the privacy impact was minimal, they felt there was enough of a risk that user agents (browsers) may want to hide a user's true battery status.

Adoption by browser vendors

The first implementations of the Battery Status API were in 2012 by Firefox [234] (mobile and desktop) and WebKit [333]. Chrome[11] also started an implementation in 2012 that was never completed, citing a general lack of interest and convincing use cases [19]. Chrome later removed and re-implemented it in 2014 on both mobile and desktop [19]. Other browsers based on Chrome's Blink engine supported the

[11]Chromium and Chrome are both based on the Blink rendering engine and expose the same Battery Status API. We refer exclusively to Chrome for the remainder of this section but our analysis is applicable to both browsers.

API soon after, such as Opera in 2014 [264]. This level of implementation fulfills W3C requirements of having at at least two independent implementations prior to finalization of a specification [11]. For a summary of support by additional browsers, see Table 5.6.

Discovery of privacy vulnerabilities

In 2015 Olejnik et al. [257] examined the W3C specification and the browser implementations of the API and found several privacy vulnerabilities. They showed that the API can be used to track users, both by using the status readouts as short-term identifiers and by using repeated readouts to reconstruct the battery capacity on certain platforms.

Battery status readouts made users of certain platforms vulnerable to tracking across sites by a third-party. The API's charge `level` returns a value between 0 and 1 which represents the ratio of the current charge remaining to the maximum capacity. The researchers found that Firefox on Linux returned a double-precision floating-point value for charge level, the result of returning the operating system's value directly without truncating the result. This means that there are a large number of possible values for the charge level. Thus, if a tracker were to see the same charge level readout on two different sites at the same instant (or within a short time window), it gives the tracker evidence that the page visits were from the same device. This is true even if the user cleared cookies between the two visits or used different browser contexts for the two visits, such as regular versus private browsing.

The researchers further pointed out that short-term tracking was possible even on platforms which didn't expose the high-precision charge level, although it was less effective. On platforms other than Firefox on Linux, the battery charge level was just two significant digits. However, by combining the charge level with `dischargeTime` and `chargeTime` the researchers estimated the possible number of states to be on

the order of millions under normal operating conditions. Thus, a tracker could still conclude that two page visits with the same status readout is likely the same device, particularly if it coupled that measurement with the device's IP address.

Finally, the researchers showed that the double-precision readouts for Firefox on Linux enabled a more sophisticated attack in which a site could recover the battery capacity. The attack works by reading the device's current charge level and calculating which possible battery capacities could result in that charge level based on how the underlying operating system battery library calculates charge level. As the tracker makes more readings, it decreases the number of possible battery capacity values that could result in the observed sequence of charge levels. In the end, a tracker could recover the actual battery capacity, and use that as a static identifier for the device. This capacity could also be included alongside other device properties in a broader device fingerprint.

Initial privacy improvements by browser vendors

In response to the 2015 report [257], Firefox fixed the precision of the battery level readout to two significant digits for all platforms [252]. Chrome did not return high precision readouts on Linux because of an implementation difference. However, had Chrome used a high precision source it would have exhibited the same behavior. To prevent this, Chrome preemptively capped the precision to no more than 2 digits on all platforms [325]. The W3C specification was amended [174] with non-normative privacy recommendations. Notable additions are:

1. *data minimization* — avoiding the exposition of high precision readings of battery status information

2. *user control* — optionally making the API subject to browser permissions that may require prompting the user prior to the use of the API

3. *incognito support* — disabling the API in private browsing modes

4. *transparency* — informing the user when the API is and has been in use

This response prevents the battery capacity from being recoverable and lessens the usefulness of status readouts as a short-term identifiers. However, even with reduced precision the battery status output will still provide additional identifying information that can be used to fingerprint and re-identify a device over short time intervals.

Discovery of misuse on the web

In Section 5.1.6 we discuss our initial discovery of misuse on the web. We found two scripts, together present on 22 sites, that used battery status as part of a device fingerprint. In Section 5.2.2 we present an automated analysis of the Alexa top 50,000 to audit the use of the API in late 2016.

One script, served by `lynxbroker.de`, retrieved only the current charge level of the device. The other script, served by `ad-score.com`, queried all properties of the `BatteryManager` interface, including the current charging status, the charge level, and the time remaining to discharge or recharge. Both scripts combine the retrieved battery information with other identifying properties of the device, such as the canvas fingerprint or system fonts, to create a device fingerprint.

Discovery of second-order privacy concerns

Until May 2016, the primary privacy concern of the API was its usefulness for online tracking. In May 2016, Uber disclosed its finding that users with a low battery are more willing to pay higher prices for a ride (i.e. more likely to book a ride during surge pricing) [84]. This sparked concerns that Uber or other companies could misuse battery status information to increase prices for users with a low battery level [82,143].

Browser	Engine	API Support (2016)	Current API Support (2018)
Firefox	Gecko	Initial support since version 10 [234]	Inaccessible to web content [271] [236]
Safari	WebKit	Not enabled. WebKit support since 2012 [333]	WebKit removed [111]
Chrome	Blink	Supported since version 38 [19]	Restrictions under consideration [254]
Edge	EdgeHTML	Not supported. Considered [226]	Not supported. Considered [226]
Yandex	Blink	Supported	Spoofing by default, opt-in [255,343]

Table 5.6: API support in various browsers and privacy strategies employed.

Mozilla cited these second order privacy concerns on W3C mailing lists during their initial discussions of whether to remove or restrict the API [77]. Materials obtained in a Subject Access Request confirmed that Uber does collect device battery level for use in fraud detection [103].

Removal of the API from browsers

As summarized in Table 5.6, support for the Battery Status API has tumbled in response to privacy concerns. At its peak in 2016, the API was implemented in the engines supporting Firefox, Chrome, and Safari (though it was disabled in Safari). In addition, other browsers built on the major engines, such as Opera and the Yandex Browser, also exposed the API to the web.

In October 2016 Mozilla announced it would remove access to API by web content in Firefox 52 [235]. The API is now restricted to internal browser code, and may eventually be exposed to WebExtensions-based browser extensions [271]. In March 2017 Firefox 52 was released with the API removed [236]. Following Mozilla's decision, WebKit, the underlying engine behind Safari browser, removed the Battery Status API from its source tree [111].

As of April 2018, Chrome developers have not provided an official stance on whether or how they will change the API. The current discussion in the relevant bug indicates that it may be disabled for non-secure contexts and cross-origin frames, reduced in precision, or removed entirely [254]. Microsoft Edge continues to have the API on its feature wishlist [226]; as of April 2018 there is no indication of a change.

Other browsers based on these engines could also restrict access to the API if desired. One such example is the Yandex Browser (built on the Blink engine), which now spoofs a fully charged status until the user explicitly enables the API using the appropriate browser setting [255]. Yandex has limited market share, but we include it in the table to show the versatility of responses by browser vendors.

Browser vendors regularly make privacy-related changes and continually deprecate unused and insecure features. However, the removal of an entire API in response to privacy concerns is unprecedented. We verified that this type of feature removal has not happened before by checking a website which tracks browser changes for compatibility purposes[12].

The future of the API

As of April 2018, it is unclear how the specification and remaining implementations will progress. Since the API was implemented in both Chrome and Firefox in 2016, it fulfilled the W3C requirement of two interoperable implementations [11]. Thus, despite only having one current implementation in 2018 (i.e. Chrome), the specification could progress to a W3C Recommendation. If there isn't sufficient interest by the authors to continue the specification, it can be published as a W3C Note, which would signify the end of active development by the W3C. The specification authors have suggested restricting access to the API to secure, top-level browsing contexts as an additional step to the privacy risks associated with the API [169].

5.2.2 Use and misuse of the API in the wild

We measured the use of the Battery Status API on the homepages of the Alexa top 50,000 sites in November 2016[13] using OpenWPM. We found that in total, the API

[12] `www.fxsitecompat.com`

[13] Since Mozilla and WebKit announced their intent to remove the API in October 2016, it is possible some sites or scripts could have changed their use of it in response to that news. We believe a 1 month time window is small enough that this is unlikely to have a significant effect on our results.

was used by 56 distinct parties on 841 sites. The majority of this usage was by third parties — 33 third parties on a total of 815 sites.

We manually classified these 33 scripts, to determine how the feature was being used around the time of its removal. We used the fingerprinting classification methodology outlined in Section 3.2.6. We classify a script as *benign* if it uses the battery status to do things we feel the API designers intended to support, such as performance and diagnostic measurements.[14] We classify a script as *tracking* if it uses the API for device identification, whether for fingerprinting, analytics, or fraud detection.

We found 16 third parties using the API for tracking at the time of measurement, 11 of which used it as part of a device fingerprint. An additional 8 third parties used the API for benign purposes. For the remaining 9 third parties, we were unable to classify the usage, either due to script obfuscation or vagueness. Scripts from the 16 trackers were present on 347 sites (or around 48% of sites on which we classified API use). The benign uses of the API were primarily from two third parties: YouTube, where the API was used in performance metrics for embedded videos, and Boomerang[15], a performance measurement library.

Representative examples of misuse

As a representative example of misuse, consider the fingerprinting script[16] served by Augur, a provider of device recognition services whose marketing material advertised "cookie-less tracking". At the time of our analysis, the script collected a large number of device properties: the device's fonts, plugins, WebGL properties, screen information, processor information, whether or not the device is blocking ads, whether or not the device is blocking cookies, and more. From the Battery Status API, the script

[14] We elaborate on intended versus unintended use in the *Representative examples of misuse* section.

[15] `https://soasta.github.io/boomerang/doc/`

[16] Located at `http[s]://cdn.augur.io/augur.min.js`

collected the current charge level and combined it with the other device properties. The script sent a heavily obfuscated payload back to Augur's server.

We discovered Augur's use of the Battery Status API for fingerprinting on 16 of the top 50,000 sites measured. In the November 2016 Princeton Web Census data (see Section 4.1) we found that 166 sites of the top 1 million embedded the script on their homepage. Several notable sites at the time were `glasses.com`, a major retailer of eyeglasses, `libertytax.com`, a tax preparation service, and `proactiv.com`, a major acne treatment provider.

While this example of misuse is clear cut, others are harder to categorize. A script by Riskified, a provider of fraud detection services for e-commerce, appeared to use Battery-related attributes as a part of a user reputation score, but did not seem to fingerprint the user. Even scripts that collected device fingerprints may not necessarily use them for tracking and behavioral profiling—another use case is to augment authentication [38], which would presumably meet fewer objections from users. In particular, the Battery Status API, due to its time-varying nature, can be used in continual authentication [300]. Regardless, all such uses appear to be unintended by the authors of the W3C specification, as they do not pertain to power management, and an analysis of mailing list discussions supports this interpretation.

A retrospective look at vendor response

Nearly half of the Battery Status API use we classified was for user profiling or identification—a use case the API designers did not intend to support. When measured in terms of distinct scripts rather than distinct sites, that fraction rises to two-thirds. Note that our measurement is conservative; parties which collect battery status information through performance or feature detection libraries can also use that information to track users, but we do not make this assumption.

At the time of Firefox's and WebKit's decisions to remove the API, the preliminary evidence suggested that it was being misused in the majority of cases [87]. Mozilla's discussion thread cites two specific uses: the research described in Section 5.2.1 and the Boomerang performance library [87]. The empirical data presented here suggest that use is indeed split between gathering performance metrics and tracking users. We found no additional categories of use, and confirm that the examples presented in the discussion thread reflect the broader usage on the web.

5.2.3 Lessons Learned & Recommendations

Based on insights from our case study, we extract a set of good privacy engineering practices and make several concrete recommendations on how standards bodies can improve the standardization process.

Information exchange between vendors and researchers is essential

Research can reveal theoretical privacy risks and their exploitation in the wild; it can also provide data on the usage of features on the web. Standards-based platforms such as the web are more conducive to research, and therefore attract more of it, compared to proprietary platforms. Further, when implementations are open-source, knowledge propagates not just from researchers to vendors but also among vendors.

Our case study illustrates these themes and their beneficial effects on privacy. After Firefox was reported to return high-precision values on Linux (Section 5.2.1—*Discovery of privacy vulnerabilities*), both Firefox and Chrome fixed the bug. Note that although Chrome did not exhibit the vulnerability, it chose to preemptively restrict the precision of the readout to prevent the possibility. This illustrates knowledge propagation between vendors, sparked by research results. Similarly, the specification was updated to include a recommendation to avoid high-precision readouts (Section 5.2.1—*Initial privacy improvements by browser vendors*).

Still, the specification process can benefit from a deeper connection to research. Deliberate attempts to break the privacy assumptions of specifications should be actively incentivized—perhaps by funding attack research, or by organizing a forum for academics and researchers to publish their privacy reviews.

The specification process should include a privacy review of implementations

On the modern web, proposed features often get deployed rapidly. At the time a specification is drafted, initial implementations are typically available in the development versions of browsers. By the time the spec is finalized, several vendors may already fully support a feature; in fact, the W3C requires at least two implementations to exist before official recommendation. We recommend that specification authors study implementations to prepare higher quality privacy assessments. Implementations enable field testing of theoretical attacks and can be examined for potential API misuses.

With the Battery Status API, the privacy risk stemmed from a difficult-to-predict interconnection of software layers, namely the browser acquiring information from the operating system in order to supply it to a web script. Such a risk is difficult to predict during the design phase, but becomes much easier to identify with access to an implementation.

In contrast to the Battery Status API, consider the Ambient Light Events API, which provides access to the light level of the device's surroundings. The specification was examined at the API design level, the implementation was tested, and the source code was reviewed—all as part of the review process. Issues identified at the implementation level led both Chrome and Firefox to address a rounding issue related to the light level data [253, 263].

API use in the wild should be audited after implementation

In removing the Battery Status API from Firefox, Mozilla was influenced by the paucity of legitimate uses of the API in the wild [87]. This underscores the importance of analyzing the early use of an API after deployment. The measurement work presented in this chapter (Section 5.1 and Section 5.2.2) shows that fingerprinting scripts are often early adopters of a new API. The benefits of doing an early audit are two-fold: misuses of the API that weren't found during the privacy assessment may be discovered, and any uncovered vulnerability can be fixed at the specification level before web compatibility and breakage become a concern.

In the past, fingerprinting abuse in the wild has been primarily measured by the academic research community. As research on fingerprinting starts to lose its novelty, academic researchers may lose the incentive for frequent measurements of fingerprinting abuse. As a replacement, we suggest measurement through built-in browser probes or a dedicated web crawling infrastructure run by browser vendors or privacy advocacy groups.

Specification authors should carry out privacy assessments with multiple threat models

Our case study shows how a seemingly innocuous mechanism can introduce privacy risks. The original 2012 specification of Battery Status API characterized the fingerprinting risk as "minimal", but did not include any analysis of that risk [173]. An enumeration of the possible fingerprinting approaches, even if minimal in expected effectiveness, may have helped avoid the blind spot. We recommend that if any privacy vulnerability is identified, possible exploitation should be modeled and analyzed in detail. Privacy assessment methodologies must evolve to keep abreast of the growing technical complexity of web APIs.

The case study shows the importance of assessing the data quality requirements of APIs, as unnecessarily precise data may expose users to unforeseen privacy risks. It also shows the importance of data minimization as a precaution against unexpected future misuses. Indeed, the Battery Status vulnerability served as a motivating example of these strategies in a W3C draft recommendation on browser fingerprint minimization [107].

Specification authors must also enumerate and analyze all relevant threat models. Some implementers such as the Tor browser operate under much stricter threat models. For example, most implementers may find it acceptable to reveal the user's operating system through a new API. But not the Tor Browser, as it attempts to maintain a uniform fingerprint across all devices [4].

Avoiding over-specification supports innovative privacy solutions

W3C specifications are expected to be well defined, but do allow implementers significant leeway. We recommend that standards exploit this flexibility to set a privacy floor yet leave room for innovative privacy solutions by implementers. Over-specification may have the unintended effect of rendering some privacy solutions uninteroperable with the standard or with other web features.

Indeed, implementers have employed novel strategies to mitigate the privacy risks of the Battery Status API (Section 5.2.1—*Removal of the API from browsers*). Yandex, which reports that the battery is fully charged, effectively disables the API by default. A user may choose to offer websites battery information in an explicit opt-in manner [343]. This opt-in mechanism is also used for the Vibration API [170], possibly addressing potential misuses [18], and giving users a consistent privacy control experience across APIs. In contrast, Firefox removed access to the entire API from web content, but allowed the API to be accessible internally and to addon-sdk exten-

sions. This allows the API to still be used by privileged code while preventing abuse from untrusted code.

Specifications should provide guidance for web developers, not just browser vendors

Specifications should not only identify points of privacy concern for browser vendors and other implementers, but should also provide useful guidance for web application developers when possible. Web developers are ultimately the end consumers of new features and are responsible for complying with local data protection regulations. To assist these developers, specifications should highlight if a particular feature provides sensitive data. Including this information in a specification will also assist browser vendors in properly documenting the APIs.

The Battery Status API specification currently describes several privacy risks and suggests mitigation strategies for browser vendors (Section 5.2.1—*Initial privacy improvements by browser vendors*), but does not provide recommendations for web developers. Other sensors maintained by W3C's Device and Sensors Working Group [13] have richer data sources and may pose more complex privacy threats. As such, draft specifications of the Generic Sensors API [190] and the Web Bluetooth API [332] recommend that web developers perform a privacy impact assessment prior to deploying applications which make use of these APIs. It would be even more helpful to highlight specific risks and providing concrete advice for mitigation.

A classic example of a standard that provides such detailed guidance is RFC 7231, which describes the HTTP protocol [129]. When discussing the disclosure of sensitive information in URLs, the specification states "Authors of services ought to avoid GET-based forms for the submission of sensitive data because that data will be placed in the request-target. Many existing servers, proxies, and user agents log or display the request-target in places where it might be visible to third parties. Such

services ought to use POST-based form submission instead." The warning has proved useful and necessary, but of course not sufficient. Indeed, studies have regularly found leaks of PII through HTTP GET form submissions [180, 216, 302], which underscores the need for such measurement research.

5.3 Summary

In this chapter we showed how device fingerprinting has increased in sophistication and adoption (Section 5.1). We examined the difficulty of designing APIs to resist fingerprinting, and distilled several suggestions for improving the process (Section 5.2). We also have several promising findings: privacy tools already block the majority of fingerprinting attempts (Section 5.1.7), and we show that measurement can help fill the gaps. Similarly, new fingerprinting techniques can be discovered by bootstrapping of off previously discovered scripts.

In contrast to stateful tracking (Chapter 4), device fingerprinting offers users significantly less control. To defend against stateful tracking users can clear their cookies, and browser vendors can continue to plug holes that allow trackers to work around cookie clears. However, both browser vendors and users have far fewer options to defend against device fingerprinting. Device fingerprints are already largely unique [110, 193], and each new API introduced has the potential to expand the device's fingerprinting surface. To compound that, changing APIs that are already deployed to reduce the fingerprinting surface is a difficult task [155].

In Chapter 6 we explore a related but distinct form of stateless tracking: tracking with PII-derived identifiers. Similar to device fingerprints, identifiers which are based on a user's personal information (e.g., their name or email address) are persistent and offer very limited user control. In contrast to device fingerprints, browser vendors and users have even fewer options to prevent the use of personal information for tracking.

Chapter 6

Third-party trackers collect PII

Many user interactions on the web involve the exchange of Personally Identifiable Information (PII). Users will provide their email address and name when registering for an account, or give their mailing address and credit card information when making a purchase. In this chapter we explore the ways trackers collect and use PII, both on the web and in emails. In some cases the trackers do not intend to collect PII, and rather receive it as a consequence of the structure of emails and websites. In other cases the collection in intentional; a tracking identifier based on PII is more persistent than those based on cookies (Chapter 4) or device fingerprinting (Chapter 5).

In particular, we found trackers collecting the hash of a user's email address—both in the contents of emails (Section 6.1.3) and on the web (Section 6.2.3). Email addresses are unique and persistent, and thus the hash of an email address is an excellent tracking identifier. LiveIntent, a "people-based" marketing company which we study in more detail in Section 6.1.3, extolled the value of email addresses as tracking identifiers in a blog post (emphasis ours): "As an identifier, **email is both deterministic and persistent**. That is, when a consumer gives out a verified email, it usually belongs to only that consumer. That can't be said of all typical advertising identifiers. Cookies, for example, live on desktop browsers that are often shared with

no way to distinguish who's using it. And whereas **email is cross-device**, cookies aren't." [138].

A user's email address will almost never change. As a consequence, clearing cookies, using private browsing mode, or switching devices won't prevent tracking. The hash of an email address can be used to connect the pieces of an online profile scattered across different browsers, devices, and mobile apps. In fact, we even discovered the same third party (Acxiom, a major data broker) receiving email address hashes both within our emails measurements (Section 6.1.3) and web measurements (Section 6.2.3). PII-based tracking identifiers can also serve as a link between browsing history profiles before and after cookie clears.

The persistence and ubiquitous presence of a user's PII makes defending against tracking a difficult task. Most email clients allow users to disable the loading of remote content, which prevents trackers from loading. However that causes emails to be unreadable. Some mail clients proxy remote content or block cookies, which helps reduce the privacy impact of a third-party tracker receiving a user's PII. We examine the current defenses deployed by mail clients in Section 6.1.6 and propose our own defense based on resource filtering in Section 6.1.7. Turning to the web, a user's primary option for protection remains resource blockers (Section 6.2.5). We found that most of the trackers studied in Section 6.2.4 were blocked by the major tracking protection lists—several of those that weren't at the time of measurement were later added by the list maintainers (Section 6.2.5).

Not all PII collection by trackers is intentional. As was often the case in privacy leaks studied by past research (Section 2.1.4), PII leaks occur as a result of the web architecture; email addresses stored in `Referer` headers end up shared with a bunch of unrelated third parties (Section 6.1.4), or scripts scoop up personal data alongside other page information they collect to support their services (Section 6.2.4).

In the case of session replay scripts, the trackers themselves attempt to mitigate this accidental collection, but we find these mitigations frequently fail in practice.

6.1 Trackers collect PII in emails

Email began as a non-interactive protocol for sending simple textual messages. But modern email clients support much of the functionality of the web, and the explosion of third-party web tracking has also extended to emails, especially mailing lists. Surprisingly, while there is a vast literature on web tracking, email tracking has seen little research.

The ostensible purpose of email tracking is for senders to know which emails have been read by which recipients. Numerous companies offer such services to email senders [51, 94, 160], and mail clients that have privacy features advertise them as a way for users to protect their privacy from email *senders* [139, 238, 341]. But we find that email tracking is far more sophisticated: a large network of third parties also receive this information, and it is linked to users' cookies, and hence to their activities across the web. Worse, with many email clients, *third-party trackers receive the user's email address when the user views emails.* Further, when users click links in emails, regardless of the email client, we find additional leaks of the email address to trackers.

We show that much of the time, leaks of email addresses to third parties are intentional on the part of commercial email senders. The resulting links between identities and web history profiles belie the claim of "anonymous" web tracking. The practice enables onboarding, or online marketing based on offline activity [42], as well as cross-device tracking, or linking between different devices of the same user [65]. And although email addresses are not always shared with third parties in plaintext—

sometimes they are hashed—we argue that hashing does little to protect privacy in this context (Section 6.3).

Email tracking is possible because modern graphical email clients render a subset of HTML. JavaScript is invariably stripped, but embedded images and stylesheets are allowed. These are downloaded and rendered by the email client when the user views the email (unless they are proxied by the user's email server; of the providers we studied (Section 6.1.6), only Gmail and Yandex do so). Crucially, many email clients, and almost all web browsers, in the case of webmail, send third-party cookies with these requests, allowing linking to web profiles. The email address is leaked by being encoded as a parameter into these third-party URLs.

When links in emails pointing to the sender's website are clicked, the resulting leaks are outside the control of the email client or the email server. Even if the link doesn't contain any identifier, the web browser that opens the link will send the user's cookie with the request. The website can then link the cookie to the user's email address; this link may have been established when the user provided her email address to the sender via a web form. Finally, the sender can pass on the email address—and other personally identifiable information (PII), if available—to embedded third parties using methods such as redirects and referrer headers.

6.1.1 Collecting a dataset of emails

We now describe how we assembled a large-scale corpus of mailing-list emails. We do not attempt to study a "typical" user's mailbox, since we have no empirical data from real users' mailboxes. Rather, our goal in assembling a large corpus is to study the overall landscape of third-party tracking of emails: identify as many trackers as possible (feeding into our enhancements to existing tracking-protection lists) and as many interesting behaviors as possible (such as different hashes and encodings of emails addresses).

High-level architecture of crawler.
Assemble a list of sites. For each site:

- *Find pages potentially containing forms.* For each page:
  - Find the best form on the page via *top-down form detection* and *bottom-up form detection*. If a form was found:
    * *Fill in the form*
    * *Fill in any secondary forms* if necessary
    * Once a form has been submitted, skip the rest of the pages and continue to next site

High-level architecture of server.
Receive and store email. For each email:

- *Check for and process confirmation links.*

Figure 6.1: High-level architecture of the email collection system, with the individual modules italicized.

To achieve scale, we use an automated approach. We extended OpenWPM (Section 3.1) to search for and fill in forms that appear to be mailing-list subscriptions. The crawler has five modules, and the server that processes emails has two modules. They are both described at a high level in Figure 6.1. We now describe each of the seven modules in turn.

Assemble a list of sites. Alexa maintains a public list of the top 1 million websites based on monthly traffic statistics, as well as rankings of the top 500 websites by category. We used the "Shopping" and "News" categories, since we found them more likely to contain newsletters. In addition, we visited the top 14,700 sites of the 1 million sites, for a total of 15,700 sites.

Detect and rank forms. When OpenWPM cannot locate a form on the landing page, it searches through all internal links (`<a>` tags) in the DOM until a page containing a suitable form is found. A ranked list of terms, shown in Table 6.1, is used to prioritize the links most likely to contain a mailing list. On each page, forms are detected using both a *top-down* and *bottom-up* procedure. The top-down procedure

Description	Keywords	Location
Email list registration	newsletter, weekly ad, subscribe, inbox, email, sale alert	link text
Generic registration	signup, sign up, sign me up, register, create, join	link text
Generic articles/posts	/article, news/, /2017	link URL
Selecting language/region	/us/, =us&, en-us	link URL
Blacklist	unsubscribe, mobile, phone	link text

Table 6.1: OpenWPM chooses links to click based on keywords that appear in the link text or URL. The keywords were generated by iterating on an initial set of terms, optimizing for the success of mailing list sign-ups on the top sites. We created an initial set of search terms and manually observed the crawler interact with the top pages. Each time the crawler missed a mailing list sign-up form or failed to go to a page containing a sign-up form, we inspected the page and updated the set of keywords. This process was repeated until the crawler was successful on the sampled sites.

examines all fields contained in `<form>` elements. Forms which have a higher `z-index` and more input fields are given a higher rank, while forms which appear to be part of user account registration are given a lower rank. If no `<form>` elements are found, we attempt to discover forms contained in alternative containers (e.g., forms in `<div>` containers) using a bottom-up procedure. We start with each `<input>` element and recursively examine its parents until one with a submit button is found. For further details, see *Top-down form detection* and *Bottom-up form detection* in Appendix A.5.

Fill in the form. Once a form is found, OpenWPM must fill out the input fields such that all inputs validate. The crawler fills all visible form fields, including: `<input>` tags, `<select>` tags (i.e., dropdown lists), and other submit `<button>` tags. Most websites use the general `text` type for all text inputs. We surveyed a number of top websites to determine common naming practices for input fields, and filled the fields with the data of the expected type. For example, name fields were filled with a generic first and last name. After submitting a form, we wait for a few seconds and re-run the procedure to fill follow-up fields, if required. For further details, see *Determining form field type* and *Handling two-part form submissions* in Appendix A.5.

Receive and store email. We set up an SMTP server to receive emails.[1] The server accepts any mail sent to an existing email address, and rejects it otherwise. It then parses the contents of the mail and logs metadata (such as the sender address, subject text, and recipient address) to a central database. All textual portions of the message contents are written to disk.

Check for and process confirmation links. Our server will check the first email sent to each email address to determine if the mailing list requires additional user interaction to confirm the subscription. If the initial email's subject or *rendered* body text includes the keywords "confirm", "verify", "validate", or "activate", we extract potential confirmation links from the email. For HTML emails we collect links which match these keywords along with additional lower-priority keywords "subscribe" or "click". For plain-text emails we simply choose the longest link text. Emails with the past-tense keywords "confirmed", "subscribed", and "activated" in subject lines are skipped, as are links with the text "unsubscribe", "cancel", "deactivate", and "view". If any link is found, it is visited using OpenWPM.

Form submission measurement. Our crawler discovered and attempted to submit forms on 3,335 sites. We received at least one email from 1,242 (37%) of those sites between the months of October 2016 and May 2017. To understand the types of form submission failures, we ran a follow-up measurement in August 2017 where we took screenshots of the pages before and after the initial and follow-up form submissions. We manually examined a random sample of sites on which a form submission was attempted. We summarize the results in Table 6.2.

[1]The mail server receives emails using SubEtha SMTP, a library offering a simple low-level API to handle incoming mail. The server accepts any mail sent to (`RCPT TO`) an existing email address, and rejects it otherwise. The mail contents (`DATA`) are parsed in MIME format using the JavaMail API, and the raw message contents are written to disk. MIME messages consist of a set of headers and a content body, with the required `Content-Type` header indicating the format of the content; notably, a *multipart* content body contains additional MIME message subparts, enabling messages to be arranged in a tree structure. To save disk space, we recursively scan multipart MIME messages for subparts with content types that are non-text (`text/*`), such as attached images or other data, and discard them before storing the messages since we do not examine any non-textual content.

Submission classification	% of sampled sites	
Total successful submissions		38%
→Mailing lists subscription	32%	
→User account registration	6%	
Failed: required a CAPTCHA		16%
Failed: unsupported form fields		25%
Unable to classify via screenshots		21%

Table 6.2: Submission success status of a sample of 252 of the 3,335 form submissions made during the sign-up crawl. The success and failure classification was determined through a manual review of screenshots taken before and after an attempted form submission.

When filling forms, OpenWPM will interact with user account registration forms, mailing list sign-up forms, and contact forms. The successful submissions were mostly mailing list sign-ups and a small number of user account registrations, which are included as they can be tied to a mailing list. The failed submissions were mostly caused by forms other than mailing lists. In fact, more than 70% of the failures caused by a CAPTCHA or unsupported field were not mailing list form submissions. Overall, only 11% of the sampled mailing list interactions resulted in a CAPTCHA. Since our primary focus is mailing lists, we leave the evaluation of complex and CAPTCHA-protected forms to future work.

Email corpus. The assembled corpus contains a total of 12,618 HTML emails from 902 sites. We received an average of around 14 emails per site and a median of 5. A few sites had very active mailing lists, with 20 sites sending over 100 emails during the test period (October 2016 and May 2017). We observe that we received no spam, which we confirmed both by manual inspection of a sample of emails as well as by finding an exact one-to-one correspondence between the 902 senders in our dataset and the unique email addresses that we generated. This ensures that the results represent the behavior of the sites where we registered, rather than spammers.

6.1.2 Measurement methods

To measure the extent of tracking in emails we had two main tasks: simulating a user who opens an email in a mail client, and simulating a user who clicks a link in an email which opens in a web browser. We are able to automate both tasks with OpenWPM using the methods outlined in this section.

Simulating a webmail client. To measure web tracking in email bodies we render the emails using a simulated webmail client in an OpenWPM instance. Many webmail clients remove a subset of HTML tags from the email body to restrict the capabilities of rendered content. In particular, Javascript is exclusively removed, while iframe tags and CSS [16] have mixed support. We simulate a permissive webmail client, one which disables Javascript and removes the `Referer` header from all requests, but applies no other restrictions to the rendered content.

The email content is served on `localhost`, but is accessed through the domain `localtest.me` (which resolves to `localhost`) to avoid any special handling the browser may have for the local network. We classify remote content requests as third-party using the method outlined in Section 3.2.1. We configure OpenWPM to run 15 measurement instances in parallel. Each email is loaded twice in its own measurement instance: once with a fresh profile, and then again keeping the same browser profile after sleeping for 10 seconds. This is intended to allow remote content on the page to load both with and without browser state present. Indeed we observe some tracking images which redirect to new domains upon every subsequent reload of the same email.

Sampling links from emails. To evaluate the privacy leaks which occur when links in emails are clicked, we generate a dataset from the HTML content of all emails and visit them individually in an instrumented browser. To extract the links from mail content, we parse all email bodies with `BeautifulSoup` [2] and extract the `src` property of all `<a>` tags. We sample up to 200 unique links per sender using

the following sampling strategy. First, we bin links across all emails from a sender by the `PS+1`[2] and `path` of the link. Next, we sample one link from each bin without replacement until there are no more links or we reach a limit of 200. This helps ensure that we have as diverse a set of landing pages as possible by stripping fragment and query string identifiers that may not influence the landing page.

Simulating link clicks. To simulate a user clicking a link, we visit each link in an OpenWPM instance using a fresh browser profile. The browser fully loads the page and sleeps for 10 seconds before closing. Unlike the email viewing simulation, we enable both Javascript and `Referer` headers. This simulation replicates what happens when a link is clicked in a standalone email client; only the URL of the clicked link is passed to the browser for handling. In a webmail client, the initial request resulting from the click may also contain a cookie and a `Referer` header containing the email client's URL. We do not simulate these headers in our crawl.

Classifying email leakage. We detect leaked email addresses using the method outlined in Section 3.2.5. Email leaks may not be intentional. If an email address is included in the query string or path of a document URL it may automatically end up in the `Referer` header of subsequent requests from that document. Requests which result in a redirect also often add the referrer of the previous request to the query string of the new request. In many instances this happens irrespective of the presence of an email address in the original request. The situation is made more complex on the web since third-party Javascript can dynamically build URLs and trigger requests.

The reduced HTML support and lack of Javascript execution in email clients makes it possible to determine intentionality for most leaks. When an email is rendered, requests can result from three sources: from elements embedded in the original

[2]For a definition of `PS+1` see Section 3.2.1.

HTML, from within an embedded iframe (if supported by the client), or from a redirected request.

1. If a leak occurs in a `Referer` header it is **unintentional**. For webmail clients the `Referer` header (if enabled) will be the client itself. A mail sender can embed an iframe which loads a URL that includes the user's email address, with the explicit intention that the user's email leak to third parties via the `Referer` header. However, we chose not to include this possibility because email senders have multiple direct options for sharing information with third parties that do not rely on the sparsely supported iframe tag.

2. If a leak occurs in a request to a resource embedded directly in the HTML of the email body (and is not the result of a redirect) it is **intentional**. We can determine intentionality since any request resulting from an HTML document must have been constructed by the email sender. Note that this does not hold for web documents, since embedded Javascript can dynamically construct requests during the page visit.

3. If a request results from a redirect, the party responsible for the leak is the party whose request (i.e., the *triggering URL*) responded with a redirect to the new location (i.e., the *target URL*). We classify a leak as **intentional** if the leaked value is hashed between the triggering URL and the target URL, or if there are more encodings or hashes of the leaked value included in the target URL than in the triggering URL. If the target URL includes a full copy of the triggering URL (in any encoding) the leak is **unintentional**. All other cases are classified as **ambiguous**, such the case where a target URL includes only the query string of the triggering URL.

In the case of link clicks, we needed to detect email leakage from content rendered in a web browser. Unlike email views, the `Referer` header was enabled for those

measurements. As such, we consider a party to have received a leak if it is contained either in the URL or the `Referer` header of the resource request to that party. Email addresses can also be shared with the party through the `Cookie` header, request `POST` bodies, websocket connections, WebRTC connections, and so on. We consider these out of scope for this analysis.

Measuring blocked tags in rendered emails. Tracking protection tools which block resource requests offer users protection against the tracking embedded in emails. We evaluate the effectiveness of these tools by checking the requests in our dataset against two major blocklists: EasyList and EasyPrivacy [5]. These lists block advertisement and tracking related requests, and are bundled with several popular blocking extensions, including AdBlock Plus [1] and uBlock Origin [10]. We use the BlockListParser library [3] to determine if a request would have been blocked[3] by an extension utilizing these lists. We expand upon our tracker detection method for web data (Section 3.2.2), and classify a request as blocked if it matches any of the following three conditions:

1. The request directly matches the filter list
2. The request is the result of a redirect and any request earlier in the redirect was blocked.
3. The request is loaded in an iframe and the iframe document request (or any resulting redirect) was blocked.

It is possible to do this classification in an offline fashion because of the lack of Javascript support in email clients. This removes the need to run measurements with one of the aforementioned extensions installed. In environments that support Javascript, content can be loaded dynamically and as the result of interactions be-

[3]We set the parser options as we would expect them to be set for a request occurring in a webmail client. For example, all requests are considered third-party requests.

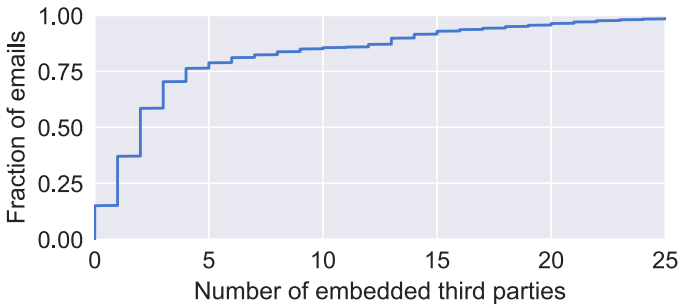


Figure 6.2: CDF of third parties per email, aggregating data across the initial viewing and re-opening of an email. In addition, 1.4% of emails have between 25 and 53 third parties.

tween several scripts. In such an environment it is much more difficult to determine which requests would have been blocked for every script appearing on the block list.

6.1.3 Privacy leaks when viewing emails

Remote resources embedded in email content can track users across emails. As we show in our survey of email clients (Section 6.1.6), many email clients allow remote resources to set persistent cookies and send those cookies with resource requests. In total, we found that 10,724 of the measured emails (85%) embedded resources from at least one third party, with an average of 5 third parties per email. The distribution of embedded third parties is far from uniform; we found a median of two per email and a small number of emails embedding as many as 50 third parties (Figure 6.2).

Table 6.3 shows the top third-party domains present in email at the time of measurement. Many of these parties also have a large presence on the web (Section 4.1.3), blurring the line between email and web tracking. On webmail clients, requests to these cross-context third parties will use the same cookies[4], allowing them to track both a user's web browsing and email habits. In total, the emails visited during our crawls embed resources from 879 third parties.

[4]The availability of cookies in a webmail client depends on both the user's browser and whether the mail provider proxies remote content. See Section 6.1.6 for more detail.

Domain	% of Emails	% of Top 1M
doubleclick.net	22.2	47.5
mathtag.com	14.2	7.9
dotomi.com	12.7	3.5
adnxs.com	12.2	13.2
tapad.com	11.0	2.6
liadm.com	11.0	0.4
returnpath.net	11.0	¡0.1
bidswitch.net	10.5	4,9
fonts.googleapis.com	10.2	39.4
list-manage.com	10.1	¡0.1

Table 6.3: Top third-party domains by percentage of the 12,618 emails in the corpus. For comparison, we show the percentage of the top 1 million websites on which these third parties are present.

Leaks of email addresses to third parties are common

In addition to being able to track email habits, 99 third parties (11%) also gain access to a user's email address, whether in plaintext or hashed. In email clients which support cookies, these third parties will receive the email address alongside any cookies they've set on the user's device. Trackers which are also present on the web will thus be able to link this address with the user's browsing history profile.

Around 19% of the 902 senders leaked the user's email address to a third party in at least one email, and in total 29% of emails contain leaks to third parties. We find that a majority of these leaks, 62% of the 100,963 leaks to third parties, are intentional. These intentional leaks mostly occur through remote content embedded directly by the sender. Furthermore, 1% of leaks are classified as unintentional with the remainder considered ambiguous. While we do not attempt to determine how these identifiers are being used, plaintext and hashed emails can be used for persistent tracking, cross-device tracking, and syncing information between parties.

The leaked addresses are often hashed. Although we can detect email addresses hashed with 24 different functions and up to three nested layers, we only find `MD5`, `SHA1`, and `SHA256` in frequent use. Table 6.4 summarizes the number of senders and

Leak	# of Senders	# of Recipients
MD5	100 (11.1%)	38 (38.5%)
SHA1	64 (7.1%)	19 (19.2%)
SHA256	69 (7.6%)	13 (13.1%)
Plaintext Domain	55 (6.1%)	2 (2.0%)
Plaintext Address	77 (8.5%)	54 (54.5%)
URL Encoded Address	6 (0.6%)	8 (8.1%)
SHA1 of MD5*	1 (0.1%)	1 (1.0%)
SHA256 of MD5*	1 (0.1%)	1 (1.0%)
MD5 of MD5*	1 (0.1%)	1 (1.0%)
SHA384	1 (0.1%)	1 (1.0%)

Table 6.4: Email address leakage to third parties by encoding. Percentages are given out of a total of 902 senders and 99 third-party leak recipients. All hashes are of the full email address. Email "domain" is the part of the address after the "@".
*These appear to be a misuse of LiveIntent's API (Section 6.1.3).

receivers of each encoding. The relatively low diversity of hashes and encodings suggests that these techniques are not being used to obfuscate the collection of email addresses. In fact, the query parameters which contain hashed emails sometimes identify the hash functions used in the parameter name (e.g., a string like `?md5=<md5 hash of email>` appearing in the HTTP request). The design of APIs like LiveIntent's, which first receives an email address and then syncs with a number of other parties (Section 6.1.3), suggests that these hashed address may be used to share or link data from multiple parties.

Table 6.5 identifies the top organizations[5] which received leaked email addresses. This shows that email address collection from emails is largely consolidated to a few major players, which are mostly distinct from the popular web trackers. In fact, only one of the top 10 organizations, Neustar, was found in the top 20 third-party organizations on the top 1 million websites in January 2016 (Section 4.1.3). Also surprising is the prevalence of leaks to IP addresses, which accounted for eight of the top 20 domains receiving email addresses. This may be due to the relatively

[5]We map domains to organizations using the classification provided by Libert [201], adding several new email-specific organizations. When an organization could not be found, we use the PS+1.

Recipient Organization	# of Senders
LiveIntent	68 (7.5%)
Acxiom	46 (5.1%)
Litmus Software	28 (3.1%)
Conversant Media	26 (2.9%)
Neustar	24 (2.7%)
apxlv.com	18 (2.0%)
54.211.147.17	18 (2.0%)
Trancos	17 (1.9%)
WPP	17 (1.9%)
54.82.61.160	16 (1.8%)

Table 6.5: Top organizations which received email address leaks by number of the 902 total senders in our dataset. A domain is used in place of an organization when it isn't clear which organization it belong to.

ephemeral nature of newsletter emails, which removes concerns of IP address churn over time.

Reopening emails brings in new third parties

Despite the lack of Javascript support, email views are dynamic. The email content itself is static, but any remote resources embedded in it may return different responses each time the email is viewed, and even redirect to different third parties. To examine the effects of this, we load every email first with a "clean" browser profile and then again without clearing the profile. Surprisingly, the average Jaccard similarity [288] between the sets of third parties loaded during the first and second views of the same email is only 60%.

The majority of emails—two-thirds—load fewer third parties when the email is reopened compared to the initial view. However, about 21% of emails load at least one resource when an email is reopened that wasn't present the first time. A small number of third parties are disproportionately responsible for this—they load different sets of additional third parties each time the email is opened (Table 6.6).

Redirecting Party	Organization	Avg add'l parties	#S	#E
pippio.com	Acxiom	5.7	7	32
liadm.com*	LiveIntent	3.7	68	1097
rlcdn.com	Acxiom	1.7	11	551
imiclk.com	MediaMath	1.3	2	4
mathtag.com	MediaMath	1.1	11	382
alcmpn.com	ALC†	0.8	6	132
emltrk.com	Litmus	0.7	41	638
acxiom-online.com	Acxiom	0.4	2	33
dyneml.com	PowerInbox	0.1	3	13
adnxs.com	AppNexus	0.1	19	277

Table 6.6: Top parties by average number of new third-party resources in a redirect chain when an email was reloaded. The number of senders (# S) out of 902 total and the number of emails (#E) out of 12,618 total on which this occurred is given for each redirecting party. We exclude redirecting parties that only exhibited this behavior in emails from a single sender. In total, there are 12 parties which exhibited this type of redirect behavior.
* Includes statistics for chains which redirected to `http://p.liadm.com/imp` in the first redirect. We observed a common pattern of URLs of the form `li.firstparty.com` redirecting first to this endpoint which then redirected to a number of other third parties.
† American List Counsel

The number of leaks between email loads stays relatively constant, with less than 50 emails leaking to new parties on the second load[6]. However, as the comparison of Table 6.6 with Table 6.5 shows, many of the top leak recipients are also responsible for redirecting to the highest number of new parties. Thus, reloading an email increases the number of potential recipients of a leak if the redirectors share data based on the email or email hash they receive.

Case study: LiveIntent

LiveIntent received email addresses from the largest number of senders, 68 in total. In this section we analyze a sample of the request chains that resulted in leaks to LiveIntent. Table 6.7 shows an example redirect chain of a single pixel embedded in

[6]We exclude leaks which occur to a different IP address on the second load. This occurs in 349 emails, but is less meaningful given the dynamic nature of IP address.

Row	Request URL
0	http://inbox.washingtonexaminer.com/imp?[...]&e=<EMAIL>&p=0
1	http://p.liadm.com/imp?[...]&m=<MD5(address)>&sh=<SHA1(address)>&sh2=<SHA256(address)> &p=0&dom=<EMAIL_DOMAIN>
2	http://x.bidswitch.net/sync?ssp=liveintent&bidder_id=5298&licd=3357&x=EGF.M[...]
3	http://x.bidswitch.net/ul_cb/sync?ssp=liveintent&bidder_id=5298&licd=3357&x=EGF.M[...]
4	http://p.adsymptotic.com/d/px/?_pid=12688&_psign=d3e69[...]&bidswitch_ssp_id=liveintent&_redirect=[...]
5	http://p.adsymptotic.com/d/px/?_pid=12688&_psign=d3e69[...]&bidswit[...]&_redirect=[...]&_expected_cookie=[...]
6	http://x.bidswitch.net/sync?dsp_id=126&user_id=84f3[...]&ssp=liveintent
7	http://i.liadm.com/s/19751?bidder_id=5298&licd=3357&bidder_uuid=<UUID_1>
8	http://cm.g.doubleclick.net/pixel?google_nid=liveintent_dbm&google_cm&google_sc
9	http://cm.g.doubleclick.net/pixel?google_nid=liveintent_dbm&google_cm=&google_sc=&google_tc=
10	http://p.liadm.com/match_g?bidder_id=24314&bidder_uuid=<UUID_2>&google_cver=1
11	http://x.bidswitch.net/sync?ssp=liveintent&bidder_id=5298&licd=
12	http://pool.udsp.iponweb.net/sync?ssp=bidswitch&bidswitch_ssp_id=liveintent

Table 6.7: Redirect chain from a LiveIntent Email Tracking Pixel found in our dataset. URL query strings are truncated for clarity (using `[...]`).

an email from the `washingtonexaminer.com` mailing list. The initial request (row 0) was to a subdomain of `washingtonexaminer.com`, and included the user's plaintext email address in the `e=` query string parameter. The domain redirected to `liadm.com` (row 1), a LiveIntent domain, and includes the `MD5`, `SHA1`, and `SHA256` hashes of the email address in the parameters `m=`, `sh=`, and `sh2=`. The URL also includes the domain portion of the user's address.

In rows 2 - 12, the request redirected through several other domains and back to itself, exchanging what appear to be partner IDs and bidder IDs. In rows 7 and 10 LiveIntent received a UUID from the domain in the previous request, which could allow it to exchange information with those trackers outside of the browser.

Request blockers help, but don't fix the problem

Privacy conscious users often deploy blocking extensions, such as uBlock Origin, Privacy Badger, or Ghostery, to block tracking requests. Since webmail clients are browser-based, these blocking extensions can also filter requests that occur while displaying email content[7]. We use our blocked tag detection methodology (Section 6.1.2) to determine which resources would have been blocked by the popular EasyList and

[7]Thunderbird supports most of the popular Firefox extensions, and as such Thunderbird users can also deploy these defenses. See Table 6.12 for more details.

Encoding	# of Senders	# of Recipients
Plaintext Address	34 (3.7%)	34 (66.7%)
MD5	21 (2.3%)	12 (23.5%)
SHA1	14 (1.6%)	6 (11.8%)
URL Encoded Address	4 (0.4%)	4 (7.8%)
SHA256	4 (0.4%)	2 (3.9%)
SHA384	1 (0.1%)	1 (2.0%)

Table 6.8: Encodings used in leaks to third parties after filtering requests with EasyList and EasyPrivacy. Totals are given out of 902 email senders and 51 third-party leak recipients in our dataset.

EasyPrivacy blocklists. We then examine the remaining requests to determine how frequently email addresses continue to leak.

Overall, the blocklists cut the number of third parties receiving leaked email addresses from any sender nearly in half, from 99 to 51. Likewise, the number of senders which leak email addresses in at least one email was greatly reduced, from 19% to just 7%. However, as Table 6.8 shows, a significant number of leaks of both plaintext and email hashes still occur. In Table 6.9 we see that there are still several third-party domains which receive email address leaks, despite blocking. Several of these domains are known trackers which could be included in the blocklists. In addition, IP addresses and CDN domains are still recipients of leaked email addresses. Blocking on other URL features, such as the URL path, could help reduce leaks to these domains.

6.1.4 Privacy leaks when clicking links in emails

In Section 6.1.3 we explore the privacy impact of a user opening and rendering an email. In this section we explore the privacy impact of a user clicking links within an email. Once a user clicks a link in an email, the link is typically opened in a web browser. Unlike email clients, web browsers will typically support Javascript and advanced features of HTML, creating many potential avenues for privacy leaks. However, the only way an email address can propagate to a page visit is through the direct embedding of the address in a link contained in the original email body.

Recipient Domain	# of Senders
mediawallahscript.com	7
jetlore.com	4
scrippsnetworks.com	4
alocdn.com	3
richrelevance.com	3
ivitrack.com	2
intentiq.com	2
gatehousemedia.com	2
realtime.email	2
ziffimages.com	2

Table 6.9: The top third-party leak recipient domains in our dataset after filtering requests with EasyList and EasyPrivacy. All recipients received leaks from less than 1% of the 902 senders studied.

Recipient Organization	# of Senders
Google	247 (27.4%)
Facebook	160 (17.7%)
Twitter	94 (10.4%)
Adobe	81 (9.0%)
Microsoft	73 (8.1%)
Pinterest	72 (8.0%)
LiveIntent	69 (7.6%)
Akamai	69 (7.6%)
Acxiom	68 (7.5%)
AppNexus	61 (6.8%)

(a) The top leak recipient organizations.

Recipient Domain	# of Senders
google-analytics.com	200 (22.2%)
doubleclick.net	196 (21.7%)
google.com	159 (17.6%)
facebook.com	154 (17.1%)
facebook.net	145 (16.1%)
fonts.googleapis.com	102 (11.3%)
googleadservices.com	96 (10.6%)
twitter.com	94 (10.4%)
googletagmanager.com	87 (9.6%)
gstatic.com	78 (8.6%)

(b) The top leak recipient domains.

Table 6.10: The top leak recipients based on a sample of simulated link clicks. All values are out of 902 total senders in our dataset.

We found that about 11% of links contain requests that leak the email address to a third party. About 12% of all emails contain at least one such link, and among this subset, there are an average of 3.5 such links per email. The percentage of the 902 senders that leak the email address in at least one link in one email is higher: 35.5%. Finally, there were over 1,400 distinct third parties that received the email address in one or more of our simulated link clicks. We expect that all statistics in this paragraph, except the first, are slight underestimates due to our limit of 200 links per sender.

Table 6.10a shows the top organizations that receive leaked email addresses, and Table 6.10b shows the top domains. Over a quarter of senders leaked the email address to Google in at least one link.

The most striking difference between these results and the corresponding results for viewing emails is that these lists look very similar to the list of top third party trackers (Section 4.1), with the addition of a small number of organizations specific to email tracking. This motivates the privacy concern that identities could potentially be attached to third-party web tracking profiles.

6.1.5 Evaluation of email tracking defenses

Defenses against tracking can be employed by several parties. We ignore mail senders and trackers themselves, since email tracking is a thriving commercial space and our evidence suggests that senders by and large cooperate with trackers to leak email addresses. We instead focus on parties who have an incentive to protect the recipient's privacy, namely the recipient's mail server, mail user agent, and the web browser.

The lines between these roles can be blurry, so we illustrate with two examples. Consider a user reading Yahoo mail via Firefox. The email server is Yahoo, the email client is Firefox together with Yahoo mail's client-side JavaScript, and the web browser is again Firefox. Or consider a user reading her university mail, via Gmail's IMAP feature, on her iPhone. For our purposes, both the university and Gmail count as email servers, since either of them is in a position to employ defenses. The email client is the Gmail iOS app, and the web browser is Safari. Table 6.11 summarizes the applicability of various defenses to the three roles. We discuss each in turn.

Content proxying. Email tracking is possible because of embedded content such as images and CSS (cascading style sheets). To prevent this, some email servers, notably Gmail, proxy embedded content. Thus, when the recipient views the email, the mail user agent does not make any requests to third parties.

Defense	Email server	Email client	Web browser
Content proxying	X		
HTML filtering	X	X	
Cookie blocking		X	X
Referrer blocking	X	X	X
Request blocking		X	X

Table 6.11: Applicability of each of the five possible defenses to each of the three contexts in which they may be deployed. An X indicates that the defense is applicable.

This defense doesn't prevent the recipient email address being leaked to third parties, since it is leaked by being encoded in the URL. In fact, it *hinders* efforts by the mail client to prevent email address leakage (see request blocking below). However, it prevents third parties from learning the user's IP address, client device properties, and when the email was read (depending on how the proxy is configured). Most importantly, it prevents the third-party cookie from being sent, and thus prevents the third party from linking the user's email address to a tracking profile. In this way it is a complement to cookie blocking.

This defense can be deployed by the email server. Conceivably the email client might have its own server component through which embedded resources are proxied, but no email clients currently work this way, and further, it would introduce its own privacy vulnerabilities, so we ignore this possibility.

HTML filtering. HTML filtering refers to modifying the contents of HTML emails to mitigate tracking. It may be applied by the email server or the client, but it is more suitable to the server since the client can generally achieve the same effect in other ways, e.g., by request blocking or modifying the rendering engine. It is rarely applied today, and only in minimal ways. In Section 6.1.7 we prototype a comprehensive HTML filtering technique.

HTML filtering modifies the content of the email body, and thus might interfere with some email authentication methods, notably Domain Keys Identified Email (DKIM). However, since filtering is carried out by the recipient's mail server (Mail

Transfer Agent) and not by intermediate mail relays, filtering can be done after the signature has been verified, and thus there is no impact on email authentication.

The following three techniques are applicable in one of two scenarios: when the email client requests embedded resources, or when the web browser handles clicks on links in emails.

Cookie blocking. Cookie blocking in the email client prevents third-party cookies from being sent when embedded content is requested. It is especially relevant in the webmail context, where the cookie allows third parties to link an email address to a web browsing profile. Even otherwise, blocking cookies is helpful since it makes it harder for third parties to compile a profile of the recipient's email viewing (they can always do this for the subset of emails where the email address is leaked).

Referrer blocking. If the email client sends the `Referer` header when loading embedded resources, it can allow several types of leaks. Depending on the implementation, the referrer may encode which client is being used and which specific email is being read. If the recipient forwarded an email to someone else and the email is being viewed in a different user's mailbox, it could leak this information. Worse, if the client supports iframes in emails, and the email address happens to be in the iframe URL, all requests to resources embedded in that iframe will accidentally leak the email address. For all these reasons, referrer blocking is a privacy-enhancing measure. There is little legitimate use for the referrer header in the context of email. While clients can certainly block the header (as can web browsers), servers can do this as well, by rewriting HTML to add the `rel='noreferrer'` attribute to links and inserting a Referrer Policy via the `meta` tag.

Request blocking. Request blocking is a powerful technique which is well known due to ad blockers and other browser privacy extensions. It relies on manually compiled *filter lists* containing thousands of regular expressions that define third-party content to be blocked. The most widely used ad-blocking list is EasyList, and the

Mail Client	Platform	Proxies Content	Blocks Images	Blocks Referrers	Blocks Cookies	Ext. Support
Gmail	Web	Yes	No*	L: Yes, I: Yes†	Yes†	Yes
Yahoo! Mail	Web	No	Yes	L: Yes, I: No	No	Yes
Outlook Web App	Web	No	Yes	No	No	Yes
Outlook.com	Web	No	No*	No	No	Yes
Yandex Mail	Web	Yes	No*	L: Yes, I: Yes†	Yes†	Yes
GMX	Web	No	No*	No	No	Yes
Zimbra	Web	No	Yes	No	No	Yes
163.com	Web	No	No*	No	No	Yes
Sina	Web	No	No	No	No	Yes
Apple Mail	iOS	No	No*	Yes	Yes	No
Gmail	iOS	Yes	No	Yes	Yes	No
Gmail	Android	Yes	No	Yes	Yes	No
Apple Mail	Desktop	No	No*	Yes	Yes	No
Windows Mail	Desktop	No	No*	Yes	No	No
Outlook 2016	Desktop	No	Yes	Yes	No	No
Thunderbird	Desktop	No	Yes	Yes	Optional	Yes

Table 6.12: A survey of the privacy impacting features of email clients as of May 2017. We explore whether the client proxies image requests, blocks images by default, blocks referrer headers from being sent (with image requests "I:" and with link clicks "L:"), blocks external resources from settings cookies, and whether or not the client supports request blocking extensions — either through the browser (for web clients) or directly (in the case of Thunderbird).
*Images are only blocked for messages considered spam.
† Blocking occurs as a result of proxied content.

most widely used tracker-blocking list is EasyPrivacy. Filter list based blocking introduces false positives and false negatives [346], but the popularity of ad blocking suggests that many users find the usability trade-off to be acceptable. While request-blocking extensions are supported primarily by web browsers, some email clients also have support for them, notably Thunderbird.

6.1.6 Survey of tracking prevention in email clients

We built an email privacy tester to discover which defenses are deployed by which popular email servers and clients.[8] Browser support for tracking protection has been extensively studied elsewhere [225], so we do not consider it here.

[8] `https://emailtracking.openwpm.com/`

The email privacy tester allows a researcher to enter an email address and the name of an email client, and then sends an email to that address containing a tracking image and a link. The image and the link both have unique URLs. The researcher views the email in the specified email client, and then clicks on the link. The server records the following information: the email address, the email client, the IP address, timestamp, and headers sent for both the image and the link requests. The list of headers includes the cookie, referrer, and user agent.

In May 2017 we created accounts with a total of 9 email providers and tested them with a total of 16 email clients using various devices available in our lab. We analyzed the data recorded by the email privacy tester, and summarize the results in Table 6.12. We found that if defenses are deployed by email servers at all, they are only enabled for specific email clients (typically the default webmail client). Therefore we do not report on servers separately, but instead fold it into the analysis of clients. We also found that HTML filtering in a general form is not deployed, but only in the limited form of image and referrer blocking, so we report on that instead. We summarize our findings in Table 6.12.

6.1.7 Our proposed defense against email tracking

We argue that tracking protection should be at the center of a defensive strategy against email tracking. It can be employed either via HTML filtering on the server or via request blocking on the client. Tracking protection (and ad blocking) based on filter lists has proven to be effective and popular in web browsers, and its limitations manageable. The other defenses we examined all have serious drawbacks: for example, content proxying comes at a cost to the email server and makes email leaks *worse*, and cookie blocking is at best a partial solution. We propose to improve tracking protection in two ways: server-side content filtering and improving client-side tracking protection lists.

Server-side email content filtering. First, we prototype a server-side HTML filtering module. We use the existing, standard EasyList and EasyPrivacy filter lists. Our filtering script is written in Python using the BlockListParser library [3]. It scans for any HTML content (`text/html`) in email bodies, parses those contents, identifies embedded resources (images or CSS) whose URLs match one of the regular expressions in the filter lists, strips them out, and rewrites the HTML.

To test the effectiveness of HTML filtering, we ran our leak detection procedure on the filtered corpus of emails. We exclude one sender due to a measurement issue. We found that 11.0% of senders will leak email addresses to a third party in at least one email, and 11.5% of emails contain embedded resources which leak email to a third-party. Overall, 62 third parties received leaked email addresses, down from 99. As tracking-protection lists improve (see below), we can expect these numbers to decrease further. These numbers are very close to the corresponding numbers for request blocking (Section 6.1.3). The two techniques aren't identical: the one difference is that in static files, filtering is limited to the URLs present in the body of the HTML and will miss those that result from a redirect. However, this difference is small, and we conclude that HTML filtering is essentially as effective as request blocking.

Note that webmail users can already deploy tracking protection, but server-side deployment will help all users, including those who use email clients that don't support request-blocking extensions.

Filling gaps in tracking-protection lists. As a second line of defense, we use our dataset to identify a list of 27,125 URLs representing 133 distinct parties which contain leaks of email addresses, but which aren't blocked by EasyList or EasyPrivacy. These include first parties in addition to third parties. We are able to identify first-party tracking URLs by observing groups of URLs of similar structure across different first-party domains. For example, 51 email senders leaked the user's email address

to a URL of the form `li.[public suffix + 1]/imp`, which appears to be part of LiveIntent's API (Section 6.1.3). We summarize the most common structures in the leaking URLs missed by tracking protection lists in Table 6.13.

URL Pattern	# of Senders
`li.[PS+1]/imp`	51 (5.7%)
`partner.[PS+1]/`	7 (0.7%)
`stripe.[PS+1]/stripe/image`	4 (0.4%)
`p.[PS+1]/esp/open`	4 (0.4%)
`api.[PS+1]/layouts/section[N]`	4 (0.4%)
`[PS+1]/customer-service`	3 (0.3%)
`mi.[PS+1]/p/rp`	3 (0.3%)
`dmtk.[PS+1]/`	3 (0.3%)
`links.[PS+1]/e/open`	3 (0.3%)
`eads.[PS+1]/imp`	3 (0.3%)

Table 6.13: The top URL patterns from URLs which leak email addresses and are missed by tracking protection lists (Section 6.1.3). The patterns are generated by stripping request URLs to hostname and path, replacing the public suffix plus one with `[PS+1]`, replacing integers with `[N]`, and stripping the last portion of the path if it ends with a file extension. The patterns are ranked by the number of senders which make at least one leaking request matching that pattern in any of the sender's emails. All values are given out of the total of 902 senders studied.

We suspect that the reason so many trackers are missed is that many of them are not active in the regular web tracking space. We have made the list of leaking URLs missed by tracking protection lists publicly available.[9] It should be straightforward to add regular expressions to filter lists based on these URLs; we suggest that filter list creators should regularly conduct scans of email corpora to identify new trackers.

6.1.8 Limitations

We mention several limitations of the work presented in this section. First, despite the large number of heuristics that went into identifying and submitting forms, it is a fundamentally hard problem, and our crawler fails in many cases, including pages requiring complex mouse interactions, pages containing very poorly structured HTML,

[9] `https://gist.github.com/englehardt/6438c5d775ffd535b317d5c6ce3cde61`

and CAPTCHA-protected form submission pages. Moreover, it is difficult to programmatically distinguish between successful and failed form submissions. Looking at received network data is impractical, since responses could easily include text for both success and failure messages. On the other hand, looking only at changes in the rendered text on the webpage is more feasible, but would require handling many possible edge cases (e.g., page redirects, alerts, pop-up windows, iframes) and might still be too unreliable to use as a metric for success.

Second, our corpus of emails is not intended to be representative, and we are unable to draw conclusions about the extent of tracking in the typical user's mailbox.

Third, our simulation of a user viewing emails assumes a permissive user agent. We expect that this closely approximates a webmail setup with default browser settings (on browsers except Safari, which blocks third-party cookies by default), but we have not tested this assumption.

6.2 Trackers collect PII on the web

The vast majority of websites today embed one or more third-party scripts in a first-party context. This practice is fundamentally insecure, because it negates the protections of the Same-Origin Policy and gives such scripts access to virtually all sensitive data on the page (see Section 2.1). Our goal in this section is to highlight the way in which third parties can and have used these privileges to access sensitive user data.

Specifically, we examine two distinct attacks. These attacks do not exploit bugs in the browser or first-party code. Rather, their existence is an inevitable consequence of loading untrusted JavaScript in a first-party origin.

1. **Autofill exfiltration.** Many web browsers automatically fill in values for known form fields. Once a user's credentials have been filled, they are vul-

nerable to exfiltration by any script present on the page. This attack is known to browser vendors, but only some browsers have mitigations in place, and they are imperfect. In fact, scripts may actually insert forms into pages that are invisible to the user as bait for autofill.

2. **Whole-DOM exfiltration.** Some third party scripts serialize the entire DOM (Document Object Model, the tree of objects that constitutes the web page) and send it to their servers. These are providers of analytics services that analyze user clicks and other interactions on the page.

In both attacks we observe the collection of sensitive user data by third-party scripts. In some cases, that data appears to be used for cross-site tracking.

6.2.1 Measurement configuration

To study each attack, we crawled 50,000 sites from the Alexa top 1 million. We used the following sampling strategy: visit all of the top 15,000 sites, randomly sample 15,000 sites from the Alexa rank range [15,000 100,000), and randomly sample 20,000 sites from the range [100,000, 1,000,000). This combination allowed us to observe the attacks on both high and low traffic sites. On each of these 50,000 sites we visited 6 pages: the front page and a set of 5 other pages randomly sampled from the internal links on the front page. All measurements were taken between June and November 2017 on Amazon EC2 servers.

6.2.2 Measurement methods

Our core measurement contribution is the development of a *bait* technique, which allows us to inject sensitive user data into the context of real websites in such a way that third-party scripts can access and exfiltrate the data without further interaction. We do not attempt to simulate a real user's interaction with a site—which would

require us to perform tasks that are difficult to automate, such as registering for an account—and instead spoof that our measurement instance has already interacted with the site. As a concrete example, we use OpenWPM's browser extension to spoof that our measurement instances have saved user credentials for every site we visit.

Our work is conceptually similar to past studies that have detected privacy leaks to third parties in real user data [277,318], but our decision to simulate user data has several distinct advantages. By simulating user data we are able to collect detailed measurements without the risk of compromising user privacy. Our technique also limits the detection of benign data collection flows, such as a first party sending data to a third-party partner when the user finishes a login. A consequence of this is that we may bait sites in ways that a real user interaction would never create, such as saving user credentials on a site that has no account support. This is not a problem for our measurements however, as our goal is to detect third parties which collect data from any pages which embed them.

The bait technique has three core components: (1) injecting sensitive data into the page context, (2) monitoring access to the data and attributing access to a specific third party, and (3) detecting transmission of the data to a third-party server. We extend OpenWPM to carry out the first two components, and provide details on how we do so in the attack sections below. The third component, data exfiltration, is detected using the method detailed in Section 3.2.5.

6.2.3 Browser login managers are vulnerable to abuse

Mechanism. All major browsers have built-in login managers that save and automatically fill in username and password data to make the login experience more seamless. The set of heuristics used to determine which login forms will be autofilled varies by browser, but the basic requirement is that a username and password field be available.

Browser	Engine	Autofill Support (2017)	Current Autofill Support (2018)
Firefox	Gecko	immediate	immediate (considering change [67])
Safari	WebKit	immediate	requires user interaction with form [44]
Chrome	Blink	u: immediate p: interaction with page	u & p: interaction with page [90]
Edge	EdgeHTML	immediate	immediate

Table 6.14: Browser autofill behavior before and after the release of our research results. Prior to releasing our results all of the major browsers would autofill usernames as soon as the user visited the page. After we released our results [28], Chrome changed their autofill policy to require user interaction with the page for both username (u) and password (p) [90] and Safari updated their autofill to require explicit user interaction with the form [44].

At the time of measurement, login form autofilling didn't require user interaction; all of the major browsers would autofill the username (often an email address) immediately, regardless of the visibility of the form. Chrome didn't autofill the password field until the user clicks or touches anywhere on the page. The other browsers we tested didn't require user interaction to autofill password fields. Thus, third-party javascript was able to retrieve the user's saved credentials by creating a form with fields for username and password, which would have been autofilled by the login manager. After releasing our results, some browser vendors changed their default behavior—we summarize browser autofill behavior before and after our discoveries in Table 6.14.

The underlying vulnerability of login managers to credential theft has been known for years. Much of the past discussion has focused on password exfiltration by malicious scripts through cross-site scripting (XSS) attacks [70, 72]. Fortunately, we didn't find password theft on the 50,000 sites that we analyzed. Instead, we found third-party scripts embedded by the first party abusing the login manager to extract email addresses.

Figure 6.3 details the attack. First, a user fills out a login form on the page and asks the browser to save the login. The third-party script does not need to be present on the login page. Then, the user visits another page on the same website which

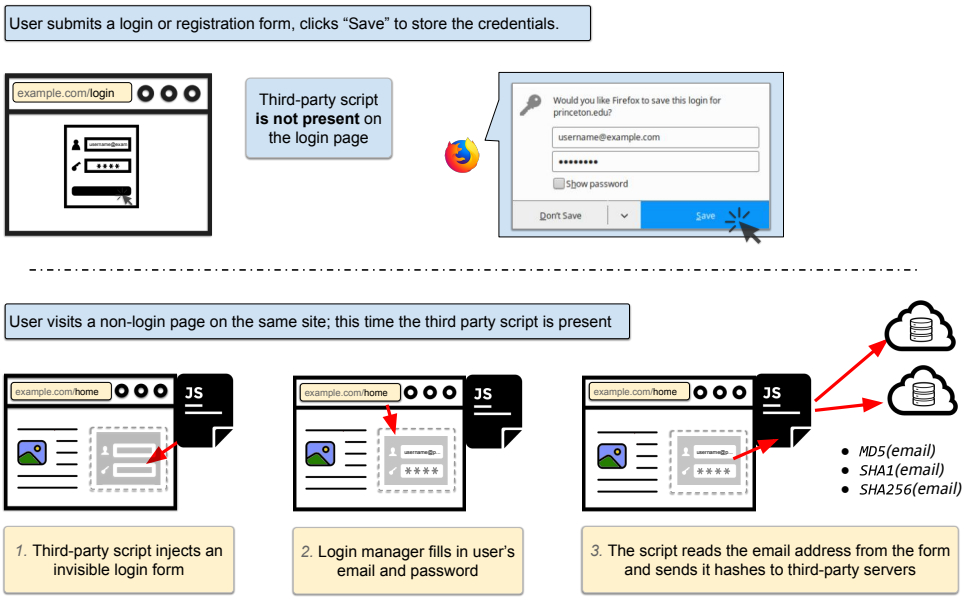


Figure 6.3: The process by which a script can extract a user's credentials from the browser login manager. First, the user logs into a website and insrtucts the browser to save their credenitals for that site; the third party does not need to be present on the login page. Next, the user visits a portion of the site which embeds the third-party script in the main page context but does not have a login box. The third-party script injects an invisible login box into the page, waits for the browser to autofill it, and then extracts the username.

includes the third-party script. The script inserts an invisible login form, which is automatically filled in by the browser's login manager. The third-party script retrieves the user's email address by reading the populated form and sends the email hashes to third-party servers.

Measurement methods. To study password manager abuse, we extended OpenWPM to simulate a user with saved login credentials and added instrumentation to monitor form access. We used Firefox's `nsILoginManager` interface to add credentials as if they were previously stored by the user. We did not otherwise alter the functionality of the password manager or attempt to manually fill login forms.

The fake credentials acted as bait (Section 6.2.2). To detect when the credentials were accessed, we added the following probes to OpenWPM's Javascript monitoring instrumentation (Section 3.1.2):

1. Monitor the DOM mutation events `DOMNodeInserted` and `DOMAttrModified`. These are used to detect the injection of a new login form into the DOM. When either of these events fire, we serialize the inserted or modified HTML elements for events that contain a password element.

2. Instrument `HTMLInputElement` and `HTMLFormElement` to intercept access to form input fields. We log the input field value that is being read to detect when the bait email (autofilled by the built-in password manager) was sniffed.

3. Store HTTP request and response data, including `POST` bodies to detect when the username or password is sent to a remote server using the leak detection method detailed in Section 3.2.5.

For both the JavaScript (1, 2) and the HTTP (3) instrumentation we store JavaScript stack traces at the time of the function call or the HTTP request. We parse the stack trace to pin down the initiators of the HTTP request or the parties responsible for inserting and accessing a form. Specifically, we select scripts that do the following:

- Inject an HTML element containing a password field

- Read the email address from the input field automatically filled by the browser's login manager

- Send the email address, or a hash of it, over HTTP

Applying this selection criteria to our measurement data collected during June 2017 produces a list of scripts that may use this technique to collect email addresses.

Company	Script address	# of sites
Adthink	`https://static.audienceinsights.net/t.js`	1047
OnAudience	`http://api.behavioralengine.com/scripts/be-init.js`	63

Table 6.15: We found two scripts which misused the browser login manager to extract user email addresses. These scripts were found on 1,110 of the Alexa top 1 million sites in the September 2017 Princeton Web Census crawl (Section 4.1). After we released our results in December 2017, both companies removed the functionality from their scripts.

We verified that the forms inserted by these scripts were not visible to the user. We also reproduced the email address collection by manually registering for several websites that embed the scripts and allowing the browser to save the credentials in the process.

Results. We found two scripts, loaded from AdThink and OnAudience, which used this technique to extract email addresses from login managers on the websites which embed them. We summarize our findings in Table 6.15. After our findings were released publicly in December 2017, the two companies stopped the practice.

We provide code snippets from both scripts in Appendix A.8. Adthink's script sent the MD5, SHA1 and SHA256 hashes of the email address to its server (`secure.audienceinsights.net`), as well as the MD5 hash of the email address to the data broker Acxiom (`p-eu.acxiom-online.com`). OnAudience's script sent the MD5 hash of the email back to its own server. In addition, their script also collected browser features including plugins, MIME types, screen dimensions, language, timezone information, user agent string, OS and CPU information. The script then generated a hash based on this browser fingerprint. OnAudience's script was most commonly present on Polish websites, including newspapers, ISPs and online retailers. 45 of the 63 sites that embedded the script at the time of measurement had the ".pl" country code top-level domain.

6.2.4 Third-party collection of PII through DOM scraping

Mechanism. The top-level context's DOM can contain sensitive information. When a user logs in to a site, the site may display her name or email address in the text of the page. Similarly, users may enter personal information, such as their address, credit card number, or social security number into forms on the page. For some sites this information may be even more sensitive, such as the user's bank account balance or medical history. Third-party scripts embedded in the top-level context have access to the same information that's displayed to the user when she visits the site. Malicious scripts can abuse this access to surreptitiously collect user information. However, this sensitive information can also get scooped up by benign scripts which collect portions of the DOM as part of the services they provide first-parties.

In this section we examine third-party scripts which collect the full contents of the DOM, as well as those that monitor all mouse movements and key presses on the page. There are a number of ways a script can grab the contents of the DOM, including: calling `document.body.outerHTML` or `document.body.innerHTML`, looping through all elements in the DOM and serializing them individually, or by calling `document.body.textContent` to retrieve all text on the page. To monitor mouse movements or key presses the scripts can add event listeners for these DOM events on nodes at the top of the DOM. Finally, scripts can listen to the `blur` or `change` events that are fired when a user interacts with input elements on the page.

Detection Method. We take a two-step approach to detecting whole DOM exfiltration. First, we append several unique *bait* values to the DOM of all frames present on a page and search for these values in the resulting network traffic. The values are added by creating a new `div` element and adding it directly to `document.body`. The `div` has the style `display:none` set to prevent it from altering the layout of the page. The added values include an email address and the string `Welcome <FirstName> <LastName>!`, where `<FirstName>` and `<LastName>` are unique strings. We chose to

include bait values that match the format of a real users PII to detect if any scripts are parsing information out of the DOM. We use the leak detection method detailed in Section 3.2.5 to discover leaks of the injected email address or name.

We discovered several instances where scripts compress and split their payload over multiple requests, which our standard leak detection method fails to detect. To capture these instances, we take a different approach:

1. **Determine which sites contain scripts that might be exfiltrating the DOM.** For each first-party site, we sum the total size of data sent to each third party in POST requests on that site. We then flag any third party that receives a total amount of data which exceeds the compressed size of the page source of the first-party site. Finally, we generate a list of sites that include at least one of these third parties.

2. **Re-measure the candidate sites with and without a large chunk of data appended to the DOM.** For all sites on which we suspect third-party DOM exfiltration, we re-measure the site twice: once with a 200 Kilobyte chunk appended directly to the `body` element of the DOM, and once without any data appended to the DOM. We chose 200 Kilobytes as the chunk size because it was large enough to outweigh small differences in page size between the crawls (such as a changing headline), but not so large as to disrupt services that collected data from the DOM.

3. **Measure the difference in payload size for each third party between the crawls.** We sum the total payload size across all POST requests which occur during a single page visit for each third party. We then compare the total POST request size between the two crawls, and flag any third-party script that had a difference greater than the compressed length of the injected chunk of data (approximately 150 Kilobytes).

The two detection methods described above provide a list of scripts which appear to collect data from the DOM. To better understand how the scripts are monitoring interaction with the DOM we use OpenWPM's Javascript call monitoring (Section 3.1.2) instrumentation to record the following: calls to `innerHTML`, `outerHTML`, `innerText`, and `outerText` on the `HTMLBodyElement` or the `documentElement`. In addition we observe all event listener registrations on the `HTMLBodyElement`, the `window.document` object, and the `window` object. We examine registrations which monitor events that capture the user's mouse movements, page interactions, and key presses.[10] We take a script's use of these events as a signal that they are monitoring user interaction with the DOM in addition to scraping the contents of the DOM.

Finally, we use a combination of the presence of DOM scraping, the registration of event handlers monitoring user interaction, and a manual examination of the marketing materials of the companies involved to determine the type of services the third-party script offers. These results of this classification are summarized below.

Results. We found no instances of malicious scripts parsing the DOM to exfiltrate user data. However, we did find a number of companies doing full-page scraping, collecting all of the text on the page or serializing portions of the DOM. Although these services do not appear to be built with the intention of collecting PII, the broad nature of the collection techniques makes it very easy for PII to get scooped up with the rest of the collected data. We summarize our findings in Table 6.16. The majority of the scripts we discovered sent requests which included the name and email address inserted by our instrumentation. Some of the scripts were only discovered by our chunk injection measurement due to unsupported payload encodings. In all cases, the data was transferred to a third-party collection endpoint via a `POST` request.

[10]The event listener registration events that we monitor include: `mouseup`, `mousedown`, `click`, `auxclick`, `mousemove`, `wheel`, `dblclick`, `select`, `contentmenu`, `mouseleave`, `mouseout`, `mouseenter`, `mouseover`, `keydown`, `keyup`, `keypress`, and `scroll`.

Service	Purpose	# sites
Yandex Metrika	Analytics	198
FullStory	Analytics	55
Hotjar	Analytics	48
SkimLinks	Advertising	34
Sessioncam	Analytics	18
UserReplay	Analytics	15
Transifex	Translation	9
VWO	Analytics	7
Tealeaf	Analytics	7
Jornaya	Analytics	5
IntelliTXT	Advertising	4
Digidip	Advertising	4
RedLink	Analytics	3
Localizer	Translation	2
Viglink	Advertising	2
Prosperent	Advertising	1
Wovn	Translation	1
xclaimwords	Unknown	1
Bkred.ru	Unknown	1
ABTasty	Analytics	1

(a) Services detected by both ID injection and chunk injection.

Service	Purpose	# sites
Clicktale	Analytics	37
Smartlook	Analytics	31
Lucky Orange	Analytics	23
Quantum Metric	Analytics	11
Inspectlet	Analytics	10
Mouseflow	Analytics	5
LogRocket	Analytics	2
SaleMove	Support	1

(b) Services detected only by chunk injection. Since our chunk injection measurement was run on a sample of the 50,000 sites measured in (a), we expect the site counts for these services to be underrepresented relative to (a).

Table 6.16: The top companies that we discovered scraping information from the DOM at the time of our June or November 2017 measurements. The apparent purpose of data collection includes: **Analytics:** heatmaps, session recording, form analytics, **Advertising:** mouse-over keyword ads, automatic affiliate link insertion, **Translation:** automatic localization, and **Support:** live customer support. Scripts grab either the text on the page or a representation of the DOM, which can range from a complete serialization to a custom encoding of some elements.

The majority of the companies collect DOM data to provide analytics services to the first party. Of these, the most commonly provided service is "session replay", which was offered by 16 companies at the time of measurement. Session replay services allow the first party to observe how their page was rendered for the user and how the user interacted with their site. All of the session replay providers collected some custom encoding of all nodes and text in the DOM, in some cases including all of the inline script and CSS content. We believe this is necessary to allow the recording to accurately reflect the experience of the user, given that many pages are built dynamically and may change from user to user.

In the remaining cases, text was extracted from the DOM and sent to a third party. While we aren't able to measure exactly *how* the text data is used, we studied the marketing material of the companies to better understand how it might be used. Several of the companies provided automatic monetization of product or retailer references by attaching affiliate links to specific keywords. For example, a reference to a new model of Nike shoes would be replaced by an affiliate link to a store where the user can purchase that shoe. IntelliTXT's advertising product is slight variation of this; rather than replacing the text with an affiliate link, it replaces the text with a mouse-over advertisement that displays in a tooltip next to the text. Finally, several of the companies offer translation services, which include automatic localization of site content.

We found no evidence that suggests the scraped data was used for advertisement personalization or cross-site tracking by any of the companies analyzed. In fact, several of the analytics companies explicitly forbid the collection of sensitive user information using their services [134, 292], and provide automated and manual features that first parties can use to prevent the collection of sensitive user data. As an example, we observed code in VigLink and Wovn scripts which filter the contents of the collected data using regular expressions. VigLink's filtering appears to be motivated by a desire to protect user privacy, as evidenced by the inclusion of references to `pii`. Their script prevented the collection of email addresses and strings of integers between 6 and 18 characters in length. Wovn's filtering appeared to be intended to prevent the collection of non-translatable strings and included email addresses.

Case study: the ineffectiveness of session replay redaction tools. To better understand the effectiveness of the data privacy features offered by third-party services, we performed an in-depth examination of the redaction tools provided by six of the companies offering session recording services: FullStory, UserReplay, SessionCam, Hotjar, Yandex, and Smartlook. Session replay analytics are meant

Redacted Field	FullStory	UserReplay	SessionCam	Hotjar	Yandex	Smartlook
Name	○	◐	◐	○	○	○
Email	○	◐	◐	○	○	○
Phone	○	◐	◐	○	○	○
Address	○	◐	◐	○†	○	○
SSN	○	◐	◐	○	○	○
DOB	○	◐	◐	○	○	○
Password	●	◐	●	●	●	◐
CC Number	●	◐*	◐	◐	○	●
CC CVC	●	◐	◐	○	○	●
CC Expiry	●	◐	◐	○	○	●

Table 6.17: Summary of the automated redaction features for form inputs enabled by default from each company at the time of measurement in November 2017.
Filled circle: Data is excluded; **Half-filled circle:** equivalent length masking; **Empty circle:** Data is sent in the clear
* UserReplay sent the last 4 digits of the credit card field in plain text
† Hotjar masked the street address portion of the address field

to provide insights into how users interact with websites and discovering broken or confusing pages. However the extent of data collected by these services is likely to exceed user expectations; text typed into forms is collected before the user submits the form, and precise mouse movements are saved, all without any visual indication to the user.

The replay services offer a combination of manual and automatic redaction tools that allow publishers to exclude sensitive information from recordings. However, in order for leaks to be avoided, publishers would need to diligently check and scrub all pages which display or accept user information. For dynamically generated sites, this process would involve inspecting the underlying web application's server-side code. Further, this process would need to be repeated every time a site is updated or the web application that powers the site is changed.

To better understand the effectiveness of these redaction practices, we set up test pages during November 2017 and installed replay scripts from the six companies. All of the companies studied offered some mitigation through automated redaction, but the coverage offered varied greatly by provider. UserReplay and SessionCam replaced all user input with an equivalent length masking text, while FullStory, Hotjar, and

Smartlook excluded specific input fields by type. We summarize the redaction of other fields in Table 6.17.

Automated redaction is imperfect; fields are redacted by input element type or heuristics, which may not always match the implementation used by publishers. For example, FullStory redacts credit card fields with the `autocomplete` attribute set to `cc-number`, but will collect any credit card numbers included in forms without this attribute. Indeed, we discovered credit card data leaking to FullStory from input fields that lacked `autocomplete` attributes (see Table 6.18).

To supplement automated redaction, several of the session recording companies, including Smartlook, Yandex, FullStory, SessionCam, and Hotjar allowed sites to further specify input elements to be excluded from the recording. To effectively deploy these mitigations a publisher would need to actively audit every input element to determine if it contains personal data. A safer approach would have been to mask or redact all inputs by default, as was done by UserReplay and SessionCam, and allow whitelisting of known-safe values. Even fully masked inputs provide imperfect protection. For example, the masking used by UserReplay and Smartlook at the time of measurement leaked the length of the user's password.

Several of the session recording companies also offered redaction options for display content, i.e. content that would be collected through scraping the DOM. Unlike user input recording, none of the companies appeared to provide automated redaction of displayed content by default; all displayed content on our test page ended up leaking. Instead, session recording companies expect sites to manually label all personally identifying information included in the DOM. Sensitive user data has a number of avenues to end up in recordings, and small leaks over several pages can lead to a large accumulation of personal data in a single session recording.

To understand how well the manual redaction features work in practice, we manually examined around 50 of the top sites on which we found session replay scripts.

We discovered several categories of sensitive information leaks during our interactions, including: passwords, medical information, student data, credit card data, and purchase information. Table 6.18 summarizes our findings.

We observed password leaks on three of the surveyed websites. On two of the sites, `propellerads.com` and `johnlewis.com`, the password leak was caused by the way the sites implemented a "show password" feature. In both instances, the sites stored the user's password in two input elements: one of type `password` and one of type `text`. When the user interacts with the "show password" feature, the sites would swap the two input elements, causing the user's password to become visible. Both FullStory's and SessionCam's automated redaction rules failed to capture the input element of type `text`. In the third case, the password leak was caused by a bug in the way FullStory's manual redaction feature applied to input fields of type `password`. In all cases we were informed by the third party services that the issues were later fixed.

With the exception of `walgreens.com`, the remainder of the leaks largely appeared to be caused by a sparse use of redaction on those pages. Walgreens made extensive use of display content redaction, but the user's name and prescription choices appeared on subsequent pages of the checkout process. Similarly, the identity verification page asked several multiple choice questions containing sensitive user data—the radio buttons for the questions were redacted from recordings, but the mouse traces would still reveal the user's answers.

6.2.5 Countermeasures to PII collection

Publishers, users, and browser vendors can all take steps to prevent data exfiltration. To protect against autofill exfiltration, publishers can isolate login forms by putting them on a separate subdomain, which prevents autofill from working on non-login pages. This does have drawbacks including an increase in engineering complexity.

First party	Third party	Data Leaked	Cause
walgreens.com	FullStory	prescriptions, name, identity	unredacted display data
propellerads.com	FullStory	passwords	"show password" feature
johnlewis.com	SessionCam	passwords	"show password" feature
wpengine.com	FullStory	passwords	bug in FullStory redaction
gradescope.com	FullStory	student data	unredacted display data
lenovo.com	FullStory	billing information	unredacted display data
bonobos.com	FullStory	credit card data	unredacted input data

Table 6.18: A sample of sensitive data leaks to session replay companies that we observed during a manual inspection of sites between November 2017 and February 2018. The sites may have changed their practices since our measurements.

Similarly, publisher could only use third-party services that scrape DOM data and record user behavior on pages that don't contain sensitive user data.

Users can install ad blockers or tracking protection extensions to prevent tracking by invasive third-party scripts. The domains used to serve the two autofill extraction scripts (behavioralengine.com and audienceinsights.net) were blocked by the EasyPrivacy blocklist at the time of measurement. Many of the domains used to serve the session replay scripts were already blocked by EasyPrivacy at the time of our measurements. Several of those that weren't, including domains from FullStory, Smartlook, and UserReplay were added after we released our results publicly [116].

Turning to browser vendors, the simplest defense against autofill exfiltration is to allow users to disable autofill. For instance, the Firefox preference `signon.autofillForms` can be set to false to disable autofilling of credentials. A less crude defense is to require user interaction before autofilling login forms. This approach was taken by Safari shortly after we released our results [44]. We summarize the browser vendor response to our research in Table 6.14.

The upcoming W3C Credential Management API [335] requires browsers to display a notification when user credentials are provided to a page. Browsers may display the same notification when login information is autofilled by the built-in login managers. Likewise, browsers could detect the presence of scripts which scrape the DOM or provide session replay services and give users a visual indication of their presence or

activation. Displays of this type won't directly prevent abuse, but they make attacks more visible to publishers and privacy-conscious users.

6.3 The ineffectiveness of hashing for privacy

We observed trackers collecting hashed email addresses in the content of emails (Section 6.1.3) as well as around the web (Section 6.2). The putative justification for email address leaks in the online ad tech industry is that the address is hashed. For example, Criteo's privacy policy states: "A hash of your email [...] does not permit your identification" [96]. Similarly, LiveIntent—the largest recipient of leaked email addresses in our study (Section 6.1.3)—states "To de-identify this information, either we or our business partners [hash it]" [205].

However, hashing of PII, including emails, is not a meaningful privacy protection. Compared to hashing of passwords, there are several reasons why hashing of email addresses is far more easily reversible via variants of a dictionary attack. First, while (at least) some users attempt to maximize the entropy of passwords, most users aim to pick memorable emails, and hence the set of potential emails is effectively enumerable. Due to the availability of cloud GPUs, trillions of hashes can be attempted with low cost [212]. In fact, past research studies achieved email hash recovery rates between 42% [211] and 70% [64] using simple heuristics to generate email addresses. Second, unlike password hashing, salting is not applicable to email hashing since multiple third parties need to be able to independently derive the same hash from the email address.

Perhaps most importantly, if the adversary's goal is to retrieve records corresponding to a known email address or set of email addresses, then hashing is pointless—the adversary can simply hash the email addresses and then look them up. For example, if the adversary is a surveillance agency, as we examine in Chapter 7, and seeks to

retrieve network logs corresponding to a given email address, this is trivially possible despite hashing.

6.4 Summary

In this chapter we showed that leaks of personal data are common on the web and in email content. Email security and privacy has not received much research attention despite its central importance in digital life. This is of concern not only because trackers can learn the recipient's IP address, when emails were opened, and so on, but also because these third parties are by and large the same ones that are involved in web tracking. We also observe web trackers collecting much of the same user data, albeit using different approaches. This means that trackers can connect email addresses to browsing histories and profiles, which leads to further privacy breaches such as cross-device tracking and linking of online and offline activities. Indeed, email-hash-based tracking is straightforward cross-device tracking, since users tend to both view emails and share their email addresses with websites across all of their devices.

Even network adversaries can benefit from the PII leaks in emails and across the web. The NSA is known to piggyback on advertising cookies for surveillance (Section 2.2), and our work suggests one way in which a surveillance agency might attach identities to web activity records, in line with our findings in Chapter 7. Indeed, nearly 91% of URLs which leaked email addresses during our email viewing simulation were sent without encryption.

Chapter 7

The surveillance implications of web tracking

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

More specifically, the adversary has the ability to inspect packet contents and wishes to either track an individual target user or surveil users *en masse*. A key challenge for the adversary is the lack of persistent identifiers visible on the network. However, the adversary can observe HTTP cookies in transit. Indeed, both the NSA and GCHQ are known to use such cookies for surveillance (Section 2.2).

In past chapters we established that third-party cookies are ubiquitous on the web (Section 4.1). This chapter starts with three insights. First, the presence of *multiple unrelated* third-party cookies on most web pages, albeit pseudonymous, can tie together most of a user's web traffic without having to rely on IP addresses (Figure 7.1). Thus the adversary can separate network traffic into clusters, with each cluster corresponding to only one user (or more precisely, one browser instance). A single user's traffic may span more than one cluster if the linking is imperfect.

Second, a significant portion of a typical user's traffic traverses U.S. borders even when the user is outside the U.S. and browses local content. As it turns out, such sites frequently include third-party resources such as analytics scripts, tracking pixels, and advertisements from U.S. servers. This leaves foreign users particularly vulnerable to the NSA's wiretaps within the U.S. under the "one-end foreign" rule (Section 2.2).

Third, although most popular websites now deploy HTTPS for authentication, many web pages reveal an already logged-in user's *identity* in plaintext. Furthermore, we find that third-party trackers impede HTTPS adoption on sites (Section 7.4), making it more difficult for sites to mitigate this attack. Thus, an adversary that can wiretap the network can not only cluster together the web pages visited by a user, but can then attach real-world identities to those clusters. This technique relies on nothing other than the network traffic itself for identifying targets.

Figure 7.1 illustrates the basis for this chapter. The adversary observes the user visit three different web pages which embed trackers X, Y or both. The user's IP address may change between visits to each page, though we assume it is consistent for the request to site A and the request to A's embedded tracker X. But there is no way to tie together her visits to pages A and B until she visits C after which all three visits can be connected. The unique cookie from X connects A and C while the one from Y connects B and C. We assume here that the user has visited pages with both trackers before so that cookies have already been set in her browser and will be

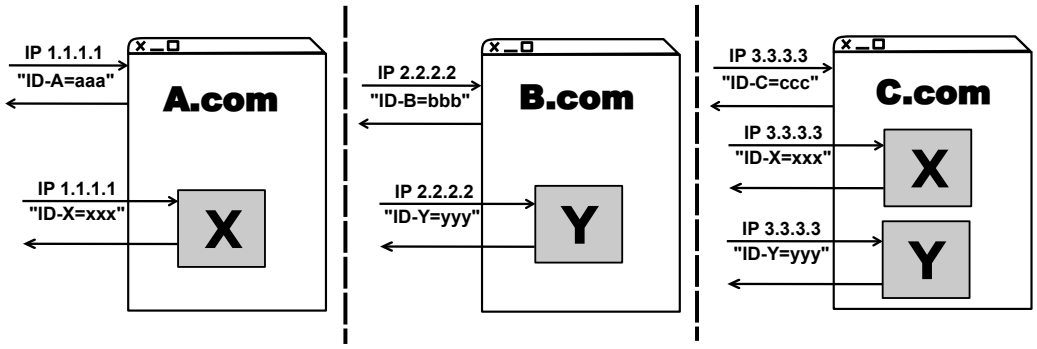


Figure 7.1: Illustration of link between each of a single browser's visits to three first-party pages using two different third-party tracking cookies. The user accesses the web at three different times, behind three different IP addresses.

sent with each request. While IP address is a *convenient* method to link a request to a first party page to the corresponding request to an embedded third party tracker, it is not necessary. In Section 7.5.1 we show how this linkage can be achieved even if the IP address cannot be observed at all or if an IP address is shared by many users.

7.1 Threat model

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

We consider only passive attacks for several reasons. First, passive attacks appear to be more powerful than generally realized, and we wish to highlight this fact.

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

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

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

The adversary's task is complicated by the fact that the IP addresses of the target(s) may change frequently. A user's IP address could change because she is physically mobile, her ISP assigns IP addresses dynamically, or she is using Tor. Leaked GCHQ documents show that their search interface even warns analysts to take care when selecting data on dynamic IPs [135]. Browsing from a smartphone is a case worth highlighting: Balakrishnan et al. find that "individual cell phones can expose different IP addresses to servers within time spans of a few minutes" and

that "cell phone IP addresses do not embed geographical information at reasonable fidelity" [48].

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

7.2 Measurement methods

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

1. Define all clients and adversaries to be studied, according to the following:

    - client: (location, browsing model, browser configuration) which encodes the client's geographic and network location, which sites the client visits, and the browser settings and plugins the client browses with.
    - adversary: (location, policy restrictions) which encodes the adversary's geographic and network location, and the policy restrictions on data use and collection.

2. For each unique (user location, browsing model) pair of interest, generate N simulated browsing profiles as defined in Section 7.2.1 and create a corresponding client instance for each one.

3. Use the measurement infrastructure (Section 3.1) to simulate the users defined in Step 2 and collect all network traffic (i.e. HTTP requests, responses, and cookies). Our measurements are summarized in Section 7.2.2.

4. For each (client location, web resource) pair of interest, determine the geographic path of traffic using the procedure described in Section 7.2.3.

5. Run the ID cookie detection algorithm detailed in Section 3.2.4 to flag identifying cookies

6. For each (client, adversary) pair of interest, do the following for all instances of the client and average the results:

    - filter the list of requests based on the geographic location and policy restrictions of the adversary using the geographic mapping in Step 4.
    - run the cookie linking algorithm detailed in Section 7.2.4 using the ID cookies detected in Step 5.
    - report the size of the connected components in the linking graph (as a ratio of total number of visits)
    - report the number of sites known to leak real-world identifiers (Section 7.2.5) contained in each component.

7.2.1 Browsing models

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

Our other browsing model aims for realism by making use of the AOL search query log dataset. The dataset contains queries made by 650,000 anonymous users over a three month period (March–May 2006). We create a browsing profile from a user's search queries as follows. First, we remove repeated queries. Next, for every search query performed by the user, we submit the query to Google search and retrieve the links for the first five results. Users were selected on the basis that they performed between 50 to 100 unique queries which resulted in browsing profiles of 250 to 500 URLs. This is almost identical to the method used by Liu et al. [203].

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

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

7.2.2 Measurement configuration

We deployed OpenWPM version 0.1.0 on Amazon EC2[1] instances in three regions: Northern Virginia, United States, Dublin, Ireland, and Tokyo, Japan. We chose these to achieve as much geographic diversity as possible from Amazon's limited set of regions. Each measurement took place on an m3.medium instance of Ubuntu 14.04

[1] `https://aws.amazon.com/ec2/`

in June 2014. All measurements were ran using 25 simulated profiles for each (user location, browsing model, browser configuration) combination.

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

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

To run the identifying cookie detection algorithm described in Section 3.2.4, we also require synchronized measurements of each site visit from two separate machines. We ran these measurements from the Northern Virginia location and visited all of the links which may be visited by any other measurement instance (13,644 links total).

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

7.2.3 HTTP Traffic geolocation

In order to determine the if an HTTP request is bound for a specific location of interest, we augment commercially available geolocation data with additional measurement data. After each measurement instance finished browsing, we ran a `traceroute`[2] to

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

each unique hostname and recorded the full output. All IPs returned in each hop of the traceroute were geo-located with the `MaxMind GeoLite2`[3] country databases.

The mapping between IP and physical location is not one-to-one. Instead, there may be many servers in different locations which all share the same IP address for various purposes. One such example is *anycasting*, the process by which several nodes share the same address and the user is routed to the nearest node.[4]

Thus, when determining if an HTTP request enters a specific country it is not sufficient to simply geolocate the IPs returned from a traceroute to that host. As a solution, we implement a simplified version of the *geo-inconsistency* check proposed by Madory, et.al. [208]. We check that the minimum round-trip time (RTT) returned by a traceroute to each hop is greater than the physical minimum RTT assuming a direct fiber-optic connection between the two locations.

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

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

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

[3] `http://dev.maxmind.com/geoip/geoip2/geolite2/`

[4] CloudFlare, for example, claims to use anycasting as part of their content delivery network: `https://www.cloudflare.com/features-cdn`

```
Data: httpRequest, testCountry
Result: True/False if httpRequest enters testCountry
origin ← latitude/longitude of origin server
hostname ← parse httpRequest.url
for each hop in hostname traceroute do
    location ← geolocate(hop.IP)
    if location not in testCountry then
        continue
    if location not city code then
        location ← closest point to origin within country
    dist ← haversine(origin,location)
    minRTT ← 2 * minimum time to travel dist
    if hop.RTT > minRTT then
        return True
end
return False
```

Algorithm 1: Origin-specific geolocation check

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

We use a simplified version of this check when examining if requests are exiting the United States. Since a request can be bound for any non-U.S. destination, we do not attempt to find the closest point in each country. Instead, we only check that the observed RTT is greater than the minimum RTT to the geolocation point regardless of the point's location.

7.2.4 Transitive Cookie Linking

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

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

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

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

```
Data: httpRequests and httpResponses for user_i
Result: Graph G_i for user_i with URL & Cookie nodes
for each visited url do
    for httpRequest do
        if HTTPS then
            continue
        urlNode ← createNode (U, req.url, req.inUS)
        G.addNode(urlNode)
        if req.referer is not empty then
            refNode ← createNode (U, ref.url)
            G.addNode(refNode)
            G.addEdge(refNode, urlNode, req.inUS)
        if req.cookie is not empty and is identifying then
            cookieNode ← createNode (C, cookie.value)
            G.addNode(cookieNode)
            G.addEdge(cookieNode, urlNode, req.inUS)
    end
    for httpResponse with Set-Cookie do
        if HTTPS then
            continue
        if cookie is identifying then
            cookieNode ← createNode (C, cookie.value)
            urlNode ← node for requested url
            G.addNode(cookieNode)
            G.addEdge(cookieNode, urlNode, req.inUS)
    end
    for httpResponse with location field do
        if HTTPS then
            continue
        urlNode ← node for requested url
        locNode ← createNode (U, loc.url)
        G.addNode(locNode)
        G.addEdge(locNode, urlNode, req.inUS)
    end
end
```

Algorithm 2: Cookie Linking algorithm

could be processed to link them individually to real world identities, or disambiguated by identifying disjoint sets of cookies for the same sites.

When evaluating adversaries restricted by policy or geographic constraints, an additional pre-processing step is required before finding the GCC. Utilizing the ge-

olocation data from Section 7.2.3, we are able to filter nodes from the cookie linking graph based on geographic restrictions. In order to determine the amount of traffic for a specific user that a U.S. restricted adversary has access to, we filter all edges from the cookie linking graph that were not added due to U.S. bound requests. We create a subgraph from this filtered edge list and continue the normal linking analysis.

7.2.5 Identity leakage in popular websites

We conducted a manual analysis in June 2014 of identity leaks on the most popular pages which allow account creation. The top-50 pages are a useful representation for how heavily-used sites with user accounts manage data, and are more likely to be visited by a real user. We identified 50 of the Alexa top 68 U.S. sites that allow for account creation and signed up test accounts when possible. We then examined the homepage, account page, and several random pages on each site to see if any of identifiers are displayed on an HTTP page. If so, an adversary collecting HTTP traffic for that user could inspect the contents of the page to find the identifier and link it to any tracking identifiers also loaded on the page. These leaks should be considered a lower bound to the data that would be available to an adversary in practice; our review of pages did not consider identifiers that were not displayed to the user (but would be available in the page source), we did not visit all pages on the sites, and we did not include PII-based tracking identifiers (Chapter 6).

7.3 Network adversaries can effectively cluster traffic

In the course of our measurements we make nearly 100,000 page visits to 13,644 distinct sites under several client and adversary models. Of these 13,644 sites, nearly

all of them (13,266) make requests for external content from a host different than the domain of the host visited.

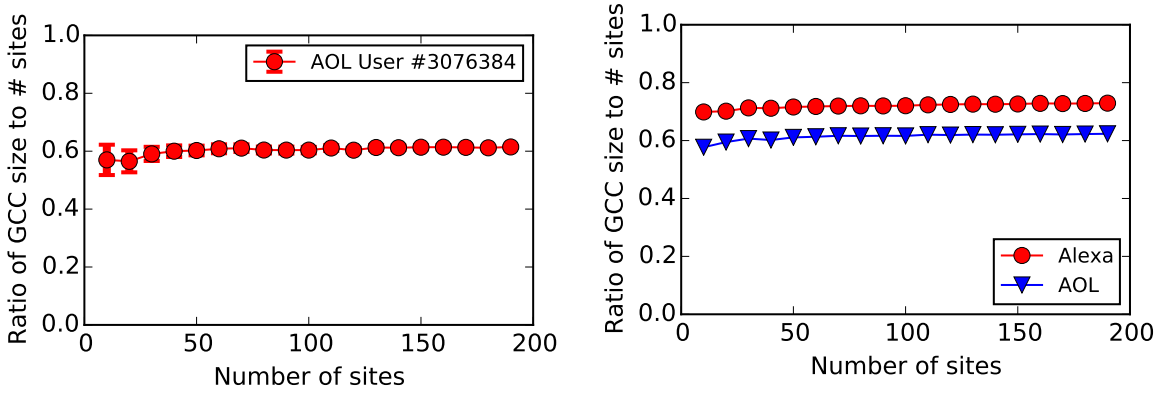
7.3.1 Clustering

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

An adversary's incomplete view. We must consider that the adversary's position on the network may not give them a full view of any user's traffic. The adversary's view may be limited due to its policy or geographic restrictions as described in Section 2.2, however even with these considerations an adversary may not see all of a user's traffic. A user may change locations on the network or alternative routes taken by packets through the network might result in gaps in the adversary's data collection. To model the adversary's partial view of the user's browsing activity, we repeat our analysis with random subsets of web pages of various sizes.

For illustration, Figure 7.2a shows how the GCC of a single AOL user's page visits (y-axis) grows as we vary the completeness of the adversary's view of the user's HTTP traffic (x-axis). Each data point was computed by taking 50 independently random samples of page visits. For each sample we apply the clustering algorithm and compute the fraction contained in the GCC. We then average the fractions across the 50 samples. Since we wish to simulate these page visits being spread out over

time, only cookies with expiration times at least three months into the future were included when computing the clusters.



(a) Individual AOL User

(b) Average AOL & Alexa Users

Figure 7.2: Clustering, random subsets of traffic

Thus for each (x, y) pair we can say that if the adversary captures x web page visits by a user in the course of a wiretap, they could link approximately $y\%$ of those visits into a single cluster. The numbers we see for this user are typical — the fraction is around 55% for even very small clusters and exceeds 60% as the cluster size increases. As discussed in Section 7.2, we average all results over $N = 25$ client instances for all (client, adversary combinations).

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

Unrestricted Adversary — AOL User Profiles We first examine the level of clustering an adversary can achieve when it is not subject to any policy or geographic restriction. The results for users simulated under the AOL browsing model and no blocking tools are included in Figure 7.2b. These results show that the clustering remains very similar to the single user case of Figure 7.2a, except that no inconsistencies remain. After just 55 web page visits observed by the adversary, the growth

of the GCC flattens out to $62.4 \pm 3.2\%$ after 200 sites. For the remaining results, we will only present values for this asymptotic case of 200 sites, as the shape of the GCC growth is nearly identical in all cases.

Unrestricted Adversary — Alexa profiles Next we examine the effect of the browser model on the ability of an unrestricted adversary to cluster user data. We hold the user location and browser configuration set to browsing within the U.S. with no blocking settings. Figure 7.2b compares the results for Alexa profiles for U.S. users against the AOL profiles. The Alexa clustering shows a similar growth pattern with an offset around 10% higher on average, with an overall level of clustering after two sites of $72.9 \pm 1\%$.

We attribute this higher rate of clustering to several factors. First, the creation of our AOL browsing model biases visited pages towards those that rank high in search results. In particular, `wikipedia.org` is consistently in the top 5 results for many queries, and accounts for about 5% of the sites not in the GCC on average. Second, search results will return a much more diverse set of sites than the Alexa model samples from, which may contain sites that include less third-party trackers (e.g. government sites). In our own data, we see that the top 500 US Alexa sites include resources from an average of 62 unique hosts, whereas the average for our AOL profiles is 23.

7.3.2 U.S. Users Under One-End Foreign

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

To understand why this occurs, we must look at the composition of HTTP traffic. For the average user in this (adversary, client) pair, at least one non-U.S. sub-resource request occurs for 31.7% of the Alexa top-500 sites in the U.S. However, the overall number of HTTP Requests leaving the United States is comparatively small, accounting for just 2.0% of all requests. Only considering traffic where an adversary could learn the top-level domain through the referrer headers, this reduces to 22.7% of visits and 1.6% of requests.[5] Although nearly a quarter of a user's browsing history is visible to the adversary, we show that cookie linking is an ineffective method to cluster this traffic.

7.3.3 Cookie Linking in Non-U.S. Traffic

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

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

Table 7.1: Clustering of non-U.S. users by an adversary with no restrictions vs. one restricted to U.S. bound traffic.

Unrestricted Adversary — non-U.S. profiles When all requests are considered, the level of clustering is similar to that of the U.S.-based Alexa user simulations. Table 7.1 shows amount of clustering that occurs for users in Ireland and Japan under

[5]These results are broadly consistent with measurements taken in past work [218].

the random subset clustering model. Simulated users in Ireland can expect around 63% of traffic to be clustered, while users in Japan can expect nearly 60%. We believe the differences between user simulations generated using Alexa's top U.S., Japan, and Ireland lists arises from the difference in the number of included third parties on the first party (i.e., 62, 38, and 30 on average, respectively).

U.S. based Adversary — non-U.S. profiles We then restrict the clustering to only consider requests which are U.S. bound, and cluster based on the information available to a geographically restricted adversary. This could include an adversary within the United States, or an adversary sitting at the entrance to undersea cables returning to the United States. Table 7.1 shows clustering capabilities of an adversary restricted to these conditions. In Ireland, we see a reduction to around 13% of page visits and in Japan we see a less severe decrease to 20% of visited sites.

7.3.4 Cookie Linking Under Blocking Tools

We now investigate several ways users may mitigate a clustering attack using blocking tools available at the time of measurement (June 2014). For all blocking configurations, we make measurements using the AOL browsing model and we examine the ability of an unrestricted adversary to cluster traffic. Measurements are run using several different privacy settings within the browser and two popular privacy and security add-ons.

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

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

Cookie blocking policies were more effective, particularly when an adversary saw a low number of page visits. We blocked cookies for sites that had not been visited

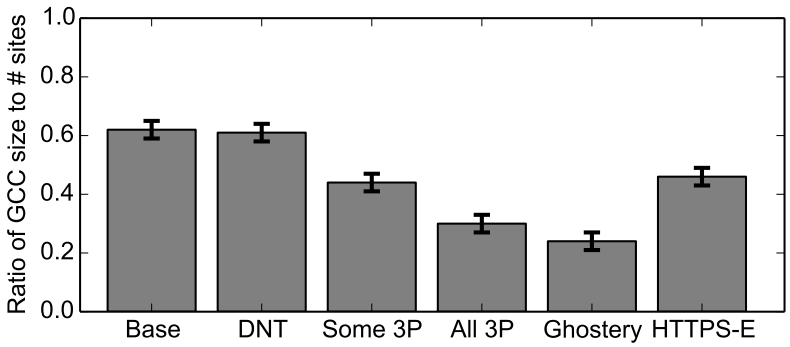


Figure 7.3: Clustering under several privacy settings as of June 2014.

in a first-party context by setting "Accept third-party cookies: From visited" in Firefox (this is also the default in Safari). When set, the level of clustering reduced to $43.9 \pm 3.2\%$. Blocking all third-party cookies further reduced this to $30.2 \pm 3.0\%$.

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

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

7.3.5 Identity Leakage

Table 7.2 summarizes our results from a manual survey of the Alexa U.S. sites. We picked the top 50 sites that supported account creation at the time of analysis (June 2014). 44 of the 50 websites used HTTPS to secure login pages.[6] Only 19 of those sites continued to use HTTPS to secure future interactions with the user after logged in. We summarize the cleartext identity leaks for the websites which no longer continue to use HTTPS after login.

Although a majority of sites secured user credentials on login pages, personally identifying information (name, username, email address) was transmitted much more frequently via HTTP. Over half of the surveyed sites leaked at least one type of identifier, and 42% (not shown in table) leaked either username or email address, which can be used to uniquely infer the user's real-world identity. Past work [183,210] has found a roughly equivalent level of leakage to occur through the `Request-URI` and `Referer` headers.

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

Table 7.2: Leakage on Alexa Top 50 supporting user accounts as of June 2014

Furthermore, we verified that pages from these popular sites that leaked identity occurred in the clusters of web pages found in our attack. Specifically, at least 5 (and an average 9.92) of the 28 sites we found to leak some type of identity were found in the giant connected component of every one of the 25 Alexa U.S. users. Due to global popularity of the top-50 U.S. sites, an average of 4.4 and 6.4 of these identity leaking sites were found in the GCC's of the Alexa Japan and Alexa Ireland

[6] 5 of the remaining 6 made POSTs with credentials and 1 made a GET with the credentials as a URL query parameter

users, respectively. Additionally, for the AOL profiles with no blocking, 9 of the 25 simulated users had at least 1 identity leaker in the GCC. Of course, there are likely also many sites outside the top 50 that leak identity and are found in these clusters, but we did not measure these.

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

7.4 Third parties impede HTTPS adoption

In the time since we made our measurements in 2014, browser vendors [52, 283], standards bodies [126, 247], and industry advocacy groups [113] have pushed for the adoption of HTTPS by default on the web. These efforts have included the creation of Let's Encrypt[7], a certificate authority which provides free TLS certificates. As a result, the web has seen an explosion of HTTPS adoption; Google's Transparency Report[8] shows that, within the United States, HTTPS page loads in the Chrome browser have risen from 44% of page loads in March 2015 to 81% in April 2018. The rise of HTTPS adoption will lessen the ability of a network adversary to perform the attacks described in this chapter.

In this section, we explore some of the remaining roadblocks to HTTPS adoption. One major roadblock identified by Publishers is the need to move all embedded third parties and trackers to HTTPS to avoid mixed-content errors [309, 331]. In January 2016 we examined HTTPS adoption on the top sites and found that trackers do indeed impede adoption. Specifically, we used the "Default Stateless" and "HTTPS Everywhere" measurement configurations described in Table 4.1.

[7] `https://letsencrypt.org/`

[8] `https://transparencyreport.google.com/https/overview`

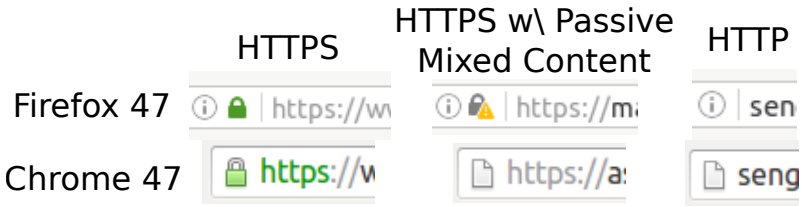


Figure 7.4: Secure connection UI for Firefox Nightly 47 and Chrome 47. Clicking on the lock icon in Firefox reveals the text "Connection is not secure" when mixed content is present.

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

Table 7.3: 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.

Table 7.3 shows the number of first-party sites that support HTTPS and the number that are HTTPS-only. At the time of measurement, the Google Transparency Report lists that 51% of Chrome page loads within the US used HTTPS. Despite this, our results reveal that HTTPS adoption by sites remained rather low.

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 [237]. *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

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

Table 7.4: 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.

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 7.4 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 displaying 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

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 7.5: 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.

content on these sites, summarized in Table 7.5. The majority are caused by third parties, rather than the site's own content, with a surprising 27% caused solely by trackers. Additional details on the method we used to determine which resources led to mixed content warnings is given in Section A.6 in the Appendix.

7.5 Discussion

7.5.1 Extending the attack: linking without IP address

So far we have assumed that the adversary sees the same source IP address on a request to a first-party site and its corresponding third-party tracker, and that this can be used to link the two requests. There are at least two scenarios in which this assumption is problematic. The first is a NAT. If two users, Alice and Bob, behind the same NAT visit the same web page at roughly the same time, the adversary sees the same IP address on all ensuing HTTP requests. The other scenario is when the user employs Tor without proper application layer anonymization, and the adversary is able to sniff cookies only on the path from the exit node to the web server. (If the user is using a properly configured Tor setup, such as the Tor browser bundle, this attack does not work at all). Since Tor will, in general, use different circuits for

communicating with different servers, the adversary will see different source IPs for the two requests (or may be able to observe only one of the requests).

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

7.5.2 Limitations

A couple of important limitations of the attack must be pointed out. First, using the Tor browser bundle likely defeats it. "Cross-origin identifier unlinkability" is a first-order design goal of the Tor browser, which is achieved through a variety of mechanisms such as double-keying cookies by first-party and third-party [270]. In other words, the same tracker on different websites will see different cookies. However, our measurements on identifier leakage on popular websites apply to Tor browser usage as well. Preventing such leakage is not a Tor browser design goal.

Simply disabling third-party cookies will also deter the attack, but it is not clear if it will completely stop it. There are a variety of stateful tracking mechanisms in addition to cookies [76, 152, 220, 347] and some tracking identifiers are based on content user identity (Chapter 6), although most are not as prevalent on the web as cookies are.

We also mention two limitations of our study. First, while a significant fraction of popular sites transmitted identities of logged-in users in the clear, we have not actually measured how frequently typical users are logged in to the sites that do

so. Anecdotal evidence suggests that this number must be high, but experiments on actual user sessions are required for an accurate estimation of vulnerability.

Second, while we take pains to model the adversary's view of our simulated users' traffic as if it were spread out over a 2-3 month period (as the searches in the AOL logs were), our actual crawls were conducted over a short time span (10 seconds between visits). There is one way in which our results might differ if our crawls were spread out: cookies might list expiry times well into the future, but servers might expire them ahead of schedule. To check that this would not be the case, we utilized the independent cookie measurement database *Cookiepedia*.[9] For a sample of cookies in our dataset with long expiry times, we manually verified that they are actually long-lived as measured in the wild by Cookiepedia.

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

7.6 Summary

While much has been said from a legal, ethical and policy standpoint about the revelations of NSA tracking, many interesting *technical* questions deserve to be considered. In this chapter we studied what can be inferred from the surveillance of web traffic and established that utilizing third-party tracking cookies enables an adversary to attribute traffic to users much more effectively than methods such as considering IP

[9] `http://cookiepedia.co.uk/`

address alone. We also showed that the main defense against network surveillance, i.e., the adoption of HTTPS on all websites, is slowed by the lack of HTTPS support by third-party services and web trackers.

Our findings provide support for recent efforts by browser vendors to restrict new APIs, particularly those that can be used for tracking, to "secure contexts" [167,322]. By validating the surveillance risks of HTTP tracking cookies, we hope that our work contributes to browser vendor discussions around deprecating HTTP cookies completely [336].

Chapter 8

Conclusion

This dissertation has shown that web tracking is sophisticated and pervasive. We showed that stateful tracking is present on nearly every website in the Alexa top 1 million sites, and is dominated by a small number of large companies. At the same time, we discovered smaller third parties using persistent tracking techniques that completely lack user control. These parties have an outsized impact on a user's privacy when compared to their presence on the web; identifier sharing through cookie syncing has the potential to inject persistent tracking information into the data used by major data brokers and advertisers. Indeed, we observed numerous instances of small parties sharing identifiers based on email addresses with major data brokers and web trackers.

Our work shows that even those users who take extreme steps to protect their privacy—by blocking trackers, disabling third-party cookies, or even using a browser that mitigates device fingerprinting—will still be exposed to web tracking of some form. Ultimately, there are few technical options to prevent the collection of personal information for the purposes of tracking, particularly if the first-party website is complicit. The injection of personal information into the web tracking identifier

graph has made it possible for companies to track across devices [65] and combine online tracking data with offline data sources.

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 dissertation, we've tried to bring this ambitious goal closer to reality. We have already seen an impact from our contributions. The research discoveries presented in this dissertation have led to number of privacy improvements in web browsers and web standards, including:

- The addition of an option to disable the Web Audio API in Firefox [71], which was later used for fingerprinting protection in the Tor Browser [269].

- The prioritization of fingerprinting protection features in the Brave browser, in particular to decision to defend against WebRTC fingerprinting [349].

- The removal of the Battery Status API from several browser engines, in part because of our discovery of abuse in the wild [87, 136].

- Enhancing the Disconnect and EasyPrivacy tracking protection lists. The added URLs include scripts which fingerprint [106], provide session replay services [80, 230], and those which abuse browser login managers [206].

- Safari [161] and Brave [310] disabled browser credential autofill without user form interaction following our discovery of abuse. Chrome changed their username autofill to require user interaction with the page [90]. Firefox is also considering restricting autofill [67].

- The addition of a fingerprinting considerations section to the Web Audio API specification [34].

In addition, our work has been used to inform policy makers, regulators, and advocacy groups. Researchers at the Federal Trade Commission (FTC) used OpenWPM in their cross-device tracking research [65]. Our work has also been used to brief the Federal Communication Commission (FCC) on the need for broadband privacy [243], to inform privacy professionals on the risks of session replay analytics scripts [146], and to argue for HTTPS adoption [149].

There have been promising developments by browser vendors and standards bodies in recognizing the need to address web tracking. Since 2017, Apple has shipped several privacy features aimed at reducing web tracking [133, 339]. During that time, Mozilla upstreamed and improved patches from the Tor Browser, which make it possible to enable fingerprint resistance in Firefox [233]. Recent specifications are starting to include text that directly addresses fingerprinting concerns early on in the design process—examples include the Media Capabilities API [188] and the Gamepad API [145].

Future work

We identify a number of possible directions for future work.

Automated detection of trackers with machine learning. In this dissertation we found that tracking protection lists continually missed trackers. Our methods are often semi-automated; a small set of "candidate" trackers are detected using automated methods, and then a human expert narrows down the candidate set to a set of known trackers. A promising direction is to use measurement and 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.

Mitigate tracking within the browser. Browsers currently enforce a limited number of restrictions on advertisers and web trackers. Instead, browsers can take a number of steps to make tracking more difficult and more transparent to the user:

- Choose more private defaults. With the exception of Safari [339], most of the major browsers have permissive defaults that do little to block tracking. Other browser vendors could move to default cookie policies that restrict storage access for third parties. Likewise, browsers can restrict APIs that are known to be abused by trackers, as was done for the Battery Status API (Section 5.2).

- Implement features to better restrict tracking scripts. All browsers currently rely on the first party to restrict trackers by embedding them in iframes or by using a Javascript sandboxing method [321]. Unfortunately, these features are not commonly used by first parties. Browser vendors can explore options for placing advertisers and tracking domains in sandboxed environments without the cooperation of the first-party website. Current proposals include `adFrame` [164] and Frozen Realms [307].

- Provide tracking indicators. Security indicators are common in browsers, and include things like the green padlock on the URL bar or TLS error pages when certificate validation fails. Chrome will display negative security indicators on HTTP connections starting July 2018 as a way to encourage HTTPS adoption [284]. Browsers have the opportunity to take a similar approach for web tracking. Many privacy addons already provide simple indicators of tracking, in the form of counts of the number of trackers (e.g., the Disconnect or uBlock Origin browser extensions) or as a letter grade (e.g., DuckDuckGo's Privacy Essentials browser extension). Other options include warning developers when trackers are present on pages with sensitive information, like a login box, or when trackers use invasive tracking techniques.

Automated measurement of tracking across desktop, mobile, and Internet of Things (IoT) devices. This dissertation has focused on automated measurement of trackers on the web, with a focus on desktop browses. While our methods naturally extend to web browsers on mobile devices, new methods and tools are needed to measure tracking in mobile applications and IoT devices. Several research groups have made process measuring tracking in mobile application [275, 291, 348], but IoT tracking measurement remains an area of future work.

Both this dissertation (Chapter 6) and past work [65] have found that a number of trackers collect enough personal information on the web seemingly for cross-device tracking. Likewise, preliminary measurements of cross-device tracking show that many of the same trackers are present on desktop and mobile devices [275]. We posit that the same is true for IoT devices [153]. Research which examines tracking across all three classes of devices is gravely needed to fully understand the extent of data collection by tracking companies.

The publisher's perspective. The publisher's perspective is often missing from web tracking measurement work. It would be helpful to better understand the relationship between publishers and the scripts they embed. Do publishers have an option for privacy protecting advertising and analytics? The failure of many sites to adapt to the recent EU General Data Protection Regulation (GDPR) [92] suggests that options are limited and difficult to deploy [274]. Likewise, it would be helpful to better understand the relationship between email senders and mailing list managers. To what extent is email tracking driven by senders versus mailing list managers? When a sender sets up a marketing campaign with a mailing list manager, is the tracking disclosed to the sender?

Summary

We have shown that web tracking is pervasive and increasing in persistence thanks to improved browser and web technologies. This may seem to paint a bleak picture that user tracking is unavoidable, however we find several promising signs that this is not the case. The transparency brought on by web measurement is an effective deterrent to web tracking, though measurement must happen regularly for the impact to last. Browser vendors and web standards bodies are starting to take an active role in mitigating web tracking, and have been responsive to measurement results. Likewise, we have seen sweeping privacy reform through the European Union's General Data Protection Regulation (GDPR). We are hopeful that automated tracking measurement will continue to provide transparency into new privacy-invasive practices, and that browser vendors, regulators, and policy makers will choose to stand on the side of protecting users.

Appendix A

Appendix

A.1 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.

A.2 Leak detection: encodings and hashes

Supported hashes and checksums: md2, md4, md5, sha, sha1, sha256, sha224, sha384, sha3-224, sha3-256, sha3-384, sha3-512, murmurhash2 (signed and unsigned), murmurhash3 32-bit, murmurhash3 64-bit, murmurhash3 128-bit, ripemd160, whirlpool, blake2b, blake2s, crc32, adler32

Supported encodings: base16, base32, base58, base64, urlencoding, deflate, gzip, zlib, entity, yenc

A.3 List of HTTP Respawning Scripts

First-Party Domains	Source of Respawn	Script Source
accountonline.com (citi.com), fling.com*, flirt4free.com, zoosk.com	Third-party: Iovation Fraud Detection	`https://mpsnare.iesnare.com/snare.jshttps://mpsnare.iesnare.com/stmgwb2.swf`
seoprofiler.com, seobook.com, bigrock.in, imperiaonline.org, mediatemple.net, resellerclub.com	First-party: Post Affiliate Pro Software	`http://seobook.com/aff/scripts/trackjs.js`
twitch.tv, justin.tv	Third-party: Shared CDN	`http://www-cdn.jtvnw.net/assets/global-6e555e3e646ba25fd387852cd97c19e1.js`
casino.com	First-party: Unknown/In-house	`http://www.casino.com/shared/js/mts.tracker.js`
xlovecam.com	First-party: Unknown/In-house	`http://www.xlovecam.com/colormaker.js`

Table A.1: Summary of HTTP respawning as of May 2014. "Source of Respawn" describes whether or not the tracking occurred in the first-party or third-party context and lists the entity responsible for the script. * Interestingly fling.com had the ID passed from the third-party context and saved in the first-party context

A.4 Fingerprinting script lists

Fingerprinting script	**# of sites**
`ct1.addthis.com/static/r07/core130.js`	5282
`i.ligatus.com/script/fingerprint.min.js`	115
`src.kitcode.net/fp2.js`	68
`admicro1.vcmedia.vn/fingerprint/figp.js`	31
`amazonaws.com/af-bdaz/bquery.js`	26
`*.shorte.st/js/packed/smeadvert-intermediate-ad.js`	14
`stat.ringier.cz/js/fingerprint.min.js`	4
`cya2.net/js/STAT/89946.js`	3
`images.revtrax.com/RevTrax/js/fp/fp.min.jsp`	3
`pof.com`	2
`*.rackcdn.com/mongoose.fp.js`	2
9 others	*9*
TOTAL	5559[1]

Table A.2: Canvas fingerprinting scripts found during a May 2014 measurement of the top 100,000 Alexa sites.
*: Some URLs are truncated or omitted for brevity.
1: There were a total of 5,542 unique sites. Some sites included canvas fingerprinting scripts from more than one domain.

Fingerprinting Script	# of sites
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[1]

Table A.3: Canvas fingerprinting scripts found during a January 2016 measurement of the Alexa top 1 million sites.
*: Some URLs are truncated for brevity.
1: There were a total of 14,371 unique sites. Some sites include fingerprinting scripts from more than one domain.

Fingerprinting Script	# of sites	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 A.4: WebRTC Local IP discovery on the Top Alexa 1 Million sites as measured in January 2016.
*: Some URLs are truncated for brevity.

A.5 OpenWPM mailing list form discovery method

This section provides additional detail on our mailing list detection methods used in Section 6.1.1.

Choosing pages on which to search for forms. OpenWPM searches through all links (`<a>` tags) on the landing page to find pages that are most likely to contain a mailing list form. It does this by matching the link text and URL against a ranked list of terms, which are shown in Table 6.1 in Section 6.1.1. As an initial step, we filter out invisible links and links to external sites. We check that the link text does not contain words in our blacklist, which aims to avoid unsubscribe pages and phone-based registration. If we have found any links that match, the crawler clicks on the one with the highest rank, then runs the form-finding procedure on the new page and any newly opened pop-up windows. If no forms are found, it goes back and repeats this process for the remaining links. The reason for clicking on generic article links is that we have come across several news sites with newsletter forms only within article pages. We also make sure to select the English language or US/English locale when available, since our keywords are in English.

Top-down form detection. For each page OpenWPM visits, it first searches through the HTML DOM for any potential email registration forms. When sites use the standard `<form>` element, it can simply iterate through each form's input fields (`<input>` tags) and see if any text fields ask for an email address (by matching on input type and keywords). If so, it marks the form as a candidate, and then chooses the best candidate using the following criteria (in order):

1. Always return the topmost form. Any form stacked on top of other elements is probably a modal or dialog, and we find that the most common use of these components is to promote a site's mailing lists. We rely on the z-index CSS property, which specifies the stacking order of an element in relation to others

(as a relative, arbitrary integer). Note that most DOM elements take the default z-index value of `auto`, inheriting the actual value from its parent; thus, the crawler recursively checks a form's parent elements until it finds a non-`auto` value, or reaches the root of the DOM tree. To break ties, it also searches for the literal strings "modal" or "dialog" within the form's HTML, since we find that such components are usually descriptively named.

2. Rank login forms lower. This is the other class of forms that often asks for an email address, so the crawler explicitly checks for the strings "login", "log in", and "sign in" within a form's HTML to avoid these when other candidates are present.

3. Prefer forms with more input fields. This is mainly helpful for identifying the correct follow-up form: if we submit our email address in the footer of a page, the same footer might be present on the page we get redirected to. In this scenario, the form we want to pick is the longer one.

Additionally, registration forms are sometimes found inside of inline frames (`<iframe>` tag), which are effectively separate HTML pages embedded in the main page. If necessary, we iterate through each frame and apply the same procedure to locate registration forms within them.

Bottom-up form detection. A growing number of sites place logical forms inside of generic container elements (e.g., `<div>` or `<span>` tags), without using any `<form>` tags. Therefore if top-down form detection fails, we take a bottom-up approach: the crawler first iterates through all the `<input>` elements on the page to check if any email address fields exist at all, then recursively examines their parents to find the first container that also contains a submit button. This container is usually the smallest logical form unit that includes all of the relevant input fields.

Determining form field type. Once a form is discovered, we need to determine which fields are contained in the form and fill each field with valid data. We skip any invisible elements, since a real user would not be expected to fill them. Some fields can be identified by their type attribute alone—for example, `tel` for phone numbers and `email` for email addresses—but these specific types were introduced in the relatively recent HTML5 standard [327], and most websites still use the general `text` type for all text inputs. In our survey of the top sites, we found that contextual hints are scattered across many tag attributes, with the most frequent being `name`, `class`, `id`, `placeholder`, `value`, `for`, and `title`. In addition, tags that contain HTML bodies (such as `<button>` tags) often contain hints in the `innerHTML`.

Handling two-part form submissions After submitting a form, we are sometimes prompted to fill out another longer form before the registration is accepted. This second form might appear on the same page (i.e., using JavaScript), or on a separate page either through a redirect or as a pop-up window. We take a simplistic approach: the crawler waits a few seconds, then applies the same form-finding procedure first on any pop-up windows and then on the original window. This approach may have the effect of submitting the same form twice, but we argue that this does not produce any adverse results—duplicate form submissions are a plausible user interaction that web services should be expected to handle gracefully.

A.6 Mixed content detection in HTTP data

To classify the cause of mixed content in Section 7.4, we used the following method. We classify trackers as described in Section 3.2.2. Additionally, we include a list of CDNs from the WebPagetest Project[1]. The mixed content URL is then classified according to the first rule it satisfies in the following list:

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

1. If the requested domain matches the landing page domain, and the request URL ends with `favicon.ico` classify as a "favicon".

2. If the requested domain matches the landing page domain, classify as the site's "own content".

3. If the requested domain is marked as "should block" by the blocklists, classify as "tracker".

4. If the requested domain is in the CDN list, classify as "CDN".

5. Otherwise, classify as "non-tracking" third-party content.

A.7 Content types for resources which caused mixed content errors.

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 A.5: 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.

A.8 Scripts exfiltrating information from browser login managers.

```
checkId: function(self) {
    var container = document.createElement('div');
    container.id = 'be-container';
    container.style.display = 'none';
    var form = document.createElement('form');
    form.attributes.autocomplete = 'on';
    var emailInput = document.createElement('input');
    emailInput.attributes.vcard_name = 'vCard.Email';
    emailInput.id = 'email';
    emailInput.type = 'email';
    emailInput.name = 'email';
    form.appendChild(emailInput);
    var passwordInput = document.createElement('input');
    passwordInput.id = 'password';
    passwordInput.type = 'password';
    passwordInput.name = 'password';
    form.appendChild(passwordInput);
    container.appendChild(form);
    document.body.appendChild(container);
    window.setTimeout(function() {
        if (self.emailRegexp.test(emailInput.value))
            self.sendHash(self, MD5(emailInput.value));
        document.body.removeChild(container)
    }, 1e3)
},
```

```
function ea() {
    var a = document.createElement("div");
    a.height = "1px";
    a.width = "1px";
    a.style.position = "absolute";
    a.style.top = "-42px";
    a.style.left = "-42px";
    a.style.display = "none";
    var b = document.getElementsByTagName("body")[0];
    b || (b = document.documentElement);
    b.appendChild(a);
    var c = document.createElement("form");
    c.method = "POST";
    c.action = "";
    a.appendChild(c);
    var d = document.createElement("input");
    try {
        d.type = "email"
    } catch (e) {
        d.type = "text"
    }
    d.name = "email";
    d.id = "pus-email";
    d.required = !0;
    d.autocomplete = "email";
    c.appendChild(d);
    d = document.createElement("input");
    d.type = "password";
    d.name = "password";
    d.id = "password";
    d.required = !0;
    d.autocomplete = "password";
    c.appendChild(d);
    t();
    setTimeout(function() {
        b.removeChild(a)
    }, 2500)
}
```

Figure A.1: Code snippets from OnAudience (left) and Adthink (right) that were responsible for the injection of invisible login forms.

Bibliography

[1] Adblock Plus - Surf the web without annoying ads! https://adblockplus.org/. Online; accessed 2017-09-05.

[2] BeautifulSoup. https://www.crummy.com/software/BeautifulSoup/. Online; accessed 2017-09-05.

[3] BlockListParser. https://github.com/shivamagarwal-iitb/BlockListParser. Online; accessed 2017-09-05.

[4] The design and implementation of the tor browser [draft]. https://www.torproject.org/projects/torbrowser/design/. Accessed: 2017-01-25.

[5] EasyList and EasyPrivacy. https://easylist.to/. Online; accessed 2017-09-05.

[6] Fingerprinting browsers using protocol handlers. https://itsecuritysolutions.org/2010-03-29_fingerprinting_browsers_using_protocol_handlers/. Accessed: 2017-01-25.

[7] Panopticlick: How unique - and trackable - is your browser? https://panopticlick.eff.org.

[8] ShareMeNot: Protecting against tracking from third-party social media buttons while still allowing you to use them. https://sharemenot.cs.washington.edu.

[9] TrackingObserver: A Browser-Based Web Tracking Detection Platform. http://trackingobserver.cs.washington.edu.

[10] uBlock Origin - An efficient blocker for Chromium and Firefox. Fast and lean. https://github.com/gorhill/uBlock/. Online; accessed 2017-09-05.

[11] W3C Technical Report Development Process. https://www.w3.org/2018/Process-20180201/. Accessed 2018-03-25.

[12] Executive Order 12333–United States intelligence activities. http://www.archives.gov/federal-register/codification/executive-order/12333.html, 1981.

[13] Device and Sensors Working Group. http://www.w3.org/2009/dap/, 2009. Accessed: 15.02.17.

[14] Privacy Interest Group Charter. https://www.w3.org/2011/07/privacy-ig-charter, 2011.

[15] NSA 'planned to discredit radicals over web-porn use'. http://www.bbc.co.uk/news/technology-25118156, November 2013.

[16] CSS Support Guide for Email Clients. Campaign Source, https://www.campaignmonitor.com/css/ (Archive: https://www.webcitation.org/6rLLXBX0E), 2014.

[17] Doubleclick ad exchange real-time bidding protocol: Cookie matching. https://developers.google.com/ad-exchange/rtb/cookie-guide, February 2014.

[18] Issue 507703. Ads using navigator.vibrate. https://bugs.chromium.org/p/chromium/issues/detail?id=507703, 2015.

[19] Battery Status API. https://www.chromestatus.com/feature/4537134732017664, 2015.

[20] Chinese planning outline for a social credit system. https://www.wired.com/beyond-the-beyond/2015/06/chinese-planning-outline-social-credit-system/, 2015. Accessed: 2018.

[21] Webrtc privacy. https://mozillamediagoddess.org/2015/09/10/webrtc-privacy/, 12 2015. Accessed: 2017-02-09.

[22] Advanced tor browser fingerprinting. http://jcarlosnorte.com/security/2016/03/06/advanced-tor-browser-fingerprinting.html, 2016. Accessed: 2017-01-25.

[23] How much do ad-tech vendors screen out cloud-based browsers? https://www.mezzobit.com/ad-tech-cloud-traffic-fraud/, 2016.

[24] HTML 5.1 W3C Recommendation, 1 November 2016. https://www.w3.org/TR/html51/browsers.html#top-level-browsing-context, 2016.

[25] Big data meets big brother as china moves to rate its citizens. https://www.wired.co.uk/article/chinese-government-social-credit-score-privacy-invasion, 2017. Accessed: 2018.

[26] Zestfinance introduces machine learning platform to underwrite millennials and other consumers with limited credit history. https://www.businesswire.com/news/home/20170214005357/en/ZestFinance-Introduces-Machine-Learning-Platform-Underwrite-Millennials, 2017. Accessed: 2018.

[27] Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach. https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election, 2018. Accessed: 2018.

[28] Gunes Acar, Steven Englehardt, and Arvind Narayanan. No boundaries for user identities: Web trackers exploit browser login managers. https://freedom-to-tinker.com/2017/12/27/no-boundaries-for-user-identities-web-trackers-exploit-browser-login-managers/, 2017.

[29] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of CCS*, 2014.

[30] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. FPDetective: dusting the web for fingerprinters. In *Proceedings of CCS*. ACM, 2013.

[31] Acxiom. Response letter to U.S. congress inquiry. https://web.archive.org/web/20130425093308/http:/markey.house.gov/sites/markey.house.gov/files/documents/Acxiom.pdf, 2012. Accessed: 2018-04-02.

[32] Acxiom. Understanding Acxiom's marketing products. https://www.acxiom.com/wp-content/uploads/2013/09/Acxiom-Marketing-Products.pdf, 2013. Accessed: 2018-04-02.

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

[34] Paul Adenot and Raymond Toy. Web Audio API – Editor's Draft. https://webaudio.github.io/web-audio-api/, 2018. Accessed: 2018-06-03.

[35] Adobe. Shared objects. https://help.adobe.com/en_US/as3/dev/WS5b3ccc516d4fbf351e63e3d118a9b90204-7d80.html.

[36] Gaurav Aggarwal, Elie Bursztein, Collin Jackson, and Dan Boneh. An analysis of private browsing modes in modern browsers. In *Proceedings of the 19th USENIX Conference on Security*, USENIX Security'10, pages 6–6, Berkeley, CA, USA, 2010. USENIX Association.

[37] Ehsan Akhgari. An overview of online ad fraud. https://ehsanakhgari.org/blog/2018-03-13/an-overview-of-online-ad-fraud, 2018.

[38] Furkan Alaca and P. C. van Oorschot. Device fingerprinting for augmenting web authentication: Classification and analysis of methods. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ACSAC '16, pages 289–301, New York, NY, USA, 2016. ACM.

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

[40] Amelia Andersdotter and Anders Jensen-Urstad. Evaluating websites and their adherence to data protection principles: Tools and experiences. In *IFIP International Summer School on Privacy and Identity Management*, pages 39–51. Springer, 2016.

[41] Julia Angwin. Meet the Online Tracking Device That is Virtually Impossible to Block. `https://www.propublica.org/article/meet-the-online-tracking-device-that-is-virtually-impossible-to-block`, 2014.

[42] Julia Angwin. Why online tracking is getting creepier. *ProPublica*, Jun 2014.

[43] Julia Angwin, Surya Mattu, and Terry Parris Jr. Facebook Doesnt Tell Users Everything It Really Knows About Them. `https://www.propublica.org/article/facebook-doesnt-tell-users-everything-it-really-knows-about-them`, 2016.

[44] Apple. About the security content of Safari 11.1. `https://support.apple.com/en-us/HT208695`.

[45] Axel Arnbak and Sharon Goldberg. Loopholes for circumventing the constitution: Warrantless bulk surveillance on americans by collecting network traffic abroad, 2014.

[46] Daniel Arp, Erwin Quiring, Christian Wressnegger, and Konrad Rieck. Privacy threats through ultrasonic side channels on mobile devices. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*, pages 35–47. IEEE, 2017.

[47] Mika Ayenson, Dietrich J Wambach, Ashkan Soltani, Nathan Good, and Chris J Hoofnagle. Flash cookies and privacy II: Now with HTML5 and ETag respawning. *World Wide Web Internet And Web Information Systems*, 2011.

[48] Mahesh Balakrishnan, Iqbal Mohomed, and Venugopalan Ramasubramanian. Where's that phone?: geolocating IP addresses on 3G networks. In *Internet Measurement Conference (IMC)*. ACM, 2009.

[49] Rebecca Balebako, Pedro Leon, Richard Shay, Blase Ur, Yang Wang, and L Cranor. Measuring the effectiveness of privacy tools for limiting behavioral advertising. In *Web 2.0 Workshop on Security and Privacy*, 2012.

[50] James Ball. NSA stores metadata of millions of web users for up to a year, secret files show. `http://www.theguardian.com/world/2013/sep/30/nsa-americans-metadata-year-documents`, 2013.

[51] Bananatag. Email Tracking for Gmail, Outlook and other clients. `https://bananatag.com/email-tracking/`. Online; accessed 2017-09-04.

[52] Richard Barnes. Deprecating Non-Secure HTTP. https://blog.mozilla.org/security/2015/04/30/deprecating-non-secure-http/, 2015. Accessed: 2018-04-29.

[53] Adam Barth. HTTP State Management Mechanism. https://tools.ietf.org/html/rfc6265.

[54] Adam Barth. The Web Origin Concept. https://tools.ietf.org/html/rfc6454.

[55] Bernhard Bauer. Providing transparency and controls for Adobe Flash Player's local storage. https://blog.chromium.org/2011/04/providing-transparency-and-controls-for.html.

[56] Howard Beales. The value of behavioral targeting. 2010.

[57] Benedict Bender, Benjamin Fabian, Stefan Lessmann, and Johannes Haupt. E-mail tracking: Status quo and novel countermeasures. In *International Conference on Information Systems (ICIS)*, 12 2016.

[58] Frédéric Besson, Nataliia Bielova, Thomas Jensen, et al. Enforcing browser anonymity with quantitative information flow. 2014.

[59] Reuben Binns, Jun Zhao, Max Van Kleek, and Nigel Shadbolt. Measuring third party tracker power across web and mobile. *arXiv preprint arXiv:1802.02507*, 2018.

[60] Paul E. Black. Ratcliff/Obershelp pattern recognition. http://xlinux.nist.gov/dads/HTML/ratcliffObershelp.html, December 2004.

[61] Károly Boda, Ádám Máté Földes, Gábor György Gulyás, and Sándor Imre. Firegloves. https://fingerprint.pet-portal.eu/?menu=6. Accessed: 2017-01-25.

[62] Károly Boda, Ádám Máté Földes, Gábor György Gulyás, and Sándor Imre. User tracking on the web via cross-browser fingerprinting. In Peeter Laud, editor, *Information Security Technology for Applications*, pages 31–46, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[63] Hristo Bojinov, Yan Michalevsky, Gabi Nakibly, and Dan Boneh. Mobile device identification via sensor fingerprinting. *arXiv preprint arXiv:1408.1416*, 2014.

[64] Dominique Bongard. De-anonymizing users of french political forums. In *Hack.lu 2013 Conference*, 2013.

[65] Justin Brookman, Phoebe Rouge, Aaron Alva, and Christina Yeung. Cross-device tracking: Measurement and disclosures. *Proceedings on Privacy Enhancing Technologies*, 2017(2):133–148, 2017.

[66] Bugzilla. Clear Recent History with Cache or Offline Website Data doesn't clear QuotaManager storage and ServiceWorkers. https://bugzilla.mozilla.org/show_bug.cgi?id=1047098.

[67] Bugzilla. Consider making signon.autofillForms = false to be the default. https://bugzilla.mozilla.org/show_bug.cgi?id=1427543.

[68] Bugzilla. Implement clearing of IndexedDB in browsingData API. https://bugzilla.mozilla.org/show_bug.cgi?id=1333050.

[69] Bugzilla. Implement clearing of LocalStorage in browsingData API. https://bugzilla.mozilla.org/show_bug.cgi?id=1355576.

[70] Bugzilla. password manager + XSS = disaster. https://bugzilla.mozilla.org/show_bug.cgi?id=408531.

[71] Bugzilla. Pref to disable Web Audio API. https://bugzilla.mozilla.org/show_bug.cgi?id=1288359.

[72] Bugzilla. Stealing Firefox saved passwords. https://bugzilla.mozilla.org/show_bug.cgi?id=1107422.

[73] Bugzilla. Tracking using intermediate CA caching. https://bugzilla.mozilla.org/show_bug.cgi?id=1334485.

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

[75] T. Bujlow, V. Carela-Espaol, J. Sol-Pareta, and P. Barlet-Ros. A survey on web tracking: Mechanisms, implications, and defenses. *Proceedings of the IEEE*, 105(8):1476–1510, Aug 2017.

[76] Elie Bursztein. Tracking users that block cookies with a HTTP redirect. http://www.elie.net/blog/security/tracking-users-that-block-cookies-with-a-http-redirect, 2011.

[77] Marcos Caceres. Re: Notes of June 30 teleconference. https://lists.w3.org/Archives/Public/public-device-apis/2016Jul/0000.html, 2016.

[78] SL Yinzhi Cao and E Wijmans. (cross-)browser fingerprinting via os and hardware level features. In *Proceedings of the 2017 Network & Distributed System Security Symposium, NDSS*, volume 17, 2017.

[79] Michael T. Capizzi and Rick Ferguson. Loyalty trends for the twentyfirst century. *Journal of Consumer Marketing*, 22(2):72–80, 2005.

[80] carbureted. Add screen tracking services, bitcoin miners, miscellaneous third party analytics, and move wishabi.net to content. https://github.com/disconnectme/disconnect-tracking-protection/

commit/09c7b279a88c8eda1ae18fe7b405e6cc315d2855, 2017. Accessed: 2018-05-28.

[81] Juan Miguel Carrascosa, Jakub Mikians, Ruben Cuevas, Vijay Erramilli, and Nikolaos Laoutaris. I always feel like somebody's watching me: measuring online behavioural advertising. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, page 13. ACM, 2015.

[82] Biz Carson. You're more likely to order a pricey Uber ride if your phone is about to die. http://uk.businessinsider.com/people-with-low-phone-batteries-more-likely-to-accept-uber-surge-pricing-2016-5, 2016.

[83] Pew Research Center. Mobile Fact Sheet. http://www.pewinternet.org/fact-sheet/mobile/, 2018. [Online; accessed 2018-02-11].

[84] Keith Chen. This Is Your Brain On Uber. http://www.npr.org/2016/05/17/478266839/this-is-your-brain-on-uber, 2016.

[85] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *Security and Privacy (S&P)*. IEEE, 2010.

[86] Dawn Chmielewski. How 'Do Not Track' Ended Up Going Nowhere. https://www.recode.net/2016/1/4/11588418/how-do-not-track-ended-up-going-nowhere, 2016.

[87] Chris Peterson. Removing the Battery Status API? https://groups.google.com/forum/#!msg/mozilla.dev.platform/5U8NHoUY-1k/9ybyzQIYCAAJ, 2016.

[88] Wolfie Christl, Katharina Kopp, and Patrick Urs Riechert. Corporate surveillance in everyday life. *Cracked Labs*, 2017.

[89] Wolfie Christl and Sarah Spiekermann. *Networks of Control. A Report on Corporate Surveillance, Digital Tracking, Big Data & Privacy*. facultas, 2016.

[90] Chromium. Users can be tracked via password manager. https://bugs.chromium.org/p/chromium/issues/detail?id=798492.

[91] Andrew Clement. IXmaps–Tracking your personal data through the NSA's warrantless wiretapping sites. In *International Symposium on Technology and Society (ISTAS)*. IEEE, 2013.

[92] European Commission. 2018 reform of EU data protection rules. https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en. Accessed: 2018-06-23.

[93] Federal Trade Commission. Cross-Device Tracking: An FTC Staff Report. https://www.ftc.gov/system/files/documents/reports/cross-device-tracking-federal-trade-commission-staff-report-january-2017/ftc_cross-device_tracking_report_1-23-17.pdf, 2017. Accessed: 2018-02-01.

[94] ContactMonkey. Email Tracking for Outlook and Gmail. https://www.contactmonkey.com/email-tracking. Online; accessed 2017-09-04.

[95] Alissa Cooper, Hannes Tschofenig, Bernard Aboba, Jon Peterson, J Morris, Marit Hansen, and Rhys Smith. Rfc 6973 — privacy considerations for internet protocols. Technical report, IETF, 2013.

[96] Criteo. Data Collection and Use. https://www.criteo.com/privacy/. Accessed: 2018-05-12.

[97] Luke Crouch. Preventing data leaks by stripping path information in HTTP Referrers. https://blog.mozilla.org/security/2018/01/31/preventing-data-leaks-by-stripping-path-information-in-http-referrers/, 2018. Accessed: 2018-02-01.

[98] Anupam Das, Nikita Borisov, and Matthew Caesar. Tracking mobile web users through motion sensors: Attacks and defenses. In *NDSS '16: The 2016 Network and Distributed System Security Symposium*, 2016.

[99] Anupam Das, Nikita Borisov, Edward Chou, and Muhammad Haris Mughees. Smartphone fingerprinting via motion sensors: Analyzing feasibility at large-scale and studying real usage patterns. *arXiv preprint arXiv:1605.08763*, 2016.

[100] Amit Datta, Michael Carl Tschantz, and Anupam Datta. Automated experiments on ad privacy settings. *Privacy Enhancing Technologies*, 2015.

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

[102] Frank Dawson. Specification Privacy Assessment (SPA) . https://yrlesru.github.io/SPA/, 2013.

[103] Paul-Olivier Dehaye. pdehaye/BigOther. https://github.com/pdehaye/BigOther/tree/master/uber, November 2016.

[104] Sanorita Dey, Nirupam Roy, Wenyuan Xu, Romit Roy Choudhury, and Srihari Nelakuditi. Accelprint: Imperfections of accelerometers make smartphones trackable. In *NDSS*, 2014.

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

[106] Disconnect. Disconnect blocks new tracking device that makes your computer draw a unique image. https://blog.disconnect.me/disconnect-blocks-new-tracking-device-that-makes-your-computer-draw-a-unique-image/, 2014. Accessed: 2018-05-28.

[107] Nick Doty. Mitigating Browser Fingerprinting in Web Specifications. https://github.com/w3c/fingerprinting-guidance/issues/3, 2015.

[108] Nick Doty. Reviewing for privacy in internet and web standard-setting. In *Security and Privacy Workshops (SPW), 2015 IEEE*, pages 185–192. IEEE, 2015.

[109] Zakir Durumeric, David Adrian, Ariana Mirian, James Kasten, Elie Bursztein, Nicolas Lidzborski, Kurt Thomas, Vijay Eranti, Michael Bailey, and J Alex Halderman. Neither snow nor rain nor mitm...: An empirical analysis of email delivery security. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, pages 27–39. ACM, 2015.

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

[111] Brad Eidson. Bug 164213 - Remove Battery Status API from the tree . https://bugs.webkit.org/show_bug.cgi?id=164213, 2016.

[112] Jochen Eisinger and Emily Stark. Referrer Policy – W3C Candidate Recommendation. https://www.w3.org/TR/referrer-policy/, 2017. Accessed: 2018-02-01.

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

[114] Ben Elgin and Vernon Silver. The Surveillance Market and Its Victims. http://www.bloomberg.com/data-visualization/wired-for-repression/, 2011.

[115] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Trans. Comput. Syst.*, 32(2):5:1–5:29, June 2014.

[116] Steven Englehardt, Gunes Acar, and Arvind Narayanan. No boundaries: Exfiltration of personal data by session-replay scripts. https://freedom-to-tinker.com/2017/11/15/no-boundaries-exfiltration-of-personal-data-by-session-replay-scripts/, 2017.

[117] Steven Englehardt, Gunes Acar, and Arvind Narayanan. No boundaries for credentials: New password leaks to Mixpanel and Session Replay Companies. https://freedom-to-tinker.com/2018/02/26/no-boundaries-for-credentials-password-leaks-to-mixpanel-and-session-replay-companies/, 2018.

[118] Steven Englehardt, Gunes Acar, and Arvind Narayanan. Website operators are in the dark about privacy violations by third-party scripts. https://freedom-to-tinker.com/2018/01/12/website-operators-are-in-the-dark-about-privacy-violations-by-third-party-scripts/, 2018.

[119] Steven Englehardt, Jeffrey Han, and Arvind Narayanan. I never signed up for this! privacy implications of email tracking. *Proceedings on Privacy Enhancing Technologies*, 2018(1):109–126, 2018.

[120] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *ACM Conference on Computer and Communications Security*, 2016.

[121] Steven Englehardt, Dillon Reisman, Christian Eubank, Peter Zimmerman, Jonathan Mayer, Arvind Narayanan, and Edward 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.

[122] Christian Eubank, Marcela Melara, Diego Perez-Botero, and Arvind Narayanan. Shining the floodlights on mobile web tracking - a privacy survey. W2SP, 2013.

[123] Chris Evans, Chris Palmer, and Ryan Sleevi. Public Key Pinning Extension for HTTP. https://tools.ietf.org/html/rfc7469.

[124] Benjamin Fabian, Benedict Bender, and Lars Weimann. E-mail tracking in online marketing: Methods, detection, and usage. In *12th International Conference Business Informatics*, 03 2015.

[125] Ayman Farahat and Michael C Bailey. How effective is targeted advertising? In *Proceedings of the 21st international conference on World Wide Web*, pages 111–120. ACM, 2012.

[126] Stephen Farrell and Hannes Tschofenig. Pervasive Monitoring Is an Attack. https://datatracker.ietf.org/doc/rfc7258/, 2014. Accessed: 2018-04-29.

[127] 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.

[128] Edward W. Felten and Michael A. Schneider. Timing attacks on web privacy. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, CCS '00, pages 25–32, New York, NY, USA, 2000. ACM.

[129] R. Fielding and J. Reschke. RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. https://tools.ietf.org/html/rfc7231, 2014.

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

[131] Nathaniel Fruchter, Hsin Miao, Scott Stevenson, and Rebecca Balebako. Variations in tracking in relation to geographic location. In *Proceedings of W2SP*, 2015.

[132] Federal Trade Commission (FTC). VIZIO to Pay $2.2 Million to FTC, State of New Jersey to Settle Charges It Collected Viewing Histories on 11 Million Smart Televisions without Users' Consent. https://www.ftc.gov/news-events/press-releases/2017/02/vizio-pay-22-million-ftc-state-new-jersey-settle-charges-it, 2017. Accessed: 2018-03-31.

[133] Brent Fulgham. Protecting Against HSTS Abuse. https://webkit.org/blog/8146/protecting-against-hsts-abuse/, 2018. [Online; accessed 2018-05-28].

[134] FullStory. Terms & Conditions. https://web.archive.org/web/20171115044316/https://www.fullstory.com/legal/terms-and-conditions/, 2017. Accessed: 2018-05-09.

[135] Ryan Gallagher. Operation Socialist: The Inside Story of How British Spies Hacked Belgiums Largest Telco. https://firstlook.org/theintercept/2014/12/13/belgacom-hack-gchq-inside-story/, 2014.

[136] gameb0y. HTML5 Battery Status API. https://github.com/brave/browser-laptop/issues/1885, 2016. Accessed: 2018-05-28.

[137] Ghostery. Are we private yet? http://www.areweprivateyet.com/.

[138] Christopher Gillespie. Why both marketers and consumers love email. https://blog.liveintent.com/marketers-consumers-love-email/. Accessed: 2018-05-12.

[139] Gmail Help. Choose whether to show images. https://support.google.com/mail/answer/145919. Online; accessed 2017-09-06.

[140] Avi Goldfarb and Catherine E. Tucker. Privacy regulation and online advertising. *Management Science*, 57(1):57–71, 2011.

[141] Steven Goldfeder, Harry Kalodner, Dillon Reisman, and Arvind Narayanan. When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies. *arXiv preprint arXiv:1708.04748*, 2017.

[142] Rob Goldman and Alex Himel. Making Ads and Pages More Transparent. https://newsroom.fb.com/news/2018/04/transparent-ads-and-pages/, 2018.

[143] Jordan Golson. Uber knows you'll probably pay surge pricing if your battery is about to die. http://www.theverge.com/2016/5/20/11721890/uber-surge-pricing-low-battery, 2016.

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

[145] Scott Graham, Ted Mielczarek, Brandon Jones, and Steve Agoston. Gamepad API – Working Draft. https://www.w3.org/TR/2018/WD-gamepad-20180508/, 2018. Accessed: 2018-05-28.

[146] Stacey Gray. Understanding Session Replay Scripts a Guide for Privacy Professionals. https://fpf.org/2018/03/05/understanding-session-replay-scripts-a-guide-for-privacy-professionals/, 2018. Accessed: 2018-05-28.

[147] Seda Gürses and Jose M del Alamo. Privacy engineering: Shaping an emerging field of research and practice. *IEEE Security & Privacy*, 14(2):40–46, 2016.

[148] Dean Hachamovitch. Google Bypassing User Privacy Settings. https://blogs.msdn.microsoft.com/ie/2012/02/20/google-bypassing-user-privacy-settings/, 2012.

[149] Joseph Lorenzo Hall. Its Time to Move to HTTPS. https://cdt.org/blog/its-time-to-move-to-https/, 2016. Accessed: 2018-05-28.

[150] Aniko Hannak, Piotr Sapiezynski, Arash Molavi Kakhki, Balachander Krishnamurthy, David Lazer, Alan Mislove, and Christo Wilson. Measuring personalization of web search. In *Conference on World Wide Web*, 2013.

[151] Aniko Hannak, Gary Soeller, David Lazer, Alan Mislove, and Christo Wilson. Measuring price discrimination and steering on e-commerce web sites. In *14th Internet Measurement Conference*, 2014.

[152] Manoj Hastak and Mary J Culnan. Persistent and unblockable cookies using HTTP headers. http://www.nikcub.com/posts/persistant-and-unblockable-cookies-using-http-headers, 2011.

[153] Alex Hern. UK homes vulnerable to 'staggering' level of corporate surveillance. https://www.theguardian.com/technology/2018/jun/01/uk-homes-vulnerable-to-staggering-level-of-corporate-surveillance. Accessed: 2018-06-03.

[154] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naive-Bayes Classifier. In *Workshop on Cloud Computing Security (CCSW)*. ACM, 2009.

[155] Brad Hill. Is preventing browser fingerprinting a lost cause? In *Technical Plenary Advisory Committee (TPAC) Meetings Week*, 2012.

[156] Andrew Hintz. Fingerprinting Websites Using Traffic Analysis. In *Privacy Enhancing Technologies*. Springer, 2003.

[157] Jeff Hodges, Colin Jackson, and Adam Barth. HTTP Strict Transport Security (HSTS). `https://tools.ietf.org/html/rfc6797`.

[158] Ralph Holz, Johanna Amann, Olivier Mehani, Mohamed Ali Kâafar, and Matthias Wachs. TLS in the wild: An internet-wide analysis of tls-based protocols for electronic communication. In *23nd Annual Network and Distributed System Security Symposium, NDSS 2016, San Diego, California, USA, February 21-24, 2016*, 2016.

[159] Chris Jay Hoofnagle and Nathan Good. Web privacy census. *Available at SSRN 2460547*, 2012.

[160] HubSpot. Start Email Tracking Today. `https://www.hubspot.com/products/sales/email-tracking`. Online; accessed 2017-09-04.

[161] Apple Inc. CVE-2018-4137. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-4137`, 2018. Accessed: 2018-05-28.

[162] Marcus Niemietz Jrg Schwenk and Christian Mainka. Same-origin policy: Evaluation in modern browsers. In *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC, 2017. USENIX Association.

[163] Samy Kamkar. Evercookie - virtually irrevocable persistent cookies. `http://samy.pl/evercookie/`, Sep 2010.

[164] Josh Karlin. AdFrame. `https://github.com/jkarlin/ad-frame`. Accessed: 2018-06-03.

[165] Dan Keating, Kevin Schaul, and Leslie Shapiro. The Facebook ads Russians targeted at different groups. `https://www.washingtonpost.com/graphics/2017/business/russian-ads-facebook-targeting/?noredirect=on&utm_term=.a338e3b27e98`, 2017.

[166] Michael Kerrisk. strace(1) - linux manual page. `http://man7.org/linux/man-pages/man1/strace.1.html`, May 2014.

[167] Jonathan Kingston. Restricting AppCache to Secure Contexts. `https://blog.mozilla.org/security/2018/02/12/restricting-appcache-secure-contexts/`, 2018. Accessed: 2018-05-09.

[168] Tadayoshi Kohno, Andre Broido, and Kimberly C Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, 2005.

[169] Anssi Konstiainen. Allow use from within secure context and top-level browsing context only. `https://github.com/w3c/battery/issues/10`, 2017.

[170] Anssi Kostiainen. Vibration API. `https://www.w3.org/TR/vibration/`, 2016.

[171] Anssi Kostiainen. Battery Status Event Specification. `https://www.w3.org/TR/2011/WD-battery-status-20110426/`, April 2011.

[172] Anssi Kostiainen. Battery Status API. `https://www.w3.org/TR/2011/WD-battery-status-20111129/`, November 2011.

[173] Anssi Kostiainen and Mounir Lamouri. Battery Status API. `https://www.w3.org/TR/2014/CR-battery-status-20141209`, 2014.

[174] Anssi Kostiainen and Mounir Lamouri. Battery Status API. `https://www.w3.org/TR/battery-status/`, July 2016.

[175] Anssi Kostiainen and Mounir Lamouri. Battery Status API. `https://www.w3.org/TR/2016/PR-battery-status-20160329`, March 2016.

[176] Anssi Kostiainen and Mounir Lamouri. Battery Status API. `https://www.w3.org/TR/2012/CR-battery-status-20120508/`, May 2012.

[177] Andy Kowl. Data Leakage Devalues Publishers' Biggest Asset – Their Audience. `https://www.pubexec.com/post/data-leakage-devalues-publishers-biggest-asset-audience/`, 2017.

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

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

[180] Balachander Krishnamurthy, Konstantin Naryshkin, and Craig Wills. Privacy leakage vs. protection measures: the growing disconnect. In *Proceedings of W2SP*, volume 2, 2011.

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

[182] Balachander Krishnamurthy and Craig Wills. Privacy diffusion on the Web: a longitudinal perspective. In *International Conference on World Wide Web*, pages 541–550. ACM, 2009.

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

[184] Balachander Krishnamurthy and Craig E Wills. Privacy leakage in mobile online social networks. In *3rd conference on Online social networks*. USENIX Association, 2010.

[185] David M. Kristol. Http cookies: Standards, privacy, and politics. *ACM Trans. Internet Technology*, 1(2):151–198, November 2001.

[186] Rich LaBarca. The Facts About Our Use of a Canvas Element in Our Recent R&D Test. `https://www.addthis.com/blog/2014/07/23/the-facts-about-our-use-of-a-canvas-element-in-our-recent-rd-test/`, 2014.

[187] Anja Lambrecht and Catherine Tucker. When does retargeting work? information specificity in online advertising. *Journal of Marketing Research*, 50(5):561–576, 2013.

[188] Mounir Lamouri. Media Capabilities – Draft Community Group Report. `https://wicg.github.io/media-capabilities/`, 2018. Accessed: 2018-05-16.

[189] Mounir Lamouri, Marcos Cceres, and Jeffrey Yaskin. Permissions API. `https://w3c.github.io/permissions/`, 2016. Accessed: 15.02.17.

[190] Tobie Langel and Rick Waldron. Generic Sensors API. `https://www.w3.org/TR/generic-sensor/`, 2017.

[191] Pierre Laperdrix, Benoit Baudry, and Vikas Mishra. Fprandom: Randomizing core browser objects to break advanced device fingerprinting techniques. In *International Symposium on Engineering Secure Software and Systems*, pages 97–114. Springer, 2017.

[192] Pierre Laperdrix, Walter Rudametkin, and Benoit Baudry. Mitigating browser fingerprint tracking: multi-level reconfiguration and diversification. In *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 98–108. IEEE Press, 2015.

[193] Pierre Laperdrix, Walter Rudametkin, and Benoit 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.

[194] Mathias Lécuyer, Guillaume Ducoffe, Francis Lan, Andrei Papancea, Theofilos Petsios, Riley Spahn, Augustin Chaintreau, and Roxana Geambasu. Xray: Enhancing the web's transparency with differential correlation. In *USENIX Security Symposium*, 2014.

[195] Mathias Lecuyer, Riley Spahn, Yannis Spiliopolous, Augustin Chaintreau, Roxana Geambasu, and Daniel Hsu. Sunlight: Fine-grained targeting detection at scale with statistical confidence. In *Proceedings of CCS*. ACM, 2015.

[196] Micah Lee. Secret "BADASS" Intelligence Program Spied on Smartphones. `https://firstlook.org/theintercept/2015/01/26/secret-badass-spy-program/`, 2015.

[197] Pedro Giovanni Leon, Lorrie Faith Cranor, Aleecia M McDonald, and Robert McGuire. Token attempt: the misrepresentation of website privacy policies through the misuse of p3p compact policy tokens. In *Proceedings of the 9th annual ACM workshop on Privacy in the electronic society*, pages 93–104. ACM, 2010.

[198] Adam Lerner, Anna Kornfeld Simpson, Tadayoshi Kohno, and Franziska Roesner. Internet jones and the raiders of the lost trackers: An archaeological study of web tracking from 1996 to 2016. In *Proceedings of USENIX Security)*, 2016.

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

[200] Hannah Li and David Evans. Horcrux: A password manager for paranoids. *arXiv preprint arXiv:1706.05085*, 2017.

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

[202] Timothy Libert. An automated approach to auditing disclosure of third-party data collection in website privacy policies. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 207–216. International World Wide Web Conferences Steering Committee, 2018.

[203] Bin Liu, Anmol Sheth, Udi Weinsberg, Jaideep Chandrashekar, and Ramesh Govindan. AdReveal: improving transparency into online targeted advertising. In *Workshop on Hot Topics in Networks*. ACM, 2013.

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

[205] LiveIntent. What information do we collect? `https://liveintent.com/services-privacy-policy/`. Accessed: 2018-05-12.

[206] lukemulks. Block ad tracking scripts used with password managers. `https://github.com/disconnectme/disconnect-tracking-protection/issues/36`, 2018. Accessed: 2018-05-28.

[207] Max Maass, Pascal Wichmann, Henning Pridöhl, and Dominik Herrmann. Privacyscore: Improving privacy and security via crowd-sourced benchmarks of websites. In *Annual Privacy Forum*, pages 178–191. Springer, 2017.

[208] Doug Madory, Chris Cook, and Kevin Miao. Who Are the Anycasters? In *Proceedings of NANOG59*, 10 2013.

[209] Gregor Maier, Fabian Schneider, and Anja Feldmann. Nat usage in residential broadband networks. In *International Conference on Passive and Active Network Measurement*, pages 32–41. Springer, 2011.

[210] Delfina Malandrino, Andrea Petta, Vittorio Scarano, Luigi Serra, Raffaele Spinelli, and Balachander Krishnamurthy. Privacy awareness about information leakage: Who knows what about me? In *Workshop on Privacy in the Electronic Society*. ACM, 2013.

[211] Matthias Marx, Ephraim Zimmer, Tobias Mueller, Maximilian Blochberger, and Hannes Federrath. Hashing of personally identifiable information is not sufficient. *SICHERHEIT 2018*, 2018.

[212] Iraklis Mathiopoulos. Running hashcat v4.0.0 in Amazons AWS new p3.16xlarge instance. `https://medium.com/@iraklis/running-hashcat-v4-0-0-in-amazons-aws-new-p3-16xlarge-instance-e8fab4541e9b`, 2017. Accessed: 2018-05-09.

[213] Vasilios Mavroudis, Shuang Hao, Yanick Fratantonio, Federico Maggi, Christopher Kruegel, and Giovanni Vigna. On the privacy and security of the ultrasound ecosystem. *Proceedings on Privacy Enhancing Technologies*, 2017(2):95–112, 2017.

[214] Jonathan Mayer. Do Not Track Is No Threat to Ad-Supported Businesses. `http://cyberlaw.stanford.edu/node/6592`, 2011.

[215] Jonathan Mayer. Tracking the trackers: Self-help tools. `https://cyberlaw.stanford.edu/blog/2011/09/tracking-trackers-self-help-tools`, 2011.

[216] Jonathan Mayer. Tracking the trackers: Where everybody knows your username. `https://cyberlaw.stanford.edu/blog/2011/10/tracking-trackers-where-everybody-knows-your-username`, 2011.

[217] Jonathan Mayer. The Turn-Verizon Zombie Cookie. `http://webpolicy.org/2015/01/14/turn-verizon-zombie-cookie/`, 2015.

[218] Jonathan Mayer and Edward W. Felten. The Web is Flat. `http://webpolicy.org/2013/10/30/the-web-is-flat/`, 2013.

[219] Jonathan R Mayer. "any person... a pamphleteer": Internet anonymity in the age of web 2.0. 2009.

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

[221] Johan Mazel, Richard Garnier, and Kensuke Fukuda. A comparison of web privacy protection techniques. *arXiv preprint arXiv:1712.06850*, 2017.

[222] Aleecia M McDonald and Lorrie Faith Cranor. Americans' attitudes about internet behavioral advertising practices. In *Proceedings of the 9th annual ACM workshop on Privacy in the electronic society*, pages 63–72. ACM, 2010.

[223] Aleecia M McDonald and Lorrie Faith Cranor. Survey of the use of Adobe Flash Local Shared Objects to respawn HTTP cookies, a. *ISJLP*, 7, 2011.

[224] Mozilla Developer Network (MDN). Web Storage API. `https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API`.

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

[226] Microsoft. Platform Status Suggestions. `https://wpdev.uservoice.com/forums/257854-microsoft-edge-developer/suggestions/6263689-battery-status-api`, 2012.

[227] Jakub Mikians, László Gyarmati, Vijay Erramilli, and Nikolaos Laoutaris. Detecting price and search discrimination on the internet. In *Workshop on Hot Topics in Networks*. ACM, 2012.

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

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

[230] MonztA. (Comment) No boundaries: Exfiltration of personal data by session-replay scripts. `https://freedom-to-tinker.com/2017/11/15/no-boundaries-exfiltration-of-personal-data-by-session-replay-scripts/#comment-28428`, 2017. Accessed: 2018-05-28.

[231] Keaton Mowery, Dillon Bogenreif, Scott Yilek, and Hovav Shacham. Fingerprinting information in JavaScript implementations. In *Web 2.0 Workshop on Security and Privacy (W2SP)*, volume 2. IEEE, 2011.

[232] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in html5. *Proceedings of W2SP*, 2012.

[233] Mozilla. Security/Fingerprinting. https://wiki.mozilla.org/Security/Fingerprinting. Accessed: 2018-05-28.

[234] Mozilla. Firefox 10 for developers (release notes) . https://developer.mozilla.org/en-US/Firefox/Releases/10, 2012.

[235] Mozilla. Firefox 52 for developers (release notes) . https://developer.mozilla.org/en-US/Firefox/Releases/52#Others, 2017.

[236] Mozilla. Firefox 52 Release Notes. https://www.mozilla.org/en-US/firefox/52.0/releasenotes/, 2017.

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

[238] Mozilla Support. Remote Content in Messages. https://support.mozilla.org/en-US/kb/remote-content-in-messages. Online; accessed 2017-09-04.

[239] Martin Mulazzani, Philipp Reschl, Markus Huber, Manuel Leithner, Sebastian Schrittwieser, Edgar Weippl, and FH Campus Wien. Fast and reliable browser identification with JavaScript engine fingerprinting. In *Web 2.0 Workshop on Security and Privacy (W2SP)*, volume 1. IEEE, 2013.

[240] Steven J Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *Security and Privacy (S&P)*. IEEE, 2005.

[241] Steven J Murdoch and Piotr Zieliński. Sampled Traffic Analysis by Internet-Exchange-Level Adversaries. In *Privacy Enhancing Technologies*. Springer, 2007.

[242] Arvind Narayanan. There is no such thing as anonymous online tracking. http://cyberlaw.stanford.edu/blog/2011/07/there-no-such-thing-anonymous-online-tracking, 2011.

[243] Arvind Narayanan. RE: Docket No. 16-106, Protecting the Privacy of Customers of Broadband and Other Telecommunication Services. https://ecfsapi.fcc.gov/file/60002080817.pdf, 2016. Accessed: 2018-05-28.

[244] Christopher Neasbitt, Bo Li, Roberto Perdisci, Long Lu, Kapil Singh, and Kang Li. Webcapsule: Towards a lightweight forensic engine for web browsers. In *Proceedings of CCS*. ACM, 2015.

[245] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. Privaricator: Deceiving fingerprinters with little white lies.

[246] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Security and Privacy (S&P)*. IEEE, 2013.

[247] Mark Nottingham. Securing the Web. https://www.w3.org/2001/tag/doc/web-https, 2015. Accessed: 2018-04-29.

[248] Mark Nottingham. Unsanctioned Web Tracking. https://www.w3.org/2001/tag/doc/unsanctioned-tracking/, 2015.

[249] Andrew Odlyzko. Privacy, economics, and price discrimination on the Internet. In *ICEC2003: Fifth International Conference on Electronic Commerce*, 2003.

[250] Office of Management and Budget (OMB). Use of Web Measurement and Customization Technologies. https://www.treasury.gov/open/Documents/OMB%20M-10-22%20Required%20Additions%20to%20the%20Privacy%20Policy.pdf. Accessed: 2018-05-12.

[251] Office of the Directory of National Intelligence (DNI). Section 702 Overview. https://www.dni.gov/files/icotr/Section702-Basics-Infographic.pdf. Accessed: 2018-05-13.

[252] Lukasz Olejnik. Bug 1124127 - Round Off Navigator Battery Level on Linux. https://bugzilla.mozilla.org/show_bug.cgi?id=1124127, 2015.

[253] Lukasz Olejnik. Bug 1299454 - Round Off Ambient Light Sensor event.value . https://bugzilla.mozilla.org/show_bug.cgi?id=1299454, 2016.

[254] Lukasz Olejnik. Issue 661792 - Battery API raises privacy issues. https://bugs.chromium.org/p/chromium/issues/detail?id=661792, 2016.

[255] Lukasz Olejnik. Issue 661792 - Battery API raises privacy issues. https://bugs.chromium.org/p/chromium/issues/detail?id=661792#c4, 2016.

[256] Lukasz Olejnik, Gunes Acar, Claude Castelluccia, and Claudia Diaz. The leaking battery. *Cryptology ePrint Archive*, Report 2015/616, 2015.

[257] Lukasz Olejnik, Gunes Acar, Claude Castelluccia, and Claudia Diaz. *The Leaking Battery*. Data Privacy Management, 2015.

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

[259] Lukasz Olejnik, Claude Castelluccia, and Artur Janc. Why Johnny Can't Browse in Peace: On the Uniqueness of Web Browsing History Patterns. In *5th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2012)*, Vigo, Spain, July 2012.

[260] Lukasz Olejnik, Steven Englehardt, and Arvind Narayanan. Battery status not included: Assessing privacy in web standards. In *Proceedings of the 2017 International Workshop on Privacy Engineering (IWPE)*, 2017.

[261] Lukasz Olejnik, Steven Englehardt, and Arvind Narayanan. Battery status not included: Assessing privacy in web standards. In *Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs)*, 2017.

[262] Lukasz Olejnik, Tran Minh-Dung, Claude Castelluccia, et al. Selling Off Privacy at Auction. 2013.

[263] Olejnik, Lukasz. Issue 642731. Round Off Ambient. Light Sensor event.value. `https://bugs.chromium.org/p/chromium/issues/detail?id=642731`, 2016.

[264] Opera. Opera 26 Release Notes. `https://www.opera.com/docs/changelogs/unified/2600/`, 2014.

[265] Rebekah Overdorf, Mark Juarez, Gunes Acar, Rachel Greenstadt, and Claudia Diaz. How unique is your. onion?: An analysis of the fingerprintability of tor onion services. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2021–2036. ACM, 2017.

[266] Ramakrishna Padmanabhan, Amogh Dhamdhere, Emile Aben, Neil Spring, et al. Reasons dynamic addresses change. In *Proceedings of the 2016 Internet Measurement Conference*, pages 183–198. ACM, 2016.

[267] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2011.

[268] Ian Paul. How to automatically delete your cookies every time you close your browser. `https://www.pcworld.com/article/2846020/how-to-automatically-delete-your-cookies-every-time-you-close-your-browser.html`, 2014. [Online; accessed 2018-02-11].

[269] Mike Perry. Determine if AudioBuffers/OfflineAudioContext are a fingerprinting vector. `https://trac.torproject.org/projects/tor/ticket/13017`. Accessed: 2018-05-28.

[270] Mike Perry, Erinn Clark, and Steven Murdoch. The design and implementation of the Tor browser [DRAFT]. `https://www.torproject.org/projects/torbrowser/design`, November 2014.

[271] Chris Peterson. Bug 1313580 - Remove web content access to Battery API. `https://bugzilla.mozilla.org/show_bug.cgi?id=1313580`, 2016.

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

[273] Privacy and Civil Liberties Oversight Board (PCLOB). Report on the Surveillance Program Operated Pursuant to Section 702 of the Foreign Intelligence Surveillance Act. `https://www.pclob.gov/library/702-Report.pdf`, 2014. Accessed: 2018-05-13.

[274] Alison DeNisco Rayome. 65% of organizations will fail to meet critical GDPR compliance by deadline. https://www.techrepublic.com/article/65-of-organizations-will-fail-to-meet-critical-gdpr-compliance-by-deadline/, 2018. Accessed: 2018-05-28.

[275] Abbas Razaghpanah, Rishab Nithyanand, Narseo Vallina-Rodriguez, Srikanth Sundaresan, Mark Allman, Christian Kreibich, and Phillipa Gill. Apps, trackers, privacy, and regulators: A global study of the mobile tracking ecosystem. 2018.

[276] Andrew Reed and Michael Kranch. Identifying https-protected netflix videos in real-time. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 361–368. ACM, 2017.

[277] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. Recon: Revealing and controlling pii leaks in mobile network traffic. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 361–374. ACM, 2016.

[278] Philipp Richter, Florian Wohlfart, Narseo Vallina-Rodriguez, Mark Allman, Randy Bush, Anja Feldmann, Christian Kreibich, Nicholas Weaver, and Vern Paxson. A multi-perspective analysis of carrier-grade nat deployment. In *Proceedings of the 2016 Internet Measurement Conference*, IMC '16, pages 215–229, New York, NY, USA, 2016. ACM.

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

[280] Franziska Roesner, Tadayoshi Kohno, and David Wetherall. Detecting and Defending Against Third-Party Tracking on the Web. In *Symposium on Networking Systems Design and Implementation*. USENIX, 2012.

[281] Jonathan Rosenberg and Henning Schulzrinne. An Offer/Answer Model with the Session Description Protocol (SDP). https://tools.ietf.org/html/rfc3264.

[282] Sonam Samat, Alessandro Acquisti, and Linda Babcock. Raise the curtains: The effect of awareness about targeting on consumer attitudes and purchase intentions. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 299–319. USENIX Association, 2017.

[283] Emily Schechter. Moving towards a more secure web. https://security.googleblog.com/2016/09/moving-towards-more-secure-web.html, 2016. Accessed: 2018-04-29.

[284] Emily Schechter. A secure web is here to stay. https://security.googleblog.com/2018/02/a-secure-web-is-here-to-stay.html, 2018. Accessed: 2018-06-03.

[285] Sebastian Schelter and Jérôme Kunegis. On the ubiquity of web tracking: Insights from a billion-page web crawl. *arXiv preprint arXiv:1607.07403*, 2016.

[286] Steven Schmeiser. Sharing audience data: Strategic participation in behavioral advertising networks. *Review of Industrial Organization*, pages 1–22, 2015.

[287] Steven Schmeiser. Online advertising networks and consumer perceptions of privacy. *Applied Economics Letters*, pages 1–5, 2017.

[288] scikit-learn. Jaccard Similarity Score. `http://scikit-learn.org/stable/modules/generated/sklearn.metrics.jaccard_similarity_score.html`. Online; accessed 2017-09-05.

[289] Ory Segal, Aharon Fridman, and Elad Shuster. Passive Fingerprinting of HTTP/2 Clients. `https://www.akamai.com/uk/en/multimedia/documents/white-paper/passive-fingerprinting-of-http2-clients-white-paper.pdf`, 2017. [Online; accessed 2018-02-11].

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

[291] Suranga Seneviratne, Harini Kolamunna, and Aruna Seneviratne. A measurement study of tracking in paid mobile applications. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '15, pages 7:1–7:6, New York, NY, USA, 2015. ACM.

[292] SessionCam. What information do we collect for our clients? `https://web.archive.org/web/20171115050443/https://sessioncam.com/privacy-policy-cookies/`, 2017. Accessed: 2018-05-09.

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

[294] Kapil Singh, Alexander Moshchuk, Helen J Wang, and Wenke Lee. On the incoherencies in web browser access control policies. In *Proceedings of S&P*. IEEE, 2010.

[295] Ashkan Soltani, Shannon Canty, Quentin Mayo, Lauren Thomas, and Chris Jay Hoofnagle. Flash cookies and privacy. In *AAAI Spring Symposium: Intelligent Information Privacy Management*, 2010.

[296] Ashkan Soltani, Andrea Peterson, and Barton 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.

[297] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing Analysis of Keystrokes and Timing Attacks on SSH. In *Security Symposium*. USENIX, 2001.

[298] Aditya K Sood and Richard J Enbody. Malvertising–exploiting web advertising. *Computer Fraud & Security*, 2011(4):11–16, 2011.

[299] Ove Sorensen. Zombie-cookies: Case studies and mitigation. In *Internet Technology and Secured Transactions (ICITST)*, pages 321–326. IEEE, 2013.

[300] Jan Spooren, Davy Preuveneers, and Wouter Joosen. Leveraging battery usage from mobile devices for active authentication. 2017.

[301] Oleksii Starov, Johannes Dahse, Syed Sharique Ahmad, Thorsten Holz, and Nick Nikiforakis. No honor among thieves: A large-scale analysis of malicious web shells. In *International Conference on World Wide Web*, 2016.

[302] Oleksii Starov, Phillipa Gill, and Nick Nikiforakis. Are you sure you want to contact us? quantifying the leakage of pii via website contact forms. *Proceedings on Privacy Enhancing Technologies*, 2016(1):20–33, 2016.

[303] Oleksii Starov and Nick Nikiforakis. Extended tracking powers: Measuring the privacy diffusion enabled by browser extensions. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1481–1490. International World Wide Web Conferences Steering Committee, 2017.

[304] Mark Stockley. Anatomy of a browser dilemma how HSTS supercookies make you choose between privacy or security. `https://nakedsecurity.sophos.com/2015/02/02/anatomy-of-a-browser-dilemma-how-hsts-supercookies-make-you-choose-between-privacy-or-security/`, 2015. [Online; accessed 2018-02-11].

[305] Jessica Su, Ansh Shukla, Sharad Goel, and Arvind Narayanan. De-anonymizing web browsing data with social networks. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1261–1269. International World Wide Web Conferences Steering Committee, 2017.

[306] Mozilla Support. Disable third-party cookies in Firefox to stop some types of tracking by advertisers. `https://support.mozilla.org/en-US/kb/disable-third-party-cookies`. Accessed: 2018-06-22.

[307] tc39. Draft Proposed Frozen Realm API. `https://github.com/tc39/proposal-frozen-realms`. Accessed: 2018-06-03.

[308] 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.

[309] Zack 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.

[310] Anthony Tseng. Disable password autofill on page load. https://github.com/brave/browser-laptop/issues/12489, 2018. Accessed: 2018-05-28.

[311] Joseph Turow, Lauren Feldman, and Kimberly Meltzer. Open to exploitation: America's shoppers online and offline. *Departmental Papers (ASC)*, page 35, 2005.

[312] Joseph Turow, Jennifer King, Chris Jay Hoofnatle, Amy Bleakley, and Michael Hennessy. Americans reject tailored advertising and three activities that enable it. *Departmental Papers (ASC)*, 2009.

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

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

[315] Thomas Unger, Martin Mulazzani, Dominik Fruhwirt, Markus Huber, Sebastian Schrittwieser, and Edgar Weippl. SHPF: Enhancing HTTP(S) Session Security with Browser Fingerprinting. In *Availability, Reliability and Security (ARES)*, pages 255–261. IEEE, 2013.

[316] Blase Ur, Pedro Giovanni Leon, Lorrie Faith Cranor, Richard Shay, and Yang Wang. Smart, useful, scary, creepy: perceptions of online behavioral advertising. In *Eighth Symposium on Usable Privacy and Security*. ACM, 2012.

[317] Jennifer Valentino-Devries, Jeremy Singer-Vine, and Ashkan Soltani. What they know. http://online.wsj.com/public/page/what-they-know-digital-privacy.html, 2012.

[318] Narseo Vallina-Rodriguez, Christian Kreibich, Mark Allman, and Vern Paxson. Lumen: Fine-grained visibility and control of mobile traffic in user-space. 2017.

[319] Narseo Vallina-Rodriguez, Srikanth Sundaresan, Christian Kreibich, and Vern Paxson. Header enrichment or isp enrichment?: Emerging privacy threats in mobile networks. In *Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization*, pages 25–30. ACM, 2015.

[320] Narseo Vallina-Rodriguez, Srikanth Sundaresan, Abbas Razaghpanah, Rishab Nithyanand, Mark Allman, Christian Kreibich, and Phillipa Gill. Tracking the trackers: Towards understanding the mobile advertising and tracking ecosystem. In *1st Data and Algorithm Transparency (DAT) Workshop*, 2016.

[321] Steven Van Acker and Andrei Sabelfeld. Javascript sandboxing: Isolating and restricting client-side javascript. In *Foundations of Security Analysis and Design VIII*, pages 32–86. Springer, 2015.

[322] Anne van Kesteren. Secure Contexts Everywhere. https://blog.mozilla.org/security/2018/01/15/secure-contexts-everywhere/, 2018. Accessed: 2018-05-09.

[323] Eline Vanrykel, Gunes Acar, Michael Herrmann, and Claudia Diaz. Leaky birds: Exploiting mobile application traffic for surveillance. In Jens Grossklags and Bart Preneel, editors, *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, 2017.

[324] Thomas Vissers, Nick Nikiforakis, Nataliia Bielova, and Wouter Joosen. Crying wolf? on the price discrimination of online airline tickets. HotPETS, 2014.

[325] Tim Volodine. Issue 1229143006: Enforce restricted precision of the Battery Status API level attribute (Closed) . https://codereview.chromium.org/1229143006, 2015.

[326] Tanvi Vyas, Andrea Marchesini, and Christoph Kerschbaumer. Extending the same origin policy with origin attributes. In *3rd International Conference on Information Systems Security and Privacy*, pages 464–473, 01 2017.

[327] W3C. 4.10 Forms - HTML5. https://www.w3.org/TR/html5/forms.html. Online; accessed 2017-09-07.

[328] W3C. Indexed Database API 2.0. https://www.w3.org/TR/IndexedDB-2/.

[329] W3C. WebVR. https://w3c.github.io/webvr/spec/1.1/, 2017.

[330] Kent Walker. Supporting election integrity through greater advertising transparency. https://www.blog.google/topics/public-policy/supporting-election-integrity-through-greater-advertising-transparency/, 2018.

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

[332] Web Bluetooth Community Group. Web Bluetooth API. https://webbluetoothcg.github.io/web-bluetooth/, 2017.

[333] WebKit. Changeset 110991. Support for Battery Status API. https://trac.webkit.org/changeset/110991, 2012.

[334] Mike West. Self-Review Questionnaire: Security and Privacy. https://w3ctag.github.io/security-questionnaire/, 2015. Accessed: 25.10.15.

[335] Mike West. Referrer Policy – W3C Candidate Recommendation. https://www.w3.org/TR/credential-management-1/, 2017. Accessed: 2018-05-09.

[336] Mike West. Cookies-over-HTTP Bad. https://github.com/mikewest/cookies-over-http-bad, 2018. Accessed: 2018-05-09.

[337] WHATWG. HTML Living Standard — iframe sandbox attributes. https://html.spec.whatwg.org/multipage/iframe-embed-object.html#attr-iframe-sandbox.

[338] Andrew M White, Austin R Matthews, Kevin Z Snow, and Fabian Monrose. Phonotactic reconstruction of encrypted VoIP conversations: Hookt on fon-iks. In *Security and Privacy (S&P)*. IEEE, 2011.

[339] John Wilander. Intelligent Tracking Prevention. https://webkit.org/blog/7675/intelligent-tracking-prevention/, 2017. [Online; accessed 2018-02-11].

[340] Xinyu Xing, Wei Meng, Dan Doozan, Nick Feamster, Wenke Lee, and Alex C Snoeren. Exposing inconsistent web search results with bobble. In *Passive and Active Measurement*, pages 131–140. Springer, 2014.

[341] Yahoo Help. Block images in your incoming Yahoo Mail emails. https://help.yahoo.com/kb/SLN5043.html. Online; accessed 2017-09-06.

[342] Jun Yan, Ning Liu, Gang Wang, Wen Zhang, Yun Jiang, and Zheng Chen. How much can behavioral targeting help online advertising? In *Proceedings of the 18th international conference on World wide web*, pages 261–270. ACM, 2009.

[343] Yandex Browser Blog. Beware Evil APIs . https://browser.yandex.com/blog/beware-evil-apis, 2016.

[344] Yaxing Yao, Davide Lo Re, and Yang Wang. Folk models of online behavioral advertising. In *CSCW*, pages 1957–1969, 2017.

[345] Ting-Fang Yen, Yinglian Xie, Fang Yu, Roger Peng Yu, and Martın Abadi. Host fingerprinting and tracking on the web: Privacy and security implications. In *Network and Distributed System Security Symposium (NDSS)*. IEEE, 2012.

[346] Zhonghao Yu, Sam Macbeth, Konark Modi, and Josep M Pujol. Tracking the trackers. In *Proceedings of the 25th International Conference on World Wide Web*, pages 121–132. International World Wide Web Conferences Steering Committee, 2016.

[347] Michal Zalewski. Rapid history extraction through non-destructive cache timing (v8). http://lcamtuf.coredump.cx/cachetime/. Accessed: 2014.

[348] Jinyan Zang, Krysta Dummit, James Graves, Paul Lisker, and Latanya Sweeney. Who knows what about me? a survey of behind the scenes personal data sharing to third parties by mobile apps. *Technology Science*, 30, 2015.

[349] Yan Zhu. Fingerprinting Protection Mode. https://github.com/brave/browser-laptop/wiki/Fingerprinting-Protection-Mode, 2018. Accessed: 2018-05-28.

[350] Yan Zhu and Mike West. Secure Contexts. `https://www.w3.org/TR/secure-contexts/`, 2016.

[351] Sebastian Zimmeck, Jie S Li, Hyungtae Kim, Steven M Bellovin, and Tony Jebara. A privacy analysis of cross-device tracking. In *Proceedings of the 26th USENIX Security Symposium*, 2017.