Web Privacy Measurement: Early Results, Engineering Challenges and Selected Applications

Christian Gabriel Eubank

Master's Thesis

Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Master of Science in Engineering

Recommended for Acceptance
by the Department of Computer Science
Princeton University

Adviser: Arvind Narayanan

June 2014

Abstract

Web Privacy Measurement experiments have heavily influenced privacy debates by shedding light into practices such as third party online tracking and price discrimination. However, these research experiments have typically been one-off projects, with different groups encountering similar methodological and engineering challenges without forming an institutional knowledge base. As an illustrative example of this trend of repeating effort through self-contained experiments, we present the results of one study comparing mobile and desktop tracking and discuss how challenges from this study and other works in the literature shaped the formation of general design principles intended to improve the efficiency of such experiments.

We present a robust and modular web measurement platform that enables scalable and repeatable experiments while avoiding many common pitfalls observed by the research community. We describe case studies performed on this framework, including the detection of unique cookie identifiers, the detection of third-party synchronization of these IDs and an examination into the personalization of the content on news sites based on user history.

Acknowledgements

I would like to thank my adviser, Arvind Narayanan, for his invaluable advice and guidance these past two years. I entered the M.S.E. program with very little experience in the realm of privacy and security and, for that matter, with research in general. I am proud of the research I've produced with Arvind and the WPM group and know the skills that I gained researching with them will serve me well in the future.

I've been very lucky to have such wonderful collaborators and friends in the WPM lab, in particular: Steven Englehardt, Peter Zimmerman and Dillon Reisman. It has been really exciting to watch the lab grow in size and experience and I could not think of a better group of people with whom I'd rather spend my time in and outside the office. I'd also like to thank Marcela Melara, Diego Perez-Botero, Gunes Acar, Marc Juarez and Claudia Diaz for being such wonderful collaborators on other works. The Princeton Computer Science Department as a whole has been a great source of support these past two years.

Finally, I'd like to thank my friends at Yale and my parents for encouraging my switch to computer science. Changing my trajectory from applying to law school to enrolling in computer science grad school required a lot of thought but it has been one of the best decisions I've made. I'd especially like to thank my parents for supporting me throughout the years both in educational pursuits and in general.

Contents

Chapter 1

Introduction

Web Privacy Measurement - the analysis and collection of data while browsing online to understand the flow and use of personal information - serves as a powerful tool in privacy and policy debates. Already, smaller-scale studies with wide media exposure [20],[32] have revealed practices such as price discrimination and price re-ordering, sparking public outcry and increased regulation. While we reserve judgement on individual practices by third-party trackers and data consumers, Web Privacy Measurement, or WPM, is valuable in reducing information asymmetry by bringing these practices to light.

Despite abundant anecdotal evidence of practices such as targeted ads, for WPM to have a serious role in privacy discussions, it must possess scientific rigor. Achieving this level of rigor is a challenging task both due to engineering and measurement concerns as well as the current disparateness of the field. In particular, performing large measurements requires a high degree of automation, but the dynamic nature of the web renders even the creation of a platform capable of driving a full browser instance over tens of thousands of sites in a stable manner a non-trivial task. The related issue of ensuring that browser profiles remain consistent throughout these crawls is another fundamental engineering challenge.

In [8] we surveyed 32 papers in the field of WPM, considering both their areas of inquiry as well as their engineering challenges. One major trend in this survey was that the various research groups performed experiments using one-off platforms on private codebases. The first major drawback of this approach is that research groups unknowingly duplicated efforts when solving the same problems in WPM. Not only only does this waste effort and lead to slightly different solutions but there currently does not exist a central codebase to serve as a repository of institutional knowledge in the field. Likewise, the closed nature of individual codebases severely hampers the repeatability of measurement experiments.

To provide a concrete example of the old paradigm of WPM studies, we describe our earlier study on the mobile browsing ecosystem and the issues that inspired our move to a centralized platform. We present a new measurement platform that solves many of the engineering problems in the literature and has the capability to replicate many of the measurement phases of these studies as well. Finally, we describe a set of new measurement experiments conducted entirely on this platform. We plan to release the platform as an open source tool in order to increase the frequency of WPM studies and discussions in the privacy sphere.

Much of the material from this thesis is drawn from our specific contributions in [8, 9, 28] as well as other currently ongoing studies.

Chapter 2

Background and Related Work

We present a survey of related work with a particular focus on the infrastructure components of previous WPM studies. This survey is primarily drawn from the methodology review in [8]. We also present a brief overview of the types of experiments conducted within the WPM space.

When conducting WPM experiments, the three major tasks are driving a crawl, collecting the data and analyzing the data. The first two tasks are conducted online and ideally should transcend the scope of individual experiments. Data analysis is typically performed offline with custom scripts written for individual studies.

Although some of the studies in the literature appear to have involved manual browsing [30, 16, 3, 14] or crowd-sourcing, conducting repeated experiments across huge numbers of sites requires some form of automation. When considering the type of user agent for a crawl, there exist a few key considerations in terms of the ease of conducting measurements experiment and arguing about the validity of the results.

2.1 Browser Automation

First, the automation platform should be easy to program in order to promote the replication of results and to minimize the possibility of bugs stemming from an overly

complicated architecture. On a related note, the platform should allow researchers to perform a similar set of actions with respect to a real browser. Second, the platform should be resource-efficient in the sense that many instances of the automation platform should be able to run on the same machine. Third, the platform should look like a real user as closely as possible or else an obvious objection to studies performed on the platform is that third parties detected the presence of an artificial user and acted in a pathological manner. We now evaluate the three most fundamental possibilities for user agents in WPM experiments: HTTP libraries (e.g. curl or wget), lightweight browsers like PhantomJS (an implementation of WebKit) and fully automated browser instances powered through drivers such as Selenium.

In practice, none of the studies used HTTP libraries since these libraries clearly do not provide a realistic representation of an actual browser. Lightweight browsers and full browser instances enjoyed heavy popularity in the literature. Acar et. al. [1] claim that PhantomJS can provide up to 200 parallel instances on a commodity desktop, a clear advantage in terms of resource efficiency. However, PhantomJS does not support plugins and, in general, may differ from consumer browsers in a significant enough manner so as to be flagged by bot detection software. Thus, studies performed on this framework risk facing serious methodological concerns as to whether their results reflect the experience of a typical user browsing online. Full browser instances offer the advantage of realism but at the cost of resource efficiency, especially when dealing with issues such as Firefox memory leakage.

In terms of actually automating online browsing, the two primary tasks consist of visiting a page and, if needed, performing a task on that page. The simplest method of visiting a site is to use JavaScript to launch a page in an iframe or a separate tab as did [25] and our earlier WPM study [9] described in Section 3.2. While this approach is extremely simple and platform-agnostic, it does not provide the end-user with much control over the course of the crawl since this involves simply visiting a static page

with a set of JavaScript commands. Perhaps more seriously, these JS-bearing pages can show up as referrers during the course of the crawl, potentially polluting the data.

In order gain more flexibility in browser actions, many researchers used frameworks such as CasperJS for PhantomJS and SlimerJS or Selenium for Firefox, Chrome and PhantomJS. Selenium was popular in many of the studies due to the fact that it implements the Web Driver API for browser automation and provides support for FireFox, Chrome and Internet Explorer. The Ghost Driver integration in PhantomJS also implements this API.[1] However, in our experience and according to researcher responses in [8], Selenium is quite unstable and unreliable for longer crawls since it is subject to frequent crashes and an unreliable blocking API. Likewise, certain desirable features such as headless browsers are supported in PhantomJS but not Selenium.

Studies such as [25, 15, 29] used plug-ins to drive the crawl and/or simultaneously perform measurements. For instance, iMacro [25] provides support for automating complex tasks on a single site.

Overall, we found that Selenium running a full browser instance provides the most realistic browsing experience but PhantomJS and lighter frameworks provide superior stability. Ideally, a web measurement platform will combine full browser automation with the stability of PhatomJS. However, since the studies we considered were primarily one-off measurements, the researchers did not invest the engineering time to build a flexible and stable platform on top of Selenium that also provided support for more advanced features such headless browsing.

2.2 Instrumentation

Instrumentation is often performed through add-ons or plugins. Perhaps the best browser instrumentation tool is FourthParty [23] which has been used in several studies either directly or as a forked version [22, 21, 7, 9, 10]. Several studies used custom

[1] `http://phantomjs.org/release-1.8.html`

plug-ins [17, 26, 27], a useful strategy for crowdsourced studies since they can extract information, visualize results or both. The drawbacks of browser extensions is that they add a time overhead, especially if they run large computations using JavaScript. Similarly, maintaining instrumentation in between browser crashes introduce a new class of engineering pitfalls, as will be discussed in more detail in Section 5.1.2.

Several studies used a proxy instead of or in addition to browser instrumentation. Mitmproxy and Fiddler were the two popular choices, while custom proxies were also seen [13, 25]. Proxies were necessary for data collection for studies lacking browser instrumentation. The authors of [25] also used proxies to modify traffic to perform automation tasks such as modifying DNT headers to measure the impact of Do-Not-Track. While this is certainly one valid approach to this problem, DNT, for instance, can be managed at the browser-level using Selenium, which provides a much more realistic approximation of a real user.

2.3 Measurement Study Targets

The set of studies we examined cover a wide variety of targets which can be partitioned into three categories: data collection, data flow and data use. Data collection encompasses the presence of online tracking, data flow captures the leakage of personal information to third parties and data use entails the presence of content personalization based on user data.

In terms of data collection, the primary focus was the measurement of stateful tracking, namely cookies, either as the primary focus [9, 27, 24, 15, 30, 29, 10] or as a secondary research question [17, 27]. Other data collection studies focused on Flash cookies (LSO's) [24, 30, 3], scripts [15] and stateless tracking such as browser fingerprinting [1].

For data flow, researchers such as [22, 14, 19] examined the flow of identifying information such as names and emails as well as sensitive infomormation such as age and sexual orientation from first parties to third parties. Another source of data flow includes cookie syncing, in which two different third parties pass cookie IDs corresponding to the same user to each other [27]. We will discuss cookie syncing in more detail and present new results in Section 5.3.

Data usage primarily entails leveraging user information when personalizing web pages. One prominent target in this sphere is the presence of personalized ads, including text, flash and image ads. These studies, [4, 12, 34], predominantly focus on Google ads but one study in particular examined a large collection of roughly 175K distinct ads [5].

In a different area, Olejnik et al examined differences in the bid prices used in real-time online auctions [27], which provides a potential window into the valuation of the ad technology industry. However, since this study relied on the fact that the bid prices were sent unencrypted over HTTP traffic, the adoption of encryption may render similar future studies infeasible.

Another application of user data, price discrimination in both online vendors and results returned by search engines were considered by [25] and [26], respectively, while [13, 35, 18] examined the search results themselves for personalization.

The vast majority of the literature examined the data in an offline fashion from databases built over the measurement phases. Of course, data analysis may require non-trivial tasks as heuristically detecting bid prices or cookie ID strings. Hence, building an infrastructure that outputs a comprehensive set of the online interactions, such as the HTTP traffic, cookies and Flash objects in a standardized way would not only enable nearly all of the studies above, but would also provide a single database schema so that research groups, in addition to releasing their own data, could release measurement scripts compatible with other groups' data.

Chapter 3

Measuring Mobile Web Tracking

In Chapter 2, we presented an overview of the various engineering decisions and research questions in the field of WPM. In this chapter, we present the results from our first foray into this space - an examination of the mobile browsing tracking ecosystem - and describe how the challenges in this study inspired the transition to the more unified measurement platform presented in Chapter 4. This chapter draws from our individual contributions in [9].

3.1 Introduction

Despite the wide variety of potential studies, WPM research primarily focuses on desktop browsing, leaving mobile web tracking a largely unstudied area. Conversely, mobile app privacy measurement has attracted a larger degree of attention (e.g. MobileScope[1] and [11], [33]). Given the fact that mobile devices provide additional signals, such as a user's location, which could enhance targeted ads as well as mobile devices' increasing computational power, the lack of mobile tracking research was initially surprising.

[1] `http://mobilescope.net/`

Folk knowledge in the community suggests that this vacuum was because of the relative difficulty of carrying out measurements on smartphones and other devices, given that they are typically locked down in some way and comparatively underpowered. The limited programmability of mobile browsers at the time and physical limitations on RAM and persistent storage further hampered data collection.

3.2 Mobile Measurement Platform

As described in the literature review, the two primary challenges of conducting measurement studies are data collection and browser automation. The primary challenges we solved were the mobile devices' limited storage for crawl data and the difficulty in automating mobile browsers.

In terms of data collection, we ported the FourthParty code base to support Android-based mobile devices, such as smartphones and tablets, by leveraging the Android SDK and the Mozilla Add-On SDK. Although we maintained the same database schema as the original desktop FourthParty, we modified the source code to remove dependency on local secondary storage and push persistence operations through a TCP connection to a desktop server. By porting storage to a desktop platform, we enabled arbitrarily long crawls on mobile devices, subject to the much looser memory constraints on the desktop servers.

When choosing an automation framework, we considered WebDriver, Mozmill and other tools, but ultimately settled on a simple JavaScript architecture. The WebDriver API, implemented by Selenium and GhostDriver, is an extremely popular choice for browser automation in the literature. However, at the time of the study, Selenium did not seem stable enough to support long studies with mobile Firefox (Fennec). Other testing and automation frameworks we considered, including Mozmill, Robocop and Scriptish, faced similar issues when interacting when Fennec.

For the mobile measurement study, we simply created a website containing a list of URLs and a JavaScript snippet that caused a browsing device to visit each one in succession in a separate tab. This approach's primary advantage is its simplicity and platform-agnosticism but this method of driving browsers is otherwise quite limited. In particular, the list of websites on the page must be pre-specified and we cannot easily isolate visits to different websites within the same crawl. On the other hand, non-isolation of websites is also an advantage: we can observe how third parties interact with a user over repeated visits. Indeed, this allowed us to identify and study "growing cookies" as will be discussed in the results section.

3.3 Experiments and Results

We perfomed crawls from the Alexa Top 500 United States sites on 6 devices: a desktop (Ubuntu 12.04, Firefox 11.0), two tablets (Asus Transformer Pad TF300T and Samsung Galaxy Tab 2), a smartphone (HTC Evo 4G), an emulated tablet (Emulated Nexus 7) and an emulated smartphone (Emulated Nexus S). In general, we found that the third-party tracking ecosystems on desktop and mobile devices were extremely similar. In particular, roughly the same set of third-party domains appeared across all crawls and approximately the same proportion of these third parties placed cookies and/or made JavaScript calls typically associated with tracking (e.g. `Window.LocalStorage.*`).

Although the results were for the most part fairly mundane, we found two surprising trends: the nearly complete lack of mobile-only trackers and the presence of growing cookies that appear to store an approximation of a user's browsing history.

3.3.1 Mobile-Only Third Parties

The sheer ubiquity and variety of mobile devices appear to provide a significant opportunity for exclusively mobile-oriented third parties to enter the advertising and analytics space. We define a domain that appeared in our HTTP logs but not in the list of sites we visited to be a third party. We call a third party mobile-only if it is one of the top 100 third-party sites in terms of the number of distinct cookies and tracking JavaScript calls it made for any of the mobile devices but not in the top 500 for desktop. The rationale is that if a tracker, for instance, appears as the top 100th track mobile device and the 200th on desktop, the difference could be explained away by statistical noise.

Contrary to our intuition, we found that each physical and emulated mobile device contained only a handful of top 100 third-party sites that did not also appear in the desktop top 500. These sites were typically the mobile versions of third-party sites present on the desktop. The two exceptions were `admarvel.com` and `mocean.mobi`, advertising networks centered around mobile devices.

The dearth of third parties that exclusively focus on mobile devices is surprising. Perhaps already-established third parties have transitioned to mobile tracking or new third parties have simply not yet entered this relatively new market. Regardless of the reason, our metric for detecting mobile-only third parties can be utilized for keeping track of the growth of this market in the future.

3.3.2 Growing Cookies

In the vast majority of cases, a cookie's value field consists of a string or integer that remained largely static throughout the duration of a particular crawl. However in certain cases, we observed that the values for third-party cookies consistently increased in size throughout a crawl. While some of these growing values are Base64 encoded

strings whose decoded binary values do not present an obvious pattern over time, other values grow in a very specific manner.

In particular, these values appear to be lists related to a user's browsing history. Figure 3.1 presents a brief analysis of a representative example. Each node of the list is delineated by separator tokens and is comprised of two values separated by another type of token. It appears that the first value contained in each node is either an ID corresponding to each visited site, or an ID describing the category of the site (essentially, an interest segment in advertising terminology).

We hypothesize that the second value in each node is a site-specific ID for the individual user. While the latter value does not have enough entropy to be a unique user ID (i.e., unique among all the users that the third party has encountered), it has enough bits to be unique when combined with the approximate creation time of the cookie.

We base our claims on two key observations. First, the growth of these cookies is characterized by the addition of new nodes somewhere in the string each time a site is visited. Second, after repeatedly running new crawls through sites known to place one type of the aforementioned cookies, the nodes corresponding to a particular site were added in the order in which we visited them.

The first node value, which is almost certainly a site or segment ID, remained constant across the different crawls. For some growing cookies these IDs had a few duplicates (different first-parties with the same ID), which rules out site-specific IDs and suggests segment IDs. For other growing cookies we did not observe duplicates, although this does not rule out the possibility of duplicates if we observed more first parties beyond the top 500. The second value in each node, which we believe to be the user ID, changed across crawls in accordance with our hypothesis.

Nevertheless, new nodes were not added to history lists in a particular order, and sometimes the entire list could be shuffled between changes. The most likely

Cookie Value String	Domain
EXAMPLE 1	
"b!!!!#!!**2-]**!!!!#>+YEL"	Netflix
"b!!!!$!!**2-]**!!!!#>+YEL!%**HWu**!!!!#>+YEI"	Cracked
"b!!!!%!!**2-]**!!!!#>+YEL!%**HWu**!!!!#>+YEI!%**ODP**!!!!#>+YF$"	Salon
EXAMPLE 2	
"b!!!!#!!**2-]**!!!!#>+YB5"	Netflix
"b!!!!$!!**2-]**!!!!#>+YB5!%**HWu**!!!!#>+YBo"	Cracked
"b!!!!%!!**2-]**!!!!#>+YB5!%**HWu**!!!!#>+YBo!%**ODP**!!!!#>+YC."	Salon
EXAMPLE 3	
"b!!!!#!%**HWu**!!!!#>+YG<"	Cracked
"b!!!!$!%**HWu**!!!!#>+YG<!%**ODP**!!!!#>+YGC"	Salon
"b!!!!%!!**2-]**!!!!#>+YGP!%**HWu**!!!!#>+YG<!%**ODP**!!!!#>+YGC"	Netflix
EXAMPLE 4	
"b!!!!#!%**ODP**!!!!#>+YLs"	Salon
"b!!!!$!!**2-]**!!!!#>+YM$!%**ODP**!!!!#>+YLs"	Netflix
"b!!!!%!!**2-]**!!!!#>+YM$!%**HWu**!!!!#>+YM.!%**ODP**!!!!#>+YLs"	Cracked
Bold text indicates site ID; Underlined text indicates user ID Hwu = cracked.com; 2-] = netflix.com; ODP = salon.com	

Figure 3.1: This figure contains the cookie value growths for `yieldmanager.com`'s history-storing `bh` cookies as recorded after visiting three sites known to place these cookies in different orders across four different crawls. Individual website nodes appear to be separated by the token `%!` while the site/segment ID and user ID are separated by `!!!!#>+`. Observe that the nodes containing site/segment IDs are added in the order in which the corresponding sites are visited. The associated strings that we believe to represent user IDs for each site remain constant within each crawl but vary across crawls. This is especially apparent when comparing Examples 1 and 2. Note that the nodes for each website appear in the same relative ordering (specifically `netflix.com` precedes `cracked.com` precedes `salon.com`). This particular ordering suggests that the third party represents the history nodes in a JavaScript associative array, accounting for the fact that the nodes consistently appear in a specific sorted order that is independent of the order in which they were visited.

explanation is that the histories are stored in a JavaScript (associative) array before being serialized into a cookie. Associative arrays do not preserve the order of insertion.

Observe that these history-storing cookies expose a privacy vulnerability. Suppose an attacker uses cross-site scripting to read the contents of one of these growing cookies. Then, he could a lookup table for the cookie's site ID's to recover (an approximation to) the victim's browsing history. An even weaker adversary could simply listen in on the communication between the user and website to intercept the contents of these cookies.

Having established the existence of growing cookies, for each device we then examined the proportion of third-party sites that place growing cookies during the crawls. The results are contained in Table 3.1.

Formally, we consider a cookie to be a growing cookie if it had been changed at least five times and if it satisfies a certain growth metric. We consider the standard growth metric, which we denote `X3`, to be an indicator of whether over the course of the crawl, the cookie's value tripled in length and had a final length of at least 25 characters. We imposed the limit of 25 characters so as to not count the common case of cookies that are added with either no value or a single-character value that is immediately changed to a short but static string. Clearly these types of cookies more than triple in size but do not grow beyond this initialization. As a stricter requirement for growth, which we denote `Strict`, we consider the subset of cookies that pass the `X3` requirements but additionally have at least 75% of their changes increasing their lengths and no more than 10% of changes decreasing their lengths.

Device	**Growing Metric**	
	X3	**X3 w/ Strict**
Desktop	8.0%	3.6%
Tablet (Asus)	3.2%	0.8%
Tablet (Galaxy)	4.4%	2.6%
Phone	1.0%	0.4%
Emulated Tablet	0.4%	0.2%
Emulated Phone	0.8%	0.6%

Table 3.1: Percentage of third parties using growing cookies

We find that regardless of the metric, a much larger proportion of top third-party sites place growing cookies on the desktop when compared to the mobile devices, with perhaps the slight exception of the Galaxy tablet. Both the emulated phone and physical phone have extremely low proportions of growing cookies.

There are two possible reasons why the desktop has a greater proportion of growing cookies. First, the relative immaturity of mobile tracking when compared to desktop tracking might mean that trackers have not yet fully ported their growing

cookie infrastructure to mobile devices. Second, the limited computational power of mobile devices might mean that companies prefer server-side tracking architectures. Developing automatic methods to parse and determine the structure of these growing cookies is one potential avenue of future work.

3.4 Returning to Desktop

Overall, the most interesting result from our measurement of mobile tracking was the presence of growing cookies, which themselves were most prevalent on desktop platforms. Furthermore, the lack of stable mobile browsing automation frameworks severely limited the ability to conduct large-scale and in-depth mobile measurement studies. For instance, one interesting line of inquiry would be to examine the behavior of trackers when visiting sites logged into social media accounts or after clicking on links within a site - tasks not generalizable using our naive JavaScript automation approach. From a data collection perspective, the ability to easily port FourthParty *instrumentation* to various devices for our own experiments further stood to highlight the lack of a similarly generalizable browser *automation* framework for mobile studies.

Given the presence of the several desktop automation frameworks that we examined when attempting to automate our mobile study, we were surprised to find that no single unifying infrastructure existed in the literature that natively implemented more complex actions when visiting sites as would have been desirable for our mobile studies. This infrastructure vacuum led to the development of our own generalized platform that will be described in the next chapter.

Chapter 4

Engineering the Web Measurement Platform

4.1 Requirements

By examining the challenges faced in other experiments, we determined that our WPM platform should be flexible in its ability to manage a wide variety of large-scale, automated studies. This core vision translated into a few key design requirements.

WPM experiments consist of three major tasks. The first task is mapping high-level commands, such as visiting URLs or extracting news articles from websites, into automated browser actions. The second task is collecting and consolidating crawl data, such as cookies set by the browser and JavaScript calls, in a unified manner. The last task is to create automated tools to perform specific analyses on the dataset in order to answer individual research questions.

While the third component is specific to individual research questions, the browser automation and measurement components are designed to be general-purpose by adhering to the following principles.

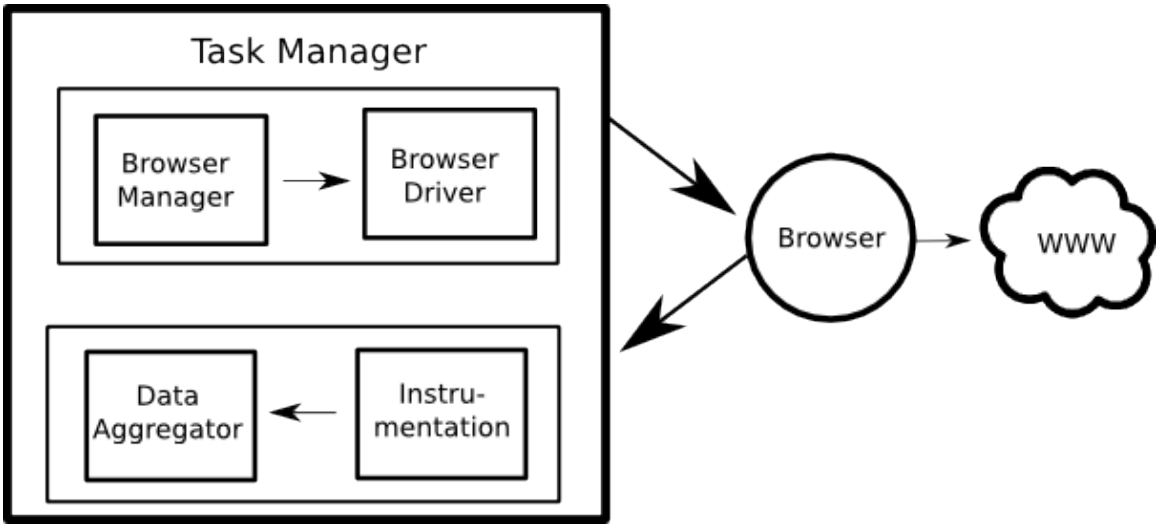

Figure 4.1: Generalized WPM framework
The Task Manager ensures crawls are robust, the Browser Manager converts high-level commands into automated browser actions and the data aggregator robustly receives and pre-processes data from browser instrumentation tools.

Stability. During the course of a long crawl, a variety of unpredictable events, such as page timeouts or browser crashes, could halt the crawl's progress or, even worse, corrupt the data. The browser automation framework should recover from such events gracefully, quickly and intact.

Abstraction layer. The overall automation API should serve as a user-friendly abstraction, hiding the complexities involved in performing browser tasks such as finding and clicking buttons on a page. Keeping automation commands at a high-level reduces the complexity of the scripts used for driving particular experiments.

Modularity. In the course of actually implementing our framework, we had to choose specific libraries for tasks such as driving the browser. However, adopting a modular design will enable us to easily switch the underlying tools in our framework for future studies.

Realism. Our automation framework should mimic a real person surfing the web as closely as possible - a task we primarily complete by choosing to drive full browser instances.

4.2 Design and Implementation

A high-level diagram of our infrastructure is contained in Figure 4.1.

We divided our browser automation and data collection infrastructure into three main modules: a Browser Manager which acts as an abstraction layer for browser automation, a Task Manager and a Data Aggregator which acts as an abstraction layer for browser instrumentation. The entire infrastructure was implemented using Python.

The motivation for requiring a Browser Manager as an abstraction layer stems from our choice to use Selenium for automation. As we and several other groups of researchers found, despite its other advantages, Selenium is quite poorly suited to running scalable automated crawls across websites — it is intended as a tool for webmasters to test their own sites, where they are in control of the site architecture. As a result, it frequently crashes or hangs indefinitely due to its blocking API [2]; in our survey of engineering challenges we found that there are no solutions to this problem, only workarounds.

We opted to use Selenium, as opposed to other browser automation frameworks such as CasperJS and SlimerJS, due to its more extensive range of web technologies, such as plugins, addons and HTML5 features. Due to the need for scaling we used pyvirtual display to interface with Xvfb in order to run headless browser instances, which both allowed us to utilize remote virtual instances for crawls and increased the number of parallel crawls that we could run on a given machine.

As an example workflow, the Browser Manager receives a command to extract information from a given news site. The manager translates it into a series of low-level commands to instruct the browser driver to visit the site, wait for the page to load, and take various actions on the page. Browser instrumentation tools parse the page content and sends the relevant data (in this case, headlines) to the Data Aggregator. The logic for translating application-specific commands into browser actions resides

entirely in the Browser Manager, enhancing modularity — we can swap out the browser and driver if necessary, without affecting the rest of the platform.

The Task Manager's chief responsibility is to recover gracefully from inevitable crashes or freezes in any of the other components without affecting the crawl. It is also responsible for management of user profiles. While the Task Manager does not conduct any of the browser automation itself, it is responsible for copying over user information (e.g. cookie databases) and preferences (e.g. whether features such as DoNotTrack are enabled) to newly-spawned browser managers. Essentially, while individual browser instances may stop and restart, the Task Manager ensures that the crawl as a whole appears to be from the same person's browser.

In terms of collecting data during the crawl, we experimented with different instrumentation options, including FourthParty and mitmproxy. While browser automation and instrumentation initially appear to be separate issues, in practice, one of these two components crashing can fatally disrupt the other. Since instrumentation tools often write data as well as collect it, browser crashes (and the corresponding instrumentation failures) can severely corrupt data.

To ensure data integrity, the Task Manager also manages the Data Aggregator subprocess, which receives data during the course of the crawl, manipulates it as necessary and writes it to a central database. A socket-based communication framework enables the Data Aggregator to separate itself from browser crashes and continue writing to the database, eventually receiving the input from a restarted browser instance without having to restart itself.

Also, since the data aggregator can receive raw data, it is able to perform complex data pre-processing tasks in real-time before writing to the database. This allows the browser instrumentation process to spend all of its computational power on parsing and manipulating pages. We have found that doing pre-processing in the same process as instrumentation can result in performance degredation.

While we have built a platform that allows us to quickly launch new WPM experiments (we present such three such studies in the next chapter), there are many possible improvements. For example, it would be useful to simulate complex user interactions on websites, such as automatically logging into websites with a given username and password or finding and clicking "Like buttons." Collaboration with researchers on such tasks as well as new WPM studies will significantly speed up the development process and reduce redundant engineering work. Accordingly, we plan to open-source our platform at the earliest opportunity.

4.3 Evaluation

Our infrastructure is still accumulating changes and improvements, so our evaluation presented here reflects the state of the platform at the time this thesis was written.

Stability. When running a light wrapper around Selenium, without our Browser Manager and Task Manager, the stability varied depending on experimental conditions but was always poor. With a timeout of 40 seconds, the best average we were able to obtain was 800 pages without a freeze or crash. This number is too small for large-scale measurement studies. We also frequently observed data corruption. With our current infrastructure we successfully recover from all crashes and have observed no data corruption after completing crawls of over 100,000 pages.

Resource usage. When using the headless configuration, we are able to run up to 24 browser instances on a 16GB commodity desktop. This is in line with the other studies we analyzed and is limited by Firefox's memory leak issues. The platform also possess support for a single Task Manager controlling multiple browser instances at once. This new multi-process module allows the end-user to issue commands to these browsers in different ways, including all at once and for-come-first-serve.

Generality. For the news measurement study in the next chapter, the browsing and measurement was completely automated with the exception of writing custom template scripts for extracting headlines. We hope to eventually automate the task of extracting categorized lists of links from arbitrary sites. Generality also applies to the fact that our infrastructure now log many storage vectors such a HTTP cookies, requests and responses and Flash objects, which enables a much richer set of studies than for crawls instrumented with FourthParty alone.

Modularity. Our platform abstracts away the details of the automation framework and trivially supports other browser/driver configurations, although we have only done large crawls with Firefox/Selenium. We support different measurement options (instrumentation via FourthParty, or proxying via mitmproxy) and have successfully tested both in isolation and in combination.

Chapter 5

Selected Applications

We now present three separate studies conducted using the measurement platform described in the previous chapter. The first study on news personalization was performed as part of [8]. We briefly present the results of this paper but our individual contributions on this study were primarily on infrastructure matters that will be described after the results. Similarly, we performed more of an infrastructure support role in the identification leakage through cookies study [28]. We conducted all of the analysis presented in the section on cookie synchronization, which is part of an ongoing study.

5.1 Measuring News Personalization

5.1.1 Motivation and Results

Although the threat of the filter bubble - the use of personalization algorithms to steer a user into seeing only content related to his or her interests - is a concern in the media, the actual level of personalization is predominately collected through anecdotal evidence or manual analysis [31]. We considered the level of personalization on news publishers by examining links to two types of recommended content: "Around

the Web" (i.e. links outside a given publisher) and "More From [Publisher]" (links owned by that publisher). These sets of related links for the publishers we considered were contained in recommendation boxes run by Taboola[1] and Outbrain[2], two large content recommendation and delivery engines which are found on many mainstream news publishers.

We created extreme profiles of users who browse a single topic, such as technology, which were trained over a period of 4 days. Classifying articles into topics leveraged the categories contained within the URL strings of the publishers themselves. During the measurement phase, we re-loaded the browser profiles built during the training period and re-collected the links, examining the categories of sites contained within the recommendation boxes.

Overall, we found that Outbrain's content, but not the Taboola's recommendations, contained a statistically-significant level of personalization. Links in "Around the Web" were skewed towards articles matching the interests of the training profile and links in "More From [Publisher]" were skewed away from these interests. However, qualitatively, these skews towards or away from the interest area were quite low - always less than 10% but often even below 3%.

Despite some limitations of the study including the short training phase and the focus on history-based personalization rather than on location-based and other types of personalization, the filter bubble appears less present than conventional wisdom would suggest. In this case, the high level of media coverage did not correspond to a high level observed personalization. In the next case study, we present a more subtle tracking issue that, despite its relative lack of media coverage, could represent a much greater privacy violation.

[1] `http://www.taboola.com`
[2] `http://www.outbrain.com`

5.1.2 Infrastructure Contributions

The three primary engineering challenges when running the news crawls were profile management, the addition of study-specific HTML-parsing and measurement code within the platform and the necessity of running many browser instances with limited computational and memory constraints.

With respect to profile management, freezes or crashes in the browser or Web Driver necessitate restarting Selenium with a fresh browser instance. However, unless the full set of cookies and other profile attributes are transferred to this new browser instance, it may appear to represent a new user in the eyes of third parties. To alleviate this concern, we added profile management so that, when the Task Manager detects a crash or freeze, it dumps the browser profile as well as preferences such as Do Not Track into a tar file.

During the profile recovery process, the Task Manager uses this tar file to cleanly reload the browser profile. Furthermore, we added high-level functions to dump these profile files at any time and load them at the beginning of a crawl. Overall, the platform natively supports profile management that hides the implementation details from the user and providing a clean interface capable of managing the dozens of distinct browser profile files necessary for the news experiments.

Although the other authors actually implemented the custom HTML-parsing code for the study, our modular design enabled this new library to be incorporated into the platform as a new high-level command in under a dozen lines of code. The successful integration of these news measurement scripts illustrates that the platform can support libraries from many different studies without bloating the core code base.

The modular architecture, especially decoupling the Task Manager from direct interactions with the browser, also enabled the platform to be easily ported to a multi-process framework in which a single Task Manager manages several browser

instances at once. As a result, we could manage the training of many news profiles without the resource overhead of many Task Manager instances.

5.2 Linking IDs through Cookies

5.2.1 Motivation and Results

In [28] we considered the problem of a network observer using third-party cookies to link an individual's unencrypted web traffic together. This attack is considerably strengthened given the threat of personally identifying information leaked by a third party in turn being linked to previously pseudonymous web traffic.

Using a set of heuristics, of which we present an expanded version in our description of the third case study, we identified third-party cookies with value strings that appear to be unique IDs. When examining HTTP traffic, we were able to connect visits to two different sites together if a common third party ID appeared while browsing both these sites. Overall, this resulting traffic graph was highly connected. When examining crawl data emulating the histories found in the AOL dataset as per the method in [17], we found that 90% of sites with embedded trackers can be linked in a single connected component.

After creating and logging into accounts for the top 50 sites (according to Alexa) that support account creation, we found that in the course of browsing, over half of these sites leak personal identifying information over insecure HTTP traffic. In particular, 28% of sites leaked a user's first name, 12% leaked the full name, 30% leaked the user name, 18% leaked the email address and 60% leaked at least one of these attributes. The fact that some of these sites also were in the giant connected component of the traffic graph is troubling because not only can this traffic be connected to a single pseudonymous user but this user can also be linked to a real-world identity.

The adoption of extensions such as "HTTPS everywhere" which makes requests over HTTPS whenever the server supports it or third-party cookie blocking may help to mitigate this attack and is a potential direction of future work. Overall, this study illustrates the privacy risks stemming from third party cookies in which traffic linking is managed by some external adversary using third-party IDs. In the third case study, we will examine the implications of ID synchronization performed by the third parties themselves.

5.2.2 Infrastructure Contributions

While concerns with browser automation dominated the engineering challenges for the news personalization study, the cookie ID study required a greater concern with respect to instrumentation. In particular, we originally opted to use FourthParty to log cookie interactions and HTTP traffic. However, when dealing with browser crashes, simply copying the old FourthParty database into the new browser profile often led to data corruption over the course of long crawls. The two workarounds were to programmatically move over the rows in the old database to the new one - an extremely inefficient approach - or, instead, dumping all of the distinct FourthParty databases into a single file and then merging them after the crawl.

Neither one of these approaches was appropriate for a generalizable infrastructure since they either required a high computational overhead or forced the end user to perform database post-processing. Furthermore, FourthParty's extensive use of JavaScript adds computational overhead to browser instances. To reduce these computational requirements and to increase usability, we used mitmproxy to capture HTTP traffic, including the flow of cookies, and pass the data to the Data Aggregator process. Since the Data Aggregator remains functional during the course of the crawl and maintains the database connection itself, the end user only has to interact with a single database per crawl, regardless of the number of browser restarts.

Additionally, the platform now supports Flash storage logging and more sophisticated cookie logging, thereby increasing the amount of data collected during the course of the crawl beyond FourthParty's standard offering. We hope that the platform's expanded database will increase the variety of questions that can be natively answered with our platform's output. If the platform is adopted by other researchers, then it will provide the ability for research groups to verify the results of other studies on their own databases without having to add additional instrumentation. Overall, just as the platform hides the browser automation details from the user, the increased data logging should also reduce the necessity of adding custom instrumentation.

5.3 Measuring Cookie Synchronization

A target of prior work such as [27], cookie synchronization occurs when third parties send each other the value strings corresponding to ID cookies - a practice that enables back-end database merges which link records from distinct psuedonymous IDs together. Often, these syncs are performed through HTTP redirects or direct HTTP requests. For instance, `tracker_a.com` could pass its ID string `xsq21` to `tracker_b.com` by making a request of the form `tracker_b.com/sync/partner=a&uid=xsq21`.

For this section of the paper[3], we will consider a basic threat model for cookie syncing - namely the ability for third parties to use these IDs to perform back-end merges. As the previous two case studies focused on the automation and instrumentation component of the WPM platform, this section will focus on a useful subroutine for WPM studies, which we first introduced in [28]: a more accurate automatic method of detecting ID cookies.

[3]This section is based on work in progress with Gunes Acar, Claudia Diaz, Steve Englehardt, Mark Juarez and Arvind Narayanan.

5.3.1 Detecting unique identifier cookies

Cookie text files contain several pieces of information, including the host (the domain who owns it) and a name-value string. For our analysis, we consider a *cookie* to be the unique (owner's domain, name) pair and the cookie's *value* to be the value component of the name-value string (see the Cookie text in the PREF cookie below). Our most fundamental analytical task was to effectively identify cookies with value strings that correspond to unique identifiers.

```
Host :  www.google.com

User-Agent :  Mozilla/5.0 (X11; Ubuntu;

Linux x86_64; rv:26.0) Gecko/20100101

Firefox/26.0

...

Referer :  http://www.unity3d.com/gallery

Cookie :  PREFID=5834573d6649ab5
```

For a cookie to be useful as an identifier, cookie values must have two important properties: persistence over time and uniqueness across different browser instances. Based on these criteria we develop heuristics that classify cookies as identifiers and attempt to avoid false positives. Our heuristics are intentionally conservative, since false positives risk exaggerating presence of cookie syncing. Our method does have some false negatives, but this is acceptable since our study will at least demonstrate a lower bound on the prevalence of this practice.

We define a cookie to be an identifier cookie if its value string:

- **Is long-lived** – the cookie's observed expiry date at time of creation is sufficiently far in the future

- **Remains stable over time within a browser instance** – once set, the cookie value does not change over time and over multiple visits to the cookie domain and even if the browser is restarted
- **Varies across browser instances** – has a different value in all the crawls we have observed
- **Passes the entropy test** – value strings are sufficiently different across different browser instances so as to serve a globally unique identifier
- **Is of constant length** – the length of the value string is invariant across browser instances

Long-lived cookies are non-session cookies with expiry times longer than three months. Our baseline ignores cookies that fail this criterion as they are too transient to track long-term user behavior without back-end synchronization.

Stable cookies have value strings that, for a given browser, remain stable across multiple browsing sessions. Cookies with dynamic value strings may simply be logging timestamps and other non-identifying information. We believe that other non-static cookies contain identifying information but do not fit within our study. For example, (google.com, __utma) changes when data are sent to Google Analytics.[4] Although we believe that cookies encoding some part of an individual's browsing patterns may indeed leak personal information, their dynamic nature excludes them for our definition of persistent identifier.

User-specific cookies have value strings that are unique across different browser instances in our dataset. Cookies that fail this criterion only mark a given browser as belonging to larger set of browsers that have been marked with a given string. An extreme case is (doubleclick.com, test_cookie), which has the same value (CheckForPermission) across all crawls in our sample.

[4] `https://developers.google.com/analytics/devguides/collection/analyticsjs/cookie-usage`

Low-entropy cookies have values that differ across different users' browsers in our measurements but are not sufficiently different enough as to be truly unique identifiers. For instance, value strings that incorporate fixed timestamps (e.g. the time that the cookie was created) may differ across different users by a few digits but are likely to not be globally unique. To mitigate this issue, we used the Ratcliff-Obershelp [6] algorithm to compute similarity scores between value strings. We filtered out all cookies with value strings that were more than 90% similar to value strings from the corresponding cookies from different crawls.

Constant-length cookies have value strings with the same, fixed length across all our datasets. We believe that unique cookie identifier strings are typically generated in a standard, fixed-length format. This belief is motivated by patterns seen during manual inspection and the likelihood of third-party libraries generating identifiers. As such, we only consider constant length cookies, keeping in mind that this heuristic may cause false negatives that render our analysis in fact overly conservative.

To collect data to identify unique cookies, we ran two simultaneous crawls using identical sets of websites visited in the same order. By conducting simultaneous 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.

Once the data were collected, we applied our heuristics when comparing cookies with the same keys across the different datasets in order to identify the cookies most likely to be unique identifiers. To increase the number of potential ID strings, we also attempted to parse the cookie value strings into parameter-value pairs according to known delimiters.

5.3.2 Detecting cookie flows

Using the heuristics outlined in the previous subsection, we were able to identify cookies with value strings that are or contain user identifiers. Given these IDs, one of the most fundamental questions of cookie syncing is to understand to what extent these identifiers flow to multiple domains. In particular, for a given ID we consider which domains know this identifier by examining cookie values and HTTP traffic.

If a domain places a cookie containing a given ID, then the domain clearly knows that ID. In fact, a telltale sign of cookie syncing is multiple domains owning cookies with the same ID value string. For HTTP traffic, if an ID appears in a domain's URL string of any time (e.g. the referrer URL), we assume that domain knows the ID.

For both HTTP requests and responses, we assume that if an ID appears in the referrer URL, then the domain owning the requested URL will learn this ID. Unfortunately, we cannot assume the reverse as third-party JavaScript executed on a first party domain will cause the first party to show up in the database as the referrer in an ID sync call even though this first party may not even be aware an ID sync is taking place. When examining HTTP redirects in the HTTP response database, we consider the URL of the location domain in the redirect. In particular, if an ID appears in the location URL then we assume the domain performing the redirect must have known the ID in order to perform the redirect.

Determining the directionality of ID syncs is a much more difficult task since, as previously mentioned, the observed referrer may not be the party performing a sync. We can typically determine the directionality of flow in HTTP redirects but overall the flows in which we could to determine both the sender and receiver is a very small fraction of all observed ID syncs. Hence, when examining cookie synchronization, we focused on which parties knew a given an ID rather than the precise paths in which the ID flowed.

5.3.3 Basic results

In order to collect the data for the sync measurements, we ran multiple crawls of the Alexa top 3,000 on Amazon EC2 instances. As the general statistics for cookie synchronization were roughly equal across the crawls, we present the data from one crawl as an illustrative example.

We detected 569 distinct ID strings dispersed across 614 distinct cookies. The fact that these IDs appeared in a greater number of cookies can partially be explained by the fact that some domains owned multiple cookies with the same ID. However, the presence of independent domains placing cookies with the same IDs also points towards cookie synchronization.

The ratio of distinct cookies to distinct IDs increases when considering the number of IDs observed being synced. We claim that an ID is involved in a sync if at least two distinct domains know it. Under these conditions, we detect 138 distinct ID strings spread across 179 distinct cookies. There were a total of 384 domains which knew at least one synced ID. The most widely-spread ID involved in synchronization was known by 45 different sites and each ID known by 5.88 domains on average, with a median of 3.

The top 20 ID collectors are contained in Table 5.1. Overall, each domain knew an average of 2.11 IDs, with a median of 1 ID. Clearly, these top domain collectors directly know the IDs of several other domains, but the typical domain involved in syncing only knows a handful of IDs. In the next subsection, we demonstrate that given the wide presence of certain IDs as well as the fact that certain domains know many IDs, even parties who do not know many IDs themselves can merge their tracking databases with many other domains.

Domain	Number of IDs known
doubleclick.net	27
rubiconproject.com	24
openx.net	23
adnxs.com	22
pubmatic.com	19
bluekai.com	17
facebook.com	17
contextweb.com	14
exelator.com	11
casalmedia.com	11
rlcdn.com	11
lijit.com	10
aol.com	9
addthis.com	9
demdex.net	9
adscale.de	9
360yield.com	8
burstnet.com	9
nexac.com	6
afy11.net	6

Table 5.1: Top-20 observed ID-syncing domains

5.3.4 Back-end database synchronization

In the process of tracking a user, a third-party domain can map that user's browsing history to a pseudonymous ID. Via cookie synchronization, a tracker can associate that user's records with other domains' IDs, even without knowing who originated a given ID. Given the long and random nature of these ID strings, trackers can use commonly-known IDs to merge their records on specific users.

More precisely, we consider two forms of database merges. In the *one-hop* model, two domains can merges their records for a user if they mutually know at least one ID. In the *two-hop* model, two parties can also use an intermediary to perform the sync if they each know at least one ID in common with this intermediary (but not necessarily with each other). This model captures a greater degree of tracker cooperation, including the presence of domains that serve as ID exchange hubs.

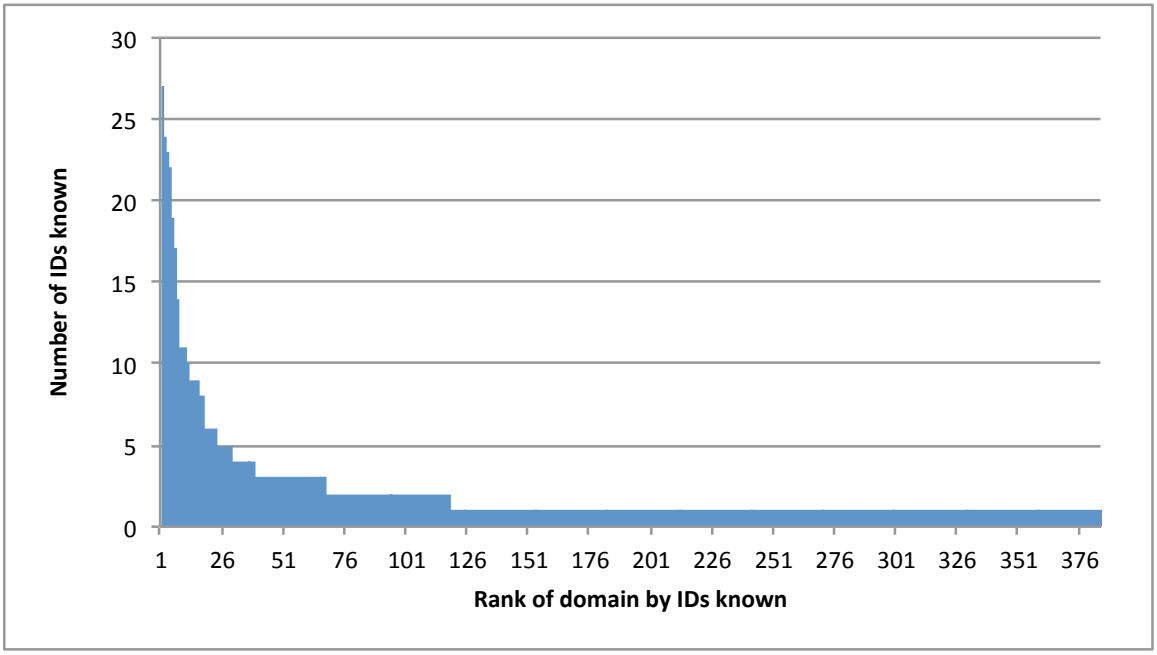

Figure 5.1: Number of IDs known by domains involved in syncs

Figure 5.1 contains the number of IDs known by each domain in descending order. Observe that this distribution contains a long tail of domains that only know a few IDs. This long tail persists when, sorting by the number of known IDs, considering the number of other domains with which a given party can perform a direct database merge (see Figure 5.2). Each domain can sync with 19.18 domains on average, with a median of 10. For the top 50 ID collectors, we observed an average and mean of 68.18 and 61.5, respectively. While the one-hop model enables the top ID collectors to merge databases with 20% of other domains observed performing syncs, only allowing direct database merges hampers the ability for domains which only know a few number of IDs to expand their records.

As illustrated in Figure 5.3, introducing intermediaries to the database merge model significantly increases the merging capacity for many of the domains. In particular, the average number of other sites a given domain can sync is now 121.44 while the median is 156. For the top 50 domains, the mean is 216.36 and the median is 232, roughly 57% of all domains observed in syncs.

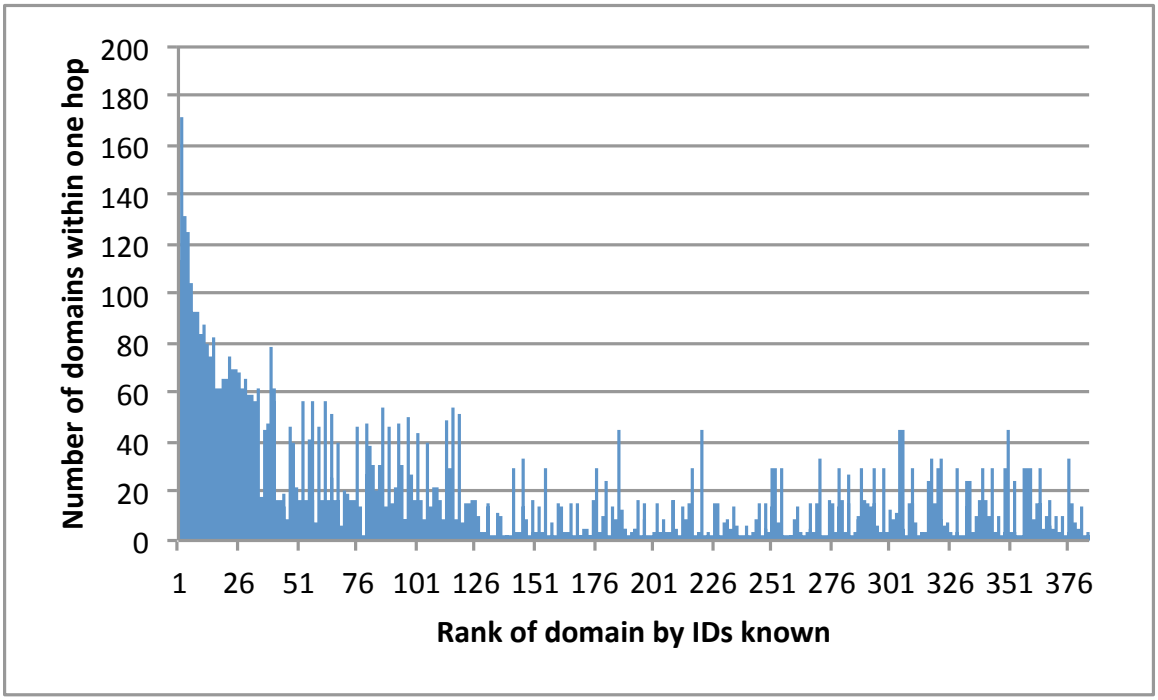

Figure 5.2: Back-end database merge capabilities with single hop

More significantly, many of the domains with only a few IDs can leverage the presence of ubiquitous IDs and intermediate domains with many IDs to communicate with a sizable fraction of parties involved in synchronization. Overall, using intermediaries in merges enables domains to connect to a large fraction of known cookie syncing parties using only one or two hops. In the next subsection, we consider the potential for cookie synchronization when tracking a user who clears his cookies.

5.3.5 Cookie re-spawning and syncing

At a given point in time, cookie synchronization provides a mechanism for linking different IDs corresponding to a single user together. These collections of linked IDs can be represented as graphs, which intuitively will be disjoint when a user clears his cookies. However, these graphs can be connected if one of these deleted cookies is re-spawned and later used in database merges. An even stronger threat model would be a cookie re-spawn followed by continued synchronization. Given the many

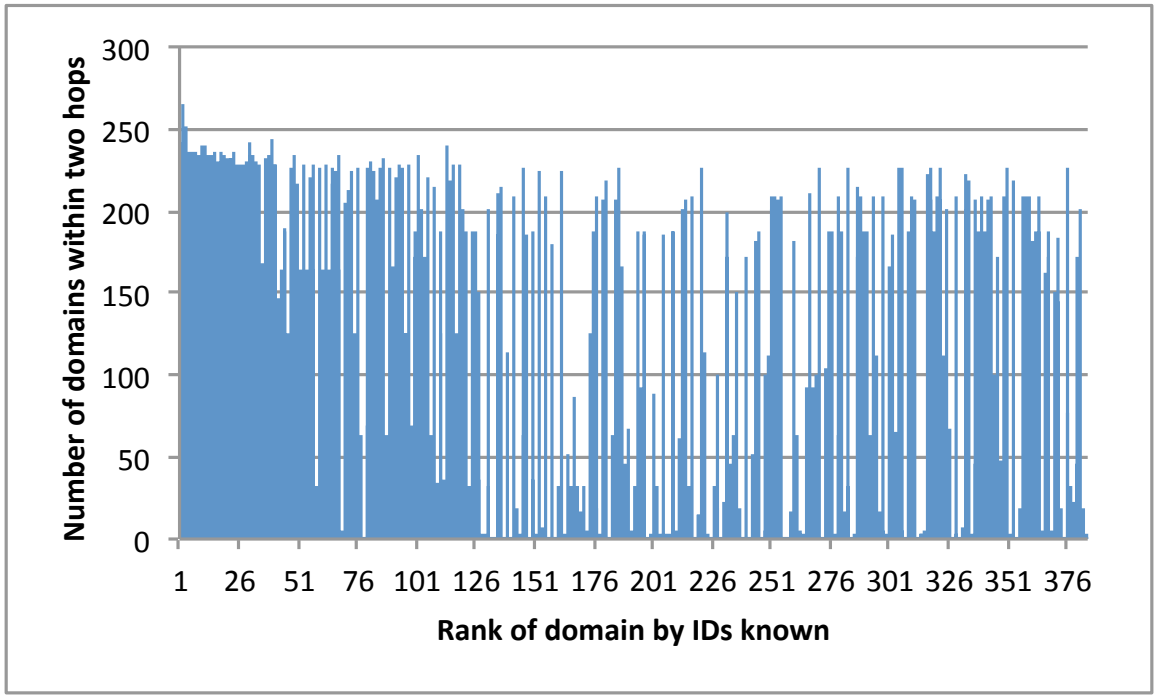

Figure 5.3: Back-end database merge capabilities with two hops

possible storage vectors (e.g. Flash and local storage) which can be used to re-spawn HTTP cookies, re-spawning and re-syncing could be a powerful tool for linking a user's browsing history between state clears.

In order to test this threat, we first ran a 3,000 site crawl on one EC2 instance. On a second instance, we ran a 3,000 site crawl from a completely fresh state. Next, we cleared the Flash storage, cache and local storage on the second machine and copied over the Flash storage from the first machine. Finally, we ran another 3,000 site crawl on the second machine. Both crawls on the second machine were conducted using the same randomly-selected browser fingerprint.

If an HTTP cookie appeared in both the first machine's and second machine's second crawl's cookie database, we claim that this cookie was re-spawned. In total, we detected 3 IDs that were re-spawned with ID strings that also appeared in the Flash storage - indicating that Flash was likely used in the re-spawning process. We only observed one of these IDs being synced.

More unexpectedly, we discovered 12 IDs that were re-spawned between the first and second crawl on the second machine that did not appear in Flash or local storage. While perhaps encoded versions of these strings appeared in one of these storage vectors, we conjecture that these cookies were re-spawned through some form of fingerprinting. Moreover, two of these IDs were observed in sync flows.

One of these IDs, `de7fbd21d41af681013b188b0459c76a`, provides a useful case study. In particular, `casino.com` made a request to a `merchenta.com` URL containing this ID as a parameter. Then, `merchenta.com` redirected to a `adnxs.com` URL containing this ID. While we cannot conclusively determine if `casino.com` made the first call itself, `merchenta.com` definitely passed this ID to `adnxs.com`, one of the top observed ID collectors contained in Table 5.1.

To demonstrate the power of this ID being re-synced, suppose that the ID syncing domains collectively attempt to track a user between state wipes. In the pre-wipe crawl, we observed 140 unique IDs and 365 syncing domains. For the post-wipe crawl, the corresponding figures are 133 and 381. Between the two crawls, we observed 271 distinct IDs and 456 syncing domains. Many of these IDs can be divided into pairs: a given ID cookie's value before the state wipe and its value after the second crawl.

Now consider the connected components in the ID sync graph. In particular, there exists an edge between two IDs if they are mutually known by at least one domain. In the first crawl, the connected component containing `de7fbd21d41af681013b188b0459c76a` had 99 IDs, corresponding to 276 domains. For the second crawl, it contained 94 IDs, corresponding to 290 domains. Both of these components correspond to a potential sync network of roughly 70% of all parties observed performing ID syncs during the course of a crawl.

Merging the two graphs through the re-spawned and re-synced ID results in a connected component with 192 IDs, corresponding to 341 domains. Once again, this figure corresponds to approximately 70% of the syncing domains. Overall, this

analysis indicates that even a single re-spawn and re-sync can enable a supermajority of the parties involved in ID synchronization to combine their records across cookie wipes. The fact that we observed this sync occurring even when clearing cookies, Flash objects, local storage and cache indicates that not only can a user's traffic be linked at fixed points in time but that a user making a concerted effort to wipe state can have his future and past browsing linked through the actions of a single party.

Chapter 6

Conclusion

Even as a collection of largely disparate studies, Web Privacy Measurement has offered insight into third-party tracking patterns and, more generally, the collection, flow and use of individuals' data. Although many of these studies included some form of automation in their measurements, we have introduced a more stable, scalable and generalizable web measurement platform, which has already been used in three distinct measurement studies.

Moreover, resolving the various automation issues, such as the problem of profile measurement between browser crashes, suggests not only is our platform much more robust than infrastructures built for one-off studies, but that it also has the potential to serve as a source of institutional knowledge in the web measurement space. Consequently, we will be releasing the platform as an open-source project in order to increase collaboration in the research community, reduce redundant engineering effort and consolidate efforts into a single platform.

We hope that this environment of increased collaboration will foster a greater number of ongoing measurements, in turn, fueling more lively debate in the privacy community and ad technology industry.

6.1 Future Work

Future work in Web Privacy Measurement can be divided into platform improvements and possibilities for future studies. In terms of growing the infrastructure, leveraging machine learning to perform tasks such as automatic button detection would provide methods of interacting with a page beyond merely visiting it. Related subroutines such as the ability to consistently log in to sites with user credentials would further contribute to this deeper level of interaction. These types of additional features can be generally clustered under the umbrella of providing a platform API that natively supports complex forms of page interaction.

Another desirable trait for a measurement platform is its ability to better mimic a real user's browsing patterns. Beyond the obvious desirability of managing the input user credentials such as a name and an email throughout a measurement, automatically generating user browsing patterns would greatly enhance the realism of experiments. In particular, many studies use fixed content such the Alexa top sites to seed their crawls. A superior model would be some sort of generative approach in which a list of sites can be built based on the demographics of the user we are trying to mimic.

When considering new targets for WPM studies, the ability to consistently label top-level domains when logging HTTP traffic and cookie interactions with our platform would enable new sets of experiments. For instance, linking changes in the cookie database with visit to top-level domains will enable further analysis in parsing and understanding the semantics of growing cookies. The presence of top-level domains will similarly enable researchers to examine ad segments present in the HTTP traffic that occurs on individual sites. Analyzing the sets of segments placed on first party sites corresponding to certain interests or demographics may reveal that ad companies can leverage these segments to learn sensitive demographic information.

Mapping first parties to third parties is a particularly for useful for privacy debates since the third-party domains themselves may be largely unknown by the general public. Although people browsing online choose to visit first parties, they are likely not aware of the specific third parties tracking them. However, revealing third-party tracking patterns and linking these parties to first parties will likely cause public debates involving first parties who, in turn, can choose whether or not to allow particular third-party trackers on their sites.

Finally, the tracking landscape will remain dynamic, with various tracking techniques gaining and losing popularity as well as changing the ways in which they are implemented. Performing periodic measurements and performing diffs on the data could reveal not only pattern in the tracking ecosystem at a fixed point but also changes over time. Consolidating these longitudinal results online as part of a web measurement census is one future application of the platform that will provide increased exposure to trackers' behavior and provide a dynamic illustration of the changing tracking landscape.

Bibliography

[1] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. FPDetective: dusting the web for fingerprinters. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1129–1140. ACM, 2013.

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

[3] 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.

[4] 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.

[5] Paul Barford, Igor Canadi, Darja Krushevskaja, Qiang Ma, and S Muthukrishnan. Adscape: Harvesting and analyzing online display ads.

[6] Paul E. Black. Ratcliff/Obershelp pattern recognition. `http://xlinux.nist.gov/dads/HTML/ratcliffObershelp.html`, December 2004.

[7] Abdelberi Chaabane, Yuan Ding, Ratan Dey, Mohamed Ali Kaafar, Keith Ross, et al. A closer look at third-party osn applications: Are they leaking your personal information? In *Passive and Active Measurement conference*, 2014.

[8] Steven Englehardt, Christian Eubank, Peter Zimmerman, Dillon Reisman, and Arvind Narayanan. Web privacy measurement: Scientific principles, engineering platform, and new results. Manuscript, 2014.

[9] Christian Eubank, Marcela Melara, Diego Perez-Botero, and Arvind Narayanan. Shining the floodlights on mobile web tracking - a privacy survey. W2SP 2013.

[10] Michael Franklin. A statistical approach to the detection of behavioral tracking on the web. Princeton University, 2013. Undergraduate senior thesis.

[11] Michael C Grace, Wu Zhou, Xuxian Jiang, and Ahmad-Reza Sadeghi. Unsafe exposure analysis of mobile in-app advertisements. In *Proceedings of the fifth*

ACM conference on Security and Privacy in Wireless and Mobile Networks, pages 101–112. ACM, 2012.

[12] Saikat Guha, Bin Cheng, and Paul Francis. Challenges in measuring online advertising systems. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 81–87. ACM, 2010.

[13] Aniko Hannak, Piotr Sapiezynski, Arash Molavi Kakhki, Balachander Krishnamurthy, David Lazer, Alan Mislove, and Christo Wilson. Measuring personalization of web search. In *Proceedings of the 22nd international conference on World Wide Web*, pages 527–538. International World Wide Web Conferences Steering Committee, 2013.

[14] Balachander Krishnamurthy, Konstantin Naryshkin, and Craig Wills. Privacy leakage vs. protection measures: the growing disconnect. In *Proceedings of the Web*, volume 2, pages 1–10, 2011.

[15] Balachander Krishnamurthy and Craig Wills. Privacy diffusion on the web: a longitudinal perspective. In *Proceedings of the 18th international conference on World wide web*, pages 541–550. ACM, 2009.

[16] Balachander Krishnamurthy and Craig E Wills. Privacy leakage in mobile online social networks. In *Proceedings of the 3rd conference on Online social networks*, pages 4–4. USENIX Association, 2010.

[17] Bin Liu, Anmol Sheth, Udi Weinsberg, Jaideep Chandrashekar, and Ramesh Govindan. AdReveal: improving transparency into online targeted advertising. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, page 12. ACM, 2013.

[18] Anirban Majumder and Nisheeth Shrivastava. Know your personalization: Learning topic level personalization in online services. In *Proceedings of the 22nd international conference on World Wide Web*, pages 873–884. International World Wide Web Conferences Steering Committee, 2013.

[19] Delfina Malandrino, Andrea Petta, Vittorio Scarano, Luigi Serra, Raffaele Spinelli, and Balachander Krishnamurthy. Privacy awareness about information leakage: Who knows what about me? In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*, pages 279–284. ACM, 2013.

[20] Dana Mattioli. On Orbitz, Mac users steered to pricier hotels. *Wall Street Journal*, 2012.

[21] Jonathan Mayer. Tracking the trackers: Self-help tools. `https://cyberlaw.stanford.edu/blog/2011/09/tracking-trackers-self-help-tools`, September 2011.

[22] Jonathan Mayer. Tracking the trackers: Where everybody knows your username. https://cyberlaw.stanford.edu/blog/2011/10/tracking-trackers-where-everybody-knows-your-username, October 2011.

[23] Jonathan R Mayer and John C Mitchell. Third-party web tracking: Policy and technology. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 413–427. IEEE, 2012.

[24] Aleecia M McDonald and Lorrie Faith Cranor. Survey of the use of Adobe flash local shared objects to respawn HTTP cookies, a. *ISJLP*, 7:639, 2011.

[25] Jakub Mikians, László Gyarmati, Vijay Erramilli, and Nikolaos Laoutaris. Detecting price and search discrimination on the internet. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 79–84. ACM, 2012.

[26] Jakub Mikians, László Gyarmati, Vijay Erramilli, and Nikolaos Laoutaris. Crowd-assisted search for price discrimination in e-commerce: first results. *arXiv preprint arXiv:1307.4531*, 2013.

[27] Lukasz Olejnik, Tran Minh-Dung, Claude Castelluccia, et al. Selling off privacy at auction. 2013.

[28] Dillon Reisman, Steven Englehardt, Christian Eubank, Peter Zimmerman, and Arvind Narayanan. Cookies that give you away: Evaluating the surveillance implications of web tracking. Manuscript, 2014.

[29] Franziska Roesner, Tadayoshi Kohno, and David Wetherall. Detecting and defending against third-party tracking on the web. In *9th USENIX Symposium on Networked Systems Design and Implementation*, 2012.

[30] 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.

[31] Neil Thurman and Steve Schifferes. The future of personalization at news websites: lessons from a longitudinal study. *Journalism Studies*, 13(5-6):775–790, 2012.

[32] Jennifer Valentino-Devries, Jeremy Singer-Vine, and Ashkan Soltani. Websites vary prices, deals based on users' information. http://online.wsj.com/news/articles/SB10001424127887323777204578189391813881534, December 2012.

[33] David Wetherall, David Choffnes, B Greenstein, Seungyeop Han, Peter Hornyack, Jaeyeon Jung, Stuart Schechter, and Xiao Wang. Privacy revelations for web and mobile apps.

[34] Craig E Wills and Can Tatar. Understanding what they do with what they know. In *Proceedings of the 2012 ACM Workshop on Privacy in the Electronic Society*, pages 13–18. ACM, 2012.

[35] Xinyu Xing, Wei Meng, Dan Doozan, Nick Feamster, Wenke Lee, and Alex C Snoeren. Exposing inconsistent web search results with Bobble.