KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Online Tracking Technologies and Web Privacy

Güneş Acar

Supervisors:
Prof. dr. Claudia Diaz
Prof. dr. ir. Bart Preneel

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

May 2017

Online Tracking Technologies and Web Privacy

Güneş ACAR

Examination committee:
Prof. dr. ir. Hendrik Van Brussel, chair
Prof. dr. ir. Joos Vandewalle, acting chair
Prof. dr. Claudia Diaz, supervisor
Prof. dr. ir. Bart Preneel, supervisor
Prof. dr. Bettina Berendt
Prof. dr. ir. Frank Piessens
Prof. dr. ir. Vincent Rijmen
Dr. Seda Gürses
Prof. dr. Arvind Narayanan
(Princeton University)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor of Engineering Science (PhD): Electrical Engineering

May 2017

Acknowledgements

I would like to thank my supervisor Prof. Claudia Diaz and Prof. Bart Preneel for giving me the chance to pursue a PhD degree and provide an excellent research environment. Claudia was always friendly and easy-going. I feel extremely lucky to be able to work with her and see her as a role model for my professional life. Bart was always very supportive and available despite his busy schedule and gave many visionary feedback about my research.

I would like to extend my gratitude to Prof. Frank Piessens, Prof. Bettina Berendt, Dr. Seda Gürses, my assessors during my doctoral studies. I am grateful that Prof. Vincent Rijmen and Prof. Arvind Narayanan accepted to be in my examination committee and provided valuable feedback. I also thank to Prof. Hendrik Van Brussel and Prof. Joos Vandewalle for chairing my defense.

I am lucky to have Seda Gürses's friendship and advices during my first years in Leuven. I am thankful to Carmela Troncoso for steering me towards the browser fingerprinting topic, which has become quite central to my research.

I would like to thank my long term office mate and coauthor Marc Juarez for being such a nice person and sharing all those deadline all-nighters.

During my PhD I had the chance to work with excellent researchers around the world from whom I have learned a lot. I am grateful to all of them, including Steven Englehardt, Arvind Narayanan from Princeton University, Lukasz Olejnik and Claude Casteluccia from INRIA/Grenoble, Sadia Afroz from UC Berkeley, Rachel Greenstadt and Rebekah Overdorf from Drexel University, Sören Preibusch from Microsoft Research, Bettina Berendt, Nick Nikiforakis and Frank Piessens from KU Leuven.

I had many productive visits at excellent research centers including Princeton/C-ITP, Privatics team at INRIA Grenoble and Information Security Group at the UCL. I am grateful to all my hosts for the friendly and welcoming atmosphere they provided.

Sara Cleemput was so kind to help with the Dutch translation of the abstract. I would still be fixing LaTeX errors without Marc Juarez's and Michael Herrman's help with LaTeX. Playing ping-pong with Enrique, Victor, Filipe, Alan and Wouter was the most relaxing thing I have done at ESAT.

Deniz Akınç and Mehmet Ali Abdulhayoğlu helped me with everything during my first days in Leuven. I cannot thank them enough.

Péla Noë was always so nice and helped with all the problems I had with the paperwork and other practicalities. I am also grateful to Péla's chickens, who lay extremely delicious eggs. Elsy and Wim were always patient with my endless questions and made all the complicated financial paperwork very easy.

I have learned a lot from collaborating with Brendan Van Alsenoy and other CITIP members while working on the Facebook investigation of the Belgian Privacy Commission.

I would like to acknowledge FWO for funding my research visit at the Electronic Frontier Foundation in San Francisco. I am grateful to Peter Eckersley and Cooper Quintin and other EFF members for their hospitality.

Finally, I would like to thank to my parents and my brother for their outstanding patience with my temporary disappearances from the real life to work on who knows which deadline.

Güneş Acar
Leuven, May 2017

Abstract

This thesis is a study of the mechanisms that enable privacy-intrusive online tracking practices. The tracking mechanisms we focus on are resilient to removal, less transparent and hard to control.

We first give a brief historical background of online tracking ecosystem and frame it as *whack-a-mole* game which led to the rise of browser fingerprinting and evercookies, two prominent examples of advanced tracking mechanisms we study.

To contribute to the understanding of real-world practices that involve browser fingerprinting, we report on the design, implementation and deployment of FPDetective, a framework for the detection and analysis of fingerprinting with a focus on font-based fingerprinting. Using FPDetective, we conduct a large scale analysis of the million most popular websites, and discover that the adoption of fingerprinting is much higher than previous studies had estimated. We study the countermeasures against font-based fingerprinting and find a vulnerability in Tor Browser's defenses, which we help to fix.

Turning to more advanced forms of fingerprinting, we present the results of previously unreported canvas fingerprinting scripts as found on the top 100,000 Alexa sites. Our results show that 5% of the top 100,000 websites employ canvas fingerprinting making it the most common fingerprinting method ever studied. Analyzing the real-world canvas fingerprinting scripts we find them to be more advanced than the original method presented by the academic research community.

We report on the first automated study of evercookies and respawning and the discovery of a new evercookie vector, IndexedDB. We found that 10 of the 200 globally most popular websites use Flash Local Shared Objects to respawn HTTP cookies deleted by the user.

Our investigation also indicates that cookie syncing, the practice of sharing

pseudonymous identifiers between domains, amplifies the reach of respawned identifiers.

We analyze the recently introduced Battery Status API to demonstrate how seemingly innocuous information can serve as a tracking identifier when exposed to web scripts with high precision. Our study shows that the capacity of users' batteries can be discovered by exploiting the high precision level readings provided by Firefox on Linux. The capacity and the level of the battery expose a fingerprintable surface that can be used to track web users in short time intervals.

Turning to mobile platforms, we quantify the extent to which mobile applications enable surveillance at scale by sending unique identifiers over unencrypted connections. To this end, we develop a framework for data collection which downloads and executes mobile applications, captures their network traffic and detects identifiers that are sent unencrypted. Analyzing 1260 Android applications we find that, on average, 57% of a user's unencrypted mobile traffic can be linked together by a global adversary. We also show how a passive network adversary can use TCP timestamps to improve his ability to target mobile users' traffic.

Finally, we present our study on the information online retail stores share with the payment providers, specifically PayPal. Our study finds that 52% of the 881 stores we analyzed shared product names, item numbers and descriptions with the payment provider, allowing it to build extensive consumption profiles about its clients across websites.

Advanced online tracking enables surreptitious profiling of individuals' online activities and is a significant privacy concern. We hope that our research will provide ground for a more informed public and policy debate about otherwise stealthy tracking technologies.

Beknopte samenvatting

Deze thesis bestudeert technische werkwijzen die online tracking van gebruikers mogelijk maken, hetgeen een schending van hun privacy vormt. De trackingmechanismen waar wij ons op concentreren zijn bestand tegen verwijdering, minder transparant en moeilijk onder controle te houden.

Eerst geven we een kort historisch overzicht van het online tracking ecosysteem, dat we kunnen zien als een whack-a-mole spel dat heeft geleid tot de opkomst van browser fingerprinting en evercookies, twee belangrijke voorbeelden van geavanceerde trackingmechanismen die we bestuderen.

Met het oog op een beter begrip van praktische toepassingen die gebruik maken van browser fingerprinting beschrijven we het ontwerp, de implementatie en de uitrol van FPDetective. Dit is een raamwerk voor het blootleggen en analyseren van fingerprinting, met een focus op lettertypegebaseerde fingerprinting. Met behulp van FPDetective voeren we een analyse uit van de één miljoen populairste websites waarbij we ontdekken dat fingerprinting veel wijdver breider is dan de schattingen uit vorige studies. We bestuderen ook de verdedigingsmaatregelen tegen lettertypegebaseerde fingerprinting en vinden een zwakte in de verdediging van de Tor browser die we helpen oplossen.

Wat betreft meer geavanceerde vormen van fingerprinting stellen we de resultaten voor van tot op heden ongerapporteeerde canvas fingerprintingscripts die werden gevonden in de top 100 000 Alexa sites. Onze resultaten tonen aan dat 5% van de top 100 000 websites canvas fingerprinting gebruikt, dit is bijgevolg de meest voorkomende fingerprintingmethode ooit bestudeerd. Als we canvas fingerprintingscripts uit de echte wereld analyseren, ontdekken we dat ze meer geavanceerd zijn dan de originele methodes voorgesteld door academici.

We rapporteren ook over de eerste geautomatiseerde studie over evercookies en respawning, en over de ontdekking van een nieuwe evercookie vector, IndexedDB. We ontdekten dat 10 van de 200 wereldwijd populairste websites Flash Local Shared Objects gebruiken om HTTP cookies te respawnen nadat ze verwijderd

zijn door de gebruiker.

Ons onderzoek toont verder aan dat cookie synchronisatie, het delen van pseudonieme identifiers tussen verschillende domeinen, het bereik van gerespawnede identifiers vergroot.

We analyseren de recent geïntroduceerde batterijstatus API om aan te tonen hoe informatie die onschuldig lijkt, kan dienen als een tracking identifier wanneer ze met grote nauwkeurigheid geopenbaard wordt aan web scripts. Onze studie toont dat de capaciteit van de batterij ontdekt kan worden door de nauwkeurige niveaulezing te benutten die in Linux door Firefox verstrekt wordt. De capaciteit en het niveau van de batterij vormen een fingerprintbaar oppervlak dat gebruikt kan worden om webgebruikers zeer frequent te tracken.

Wat betreft mobiele platformen, meten we de mate waarin mobiele toepassingen surveillance op grote schaal mogelijk maken door unieke identifiers te versturen over ongeëncrypteerde verbindingen. Hiervoor ontwikkelen we een raamwerk voor dataverzameling. Dit raamwerk downloadt mobiele toepassingen en voert ze uit, het registreert alle netwerkverkeer dat hierbij plaatsvindt en detecteert identifiers die ongeëncrypteerd verstuurd worden. We analyseren 1260 Android toepassingen en achterhalen dat een globale tegenstander gemiddeld 57% van het ongeëncrypteerd mobiel verkeer van een gebruiker aan elkaar kan linken. We tonen ook hoe een passieve netwerktegenstander TCP timestamps kan gebruiken om zijn nauwkeurigheid nog te verhogen.

Ten slotte presenteren we onze studie over de informatie die online winkels delen met betaalproviders, meerbepaald PayPal. Onze studie toont aan dat 52% van de 881 winkels die we analyseerden productnamen, artikelnummers en beschrijvingen meedelen aan de betaalprovider, waardoor deze laatste uitgebreide consumptieprofielen kan opstellen van zijn klanten over verschillende websites heen.

Geavanceerde online tracking maakt verdoken profiling van de online activiteiten van individuen mogelijk en vormt een significant privacy-probleem. We hopen dat ons onderzoek de basis zal leggen voor een beter geïnformeerd publiek debat en beleidsdebat over anderszins verdoken trackingtechnologieën.

Contents

List of Figures

List of Tables

Abbreviations

API Application Programming Interface.

CSS Cascaded Style Sheets.

DNT Do Not Track.

ETag Entity Tag.

FTC Federal Trade Commission.

HPKP HTTP Public Key Pinning.

HTTP Hypertext Transfer Protocol.

LSO Local Shared Object.

OS Operating System.

RTB Real Time Bidding.

SOP Same Origin Policy.

VPN Virtual Private Network.

W3C World Wide Web Consortium.

Part I

Online Tracking and Web Privacy

Chapter 1

Introduction

The World Wide Web, which was originally imagined as a "web" of hyperlinked documents for scientific publishing, has evolved into a powerful and ubiquitous platform. The omnipresence of the web in diverse application domains, along with technologies that allow websites to identify users have given rise to privacy problems that did not exist before. One such problem is online tracking, monitoring individuals' browsing behavior across websites.

Online tracking enables surreptitious surveillance of individuals' activities and may lead to revealing information about users' religious beliefs, medical conditions, sexual orientations or political affiliations. Detailed behavioral profiles can be used to target more relevant ads, nudge users to buy certain products, entice them to install unwanted apps [103] or influence voters during electoral process [16, 86].

The main goal of this thesis is to advance the understanding of online tracking by providing an in-depth technical analysis of tracking technologies and their deployment. We focus on advanced, resilient and elusive forms of tracking such as browser fingerprinting, evercookies and cookie syncing.

Online tracking have many different uses. Most prominently, online behavioral advertising leverage tracking to target users with ads based on their profiles, interests, emotions [92, 113] and *moments* [53]. Several data broker companies augment these online profiles by merging them with information about users' offline activities including shopping at retail stores, voter registrations and criminal records [94]. Yet, there may be other reasons for tracking such as analytics, fraud detection and account protection.

A primary concern with online tracking is the lack of user consent and awareness about the constant collection of user activities. Studies suggest that users are concerned about how their online activities are monitored and shared with other parties [66,68,106,108] and they have little awareness about the technologies that enable it [66]. Although there are mechanisms provided by the ad industry such as opt-out pages, users find them to be very confusing and unusable [31, 61, 66]. Exhibiting a lack of faith in the organizations [63], millions of users turn to privacy enhancing technologies such as ad blockers [84, 93].

Tracking may also be used by governments who piggyback on the commercial tracking infrastructure. Documents revealed by the former NSA contractor Edward Snowden show that intelligence agencies use advertising identifiers sent over the network to identify the traffic and location of individuals [5, 102].

Price steering and discrimination are two other practices that can utilize behavioral profiles compiled with online tracking. Price steering refers to the practice of reordering search results to place more expensive items at the top [49]. Price discrimination, on the other hand, refers to the practice of showing different prices to different users for the same product [49]. Researchers found evidence of price steering and discrimination based on users' click and purchase history, their location and device they use to access the service, among others [49,70,109]. Although price discrimination is a common practice in the offline world, online tracking may enable more personal and targeted forms of discrimination that can be based on sensitive attributes of users' profiles [76]. Motivated by the evidence of price discrimination practices, European Commission recently started an investigation on price discrimination by e-commerce sites [28].

Targeted political advertising is another privacy issue that received a lot of attention lately. Tim Berners-Lee, the inventor of the web, is among those who raised concerns about the questionable practices such as voter suppression [48,72] and targeting different, possibly conflicting messages to different groups [23].

Privacy risks of online tracking and behavioral advertising are recognized by the law and policy makers around the world. In the European Union data protection regime, explicit user consent needs to be taken before collecting browser fingerprint or storing cookies — unless certain exceptions apply [85]. In the US, the Federal Trade Commission (FTC) is tasked to protect consumers against unfair or deceptive practices. The FTC has taken several enforcement actions against technology companies including Google, Facebook and online advertising networks that failed to honor opt outs [29].

Finally, by requiring websites to load and run scripts from untrusted third parties, advertising and tracking enables *drive-by attacks* that can compromise users' computers [36, 117]. Researchers found that even highly popular and reputable

websites such as New York Times and BBC served ransomware to their visitors through the advertising networks they embed on their site [96]. Researchers found that malware authors use browser fingerprinting to check whether a device is vulnerable [46] or belongs to a security researcher or honeypot [97].

1.1 Contributions of the Thesis

The overarching contribution of this thesis is to advance the understanding of online tracking. Our studies have motivated several follow up projects, and had significant real-world impact which includes improvement of web standards, identification of vulnerabilities in countermeasures as well as increased attention to the topic in public and regulatory debate.

Pioneering studies of real-world fingerprinting practices. First, we report on the design and implementation of FPDetective, a framework for the detection and analysis of browser fingerprinting with a focus on font-based fingerprinting. Using FPDetective, we analyze the most popular one million websites and discover that the adoption of fingerprinting is much higher than previous studies had estimated. We evaluate countermeasures designed to defend against fingerprinting and find weaknesses in them that can be exploited to bypass their protection. The study helped fix the vulnerabilities in the countermeasures, raise awareness about browser fingerprinting and its different uses.

Canvas fingerprinting is another type of browser fingerprinting that was first suggested by researchers in 2012 [74]. We present the results of previously unreported canvas fingerprinting scripts as found on the top 100,000 Alexa sites. We find canvas fingerprinting to be the most common fingerprinting method ever studied, with more than 5% prevalence. Analysis of the real-world scripts revealed that they went beyond the techniques suggested by the academic research community.

Automated analysis of cookie syncing, evercookies and respawning. We develop a method for the automated detection of cookie syncing, evercookies and cookie respawning. We use the *strace* debugging tool for low-level monitoring of the browser and the Flash plugin player. We find respawning by Flash cookies on 10 of the 200 most popular sites. We uncover a new evercookie vector, IndexedDB which is a new storage Application Programming Interface (API) in the browser. We find respawned identifiers used in cookie syncing, which amplifies the effect of reinstantiating by passing reinstantiated identifiers to other domains.

A new device fingerprinting vector based on the Battery Status API. Analyzing a new browser API that exposes battery information to web scripts we show how seemingly innocuous information provided by the API can serve as a tracking identifier when implemented incorrectly. We show that Firefox's implementation of the Battery Status API allows the discovery of a battery's capacity, provides short-term identifiers that facilitate tracking and can potentially be used for reinstantiating identifiers (*respawning*).

We propose a solution that reduces the Battery Status API's fingerprintable surface by rounding the level readings provided by the API. Our fix does not cause any loss in the effective functionality of the API. We filed a bug report for Mozilla Firefox to communicate the problem and the proposed solution. The fix was quickly implemented and deployed by Mozilla engineers shortly after our bug report.

After the publication of our study, the editors of the World Wide Web Consortium (W3C) Battery Status API standard recognized the privacy risks presented in our study and updated the standard to mention the potential privacy threats [58]. After a follow-up study discovered the use of the API seemingly for tracking purposes [38], several manufacturers including Mozilla removed the Battery Status API from the browsers and underlying rendering engines [2, 10, 89].

Large-scale, automated study on surveillance implications of mobile apps. We present an automated analysis of 1260 Android apps from 42 app categories and show how mobile apps enable third party surveillance by sending unique identifiers over unencrypted connections.

We show how a network adversary can improve his ability to target mobile users' traffic by using TCP timestamps for passive network fingerprinting. The analysis framework we develop for the study is able to download and run mobile applications, capture their network traffic and automatically detect identifiers that are sent unencrypted.

We evaluate two mobile ad-blocking tools: Adblock Plus for Android [6] and Disconnect Malvertising [7]. Our analysis reveals that these tools have a limited effect preventing mobile apps from leaking identifiers.

Investigate the tracking capabilities of online payment providers. We conduct the first industry-wide, empirical survey that quantifies the flows of customer data from 881 merchants to PayPal. We investigate which items of personal data and which transaction details merchants are sharing with PayPal as customers complete their checkout. We quantify the prevalence of data flows towards PayPal and measure the amount of data shared above pure order totals. Our survey of the ecosystem also looks for per-sector differences in data sharing

with payment providers or whether more popular websites leak more or less personal details.

1.2 Structure of the Thesis

The thesis consists of two parts. In Part I, we present an introduction to the thesis by stating its objectives, contributions and the motivations behind it. In Chapter II, titled 'Online Tracking Mechanisms," we present the state-of-the-art in online tracking studies to contextualize the thesis. We conclude Part I by presenting the conclusion and potential future work for our thesis.

Part II consists of six publications included in the thesis:

1. A brief historical background of online tracking ecosystem and *raison d'être* of advanced tracking techniques like browser fingerprinting and evercookies [77].
2. Analysis and survey of font-based browser fingerprinting [19].
3. Study of advanced online tracking, including canvas fingerprinting, evercookies and cookie syncing [18].
4. Privacy analysis of the Battery Status API [81].
5. Analysis of surveillance implications of mobile application tracking [111].
6. Study of information leakage by online payment platforms [90].

Chapter 2

Online Tracking Mechanisms

In this chapter, we provide an overview of the different online tracking mechanisms that we have studied. For each topic, we state our contributions and notable research conducted since then.

Online tracking is virtually invisible to users who have little to no control over the processing and further dissemination of their personal information. Even website publishers who embed resources from third-parties may be unaware of the exact purpose and behavior of the third-party scripts that they embed on their websites. Thus, transparency of tracking technologies should be improved to have an informed public and regulatory debate.

Increasing the transparency of the online tracking ecosystem requires a thorough technical analysis of web technologies and protocols, particularly due to the elusive nature of advanced tracking mechanisms.

We first provide an overview of cookie-based tracking, which is the traditional form of online tracking that has been in use since the mid-nineties [62]. The tracking enabled by cookies is well understood and modern browsers provide interfaces that enable users to express their cookie preferences as well as block, inspect and clear cookies.

In contrast, more persistent forms of online tracking have the potential to circumvent users' tracking preferences, are hard to discover and resilient to removal. Our studies focus on three main advanced tracking techniques: evercookies, cookie syncing and browser fingerprinting.

We then discuss the privacy risks of mobile applications that send cookies or other identifiers in unencrypted channels. We conclude with an overview of

countermeasures available against online tracking.

2.1 Tracking with HTTP Cookies

Cookies were invented in 1994 at Netscape to address a fundamental limitation of the Hypertext Transfer Protocol (HTTP) protocol. The protocol did not provide support to keep state or memory. Thus, it was not possible to remember shopping cart content, store language preferences or manage login state.

Some of the proposals to address this limitation involved adding unique identifiers to each browser. Disgruntled by the privacy invasive proposals, Netscape engineer Lou Montulli came up with the idea of cookies [71]. Cookies are small pieces of text-based data that are stored in the browser and contain identifiers, preferences or other strings. They are domain-specific rather than global and isolated per domain, which prevents linkability across multiple contexts. The Same Origin Policy (SOP) [112] is the primary web security mechanism to enforce the isolation between origins [1]. SOP prevents domains reading each other's cookies or, for instance, a malicious script hijacking an online banking session by stealing authentication cookies.

The domain isolation property of cookies was undermined by the introduction of third-parties that are present on multiple websites. Whenever a user visits a site where a third-party is present, the third-party will receive the user's cookie. The unique identifier in the cookie enables the third-party to link together the user's visits across different websites.

The process is illustrated in Figure 2.1, where an imaginary third-party called `advertising.com` is present on two websites. First, `advertising.com` places a new cookie with a unique identifier using the `Set-Cookie` HTTP response header. Whenever the user visits either of the websites, the browser will send any existing cookies to their respective domains in the *Cookie* HTTP header [2]. Hence, `advertising.com` will receive the cookie it previously stored. The unique identifier in the cookie makes it trivial to match the visits across the two sites.

Today, cookies are the most prevalent mechanism for third-party tracking [65,95]. According to a recent survey of the most visited 1 million sites, there are over 81,000 third-parties present on at least two websites. Of those, a handful of

[1] Origin is defined by the tuple: <scheme, host, port> e.g., `http://example.com:80` or `https://example.com:443`

[2] Cookie attributes such as path and secure flag also play a role determining when the cookie will be sent.

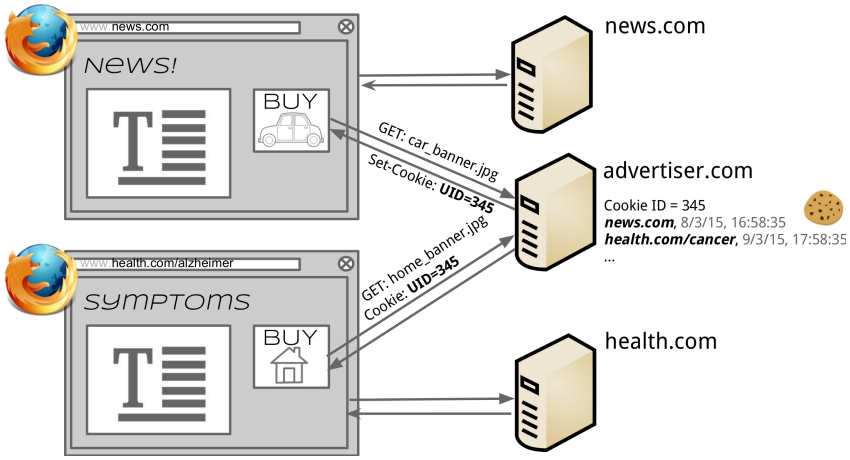


Figure 2.1: Cookie-based third-party tracking

giant technology companies including Google, Facebook and Twitter are present on more than 10% of sites [38].

Our contributions. Cookies can be used for many different purposes, such as storing language preferences or session-based tokens that enable authentication. To be useful for tracking cookies need to be long-lived and contain a unique identifier. In [18] we developed a method for detecting unique and persistent identifiers stored in cookies. Our method uses the Ratcliff-Obershelp algorithm [24]. Detecting identifiers allowed us to study the interaction between HTTP cookies and Flash cookies (or Local Shared Objects (LSOs)).

Follow-up work. In a recent longitudinal study of third-party tracking, Lerner et al. used Internet Archive to investigate the evolution of third-party tracking from 1996 to 2016. Using the archived version of websites, the researchers presented a unique view of the increase in prevalence and complexity of third-party tracking in the last three decades [62].

2.2 Evercookies

Evercookies (also known as zombie cookies or supercookies) take advantage of the lack of an interface for managing certain in-browser storage mechanisms to

be more resilient against removal by the user. This includes, for example, Flash cookies, localStorage and cache Entity Tag (ETag).

In 2010, Soltani et al. found that Flash cookies are used to regenerate previously removed HTTP cookies, a technique referred to as "respawning" [101]. They found that 54 of the 100 most popular sites stored Flash cookies, 41 of which contained matching content with the HTTP cookies. The matching content is interpreted as an indication of redundant storage to enable respawning.

In 2010, Samy Kamkar presented the "Evercookie," a resilient tracking library that uses multiple storage mechanism including Flash cookies, localStorage, sessionStorage and cache ETags [54]. Kamkar used a set of novel techniques, such as storing identifier strings in a canvas image that is then force-cached and read from the cached image on subsequent visits. Instead of just respawning HTTP cookies by Flash cookies, the Evercookie script checks the cleared vectors in the background and respawns all the missing identifiers from any copy that persists. This makes Evercookie extremely resilient against removal by users.

In 2011, Ayenson et al. found the first use of cache ETags and localStorage for respawning cleared cookies [22]. This was the first study that found browser cache and HTML5 localStorage mechanisms used as evercookies.

In 2011, McDonald and Cranor analyzed the home pages of the top 100 sites and 500 other randomly selected websites. They found only two cases of respawning on the top 100 websites and no respawning in the other 500 sites [67].

Respawning cookies cleared by users can be seen as a way to circumvent users' privacy preferences and is thus controversial. The relative low prevalence of respawning found by McDonald and Cranor may be due to the high profile settlement that was triggered by previous results [100].

Our contributions. In [18] we described an automated detection method for evercookies and cookie respawning. Applying this method, we detected respawning by Flash cookies on 10 of the 200 most popular sites and found 33 different Flash cookies were used to respawn over 175 HTTP cookies on 107 of the top 10,000 sites. We also uncovered a new evercookie vector, IndexedDB that had not been reported before.

Follow up work. In 2015, independent researcher Yan Zhu demonstrated a way to abuse the HTTP Public Key Pinning (HPKP) as an evercookie vector [118]. HPKP is a recently introduced security mechanism against fraudulent certificates. HPKP works as follows: in the first visit to a site, browsers store the hash (called "pin") of the `subjectPublicKeyInfo` field of the site's certificate [3, 14, 27]. In subsequent visits, the browser rejects certificates without a valid pin and optionally reports the validation failures to an endpoint

designated by the server [14]. This validation mechanism protects against fraudulent sites with fake certificates. HPKP also allows websites to designate a *backup pin*, which can be used when the original certificate expires or needs to be revoked. The evercookie technique by Zhu re-purposes the HPKP backup pin to store a unique identifier for each user. To retrieve these unique identifiers, the server includes a resource from a subdomain with an invalid pin. This triggers the HPKP reporting mechanism to send the pins, including the backup pin with the unique ID, to the reporting endpoint, thus enabling identification.

2.3 Cookie Syncing

As explained in Section 2.1, SOP is a security mechanism that prevents origins from accessing cookies of other origins, effectively isolating cookies per domain. This implies that domains know users by different identifiers, which are typically stored in cookies. When two domains want to carry out a joint transaction about a user, they need to map the identifiers by which they know this user. Real Time Bidding (RTB), where different parties compete to show an ad to a user, is an example of a joint transaction. In *cookie syncing* (also known as *cookie matching*), two parties exchange the identifiers associated with a given user. This allows domains to learn the identifiers assigned by other domains, effectively bypassing SOP. Further, cookie syncing enables back-end server-to-server data merges hidden from public view.

In 2013, Olejnik et al. studied cookie syncing and found that it can significantly improve the tracking and profiling capabilities of online trackers. Their study analyzed the browsing histories of 100 volunteers. They found, on average, 60 cookies are synced when a user visits 40 sites [82]. Further, results showed that Facebook (`facebook.com`) and AppNexus (`adnxs.com`) synced their cookies for 91% of the volunteers.

Our contributions. In [18], we conduct a survey of cookie syncing practices. In comparison to [82], our study is large-scale, covering the top 3,000 Alexa sites and it is based on crawling rather than crowd-sourcing. Further, we study how cookie syncing amplifies the privacy harms of respawning by disseminating the regenerated identifiers to multiple parties.

To detect cookie syncing, we look for unique identifiers in the HTTP traffic and cookie contents. In particular, we search identifiers that are known by multiple domains, which is an indication of cookie syncing.

Our study finds that 435 identifiers are synced while crawling 3,000 sites. We also count the number of identifiers known by each party and find that the

top two parties, `gemius.pl` and `doubleclick.net`, know 33 and 32 different identifiers respectively. When we measure the number of parties sharing a particular identifier, we find that, in our dataset, one of the identifiers is known to as many as 43 parties. On average, a party knows 2.0 identifiers and an identifier is shared by 3.4 parties.

With respect to the interaction between cookie syncing and respawning, our study finds that cookie syncing amplifies the effect of respawning. Considering users who clear their cookies, respawning followed by syncing can be used to distribute the regenerated identifiers to several domains. This may enable, for example, linking users' browsing history before cookie clearing to their history after cookie clearing. We quantify this effect by measuring the extent to which a user's browsing history can be linked by respawning identifiers. We find that considering cookie syncing increases the linkability of a user's browsing history from 1.4% to 11%.

Follow-up work. Following a methodology similar to ours, a 2016 study of cookie syncing on the top 100,000 sites found that 45 of the most prevalent 50 third-parties sync cookies with at least one other party [38]. Their study found `doubleclick.net` as the most prolific cookie syncing party, sharing 108 different cookies with 118 other parties.

2.4 Browser Fingerprinting

The idea of combining different characteristics of browsers to obtain a unique identifier was first proposed in 2009 by Mayer [64]. The study was based on an experiment where the researcher collected the fingerprints of 1328 web clients. By hashing the concatenated contents of the device properties `navigator`, `screen`, `navigator.plugins` and `navigator.mimeTypes`, he found that more than 96% of the browsers had a unique fingerprint.

A year later, Eckersley conducted a study called *Panopticlick*, where he collected the fingerprints of nearly half a million browsers. Panopticlick used an extended set of fingerprint features that include system fonts, timezone and HTTP Accept headers. The study showed that 94.2% of browsers with Flash or Java had a unique fingerprint [35].

The Panopticlick study showed that *stateless tracking* is feasible, i.e., websites can track users without storing any cookies or identifiers in their browsers. This has grave implications for online privacy because it allows websites to silently circumvent preferences expressed in privacy settings and invalidate practices

Browser Characteristic	bits of identifying information	one in x browsers have this value	value
Limited supercookie test	0.35	1.28	DOM localStorage: Yes, DOM sessionStorage: Yes, IE userData: No
Hash of canvas fingerprint	9.03	524.39	017e6b3b355af5bb9c5c0da3d7cccd8c
Screen Size and Color Depth	2.4	5.28	1920x1080x24
Browser Plugin Details	6.76	108.41	Plugin 0: Shockwave Flash; Shockwave Flash 24.0 r0; libflashplayer.so; (Shockwave Flash; application/x-shockwave-flash; swf) (FutureSplash Player; application/futuresplash; spl).
Time Zone	2.01	4.03	-60
DNT Header Enabled?	1.18	2.26	False
HTTP_ACCEPT Headers	2.49	5.6	text/html, */*; q=0.01 gzip, deflate, br en-US,en;q=0.5
Hash of WebGL fingerprint	9.0	511.57	955c85482ea84b8d1844d5c874e3c41c
Language	0.77	1.71	en-US
System Fonts	17.94	251181.0	Abyssinica SIL, Accanthis ADF Std, Accanthis ADF Std No2, Accanthis ADF Std No3, Amiri, Amiri Quran, Asana Math, Bitstream Charter, Cabin, Century Schoolbook L, cmex10, cmmi10, cmr10, cmsy10, Comfortaa, Courier, Courier 10 Pitch, DejaVu Sans, DejaVu Sans Condensed, DejaVu Sans Light, DejaVu Sans Mono, DejaVu Serif, DejaVu Serif Condensed, Dingbats, Droid Arabic Naskh, Droid Sans, Droid Sans Armenian, Droid Sans Ethiopic, Droid Sans Fallback, Droid Sans Georgian, Droid Sans Hebrew, Droid Sans Japanese, Droid Sans Mono, Droid Sans Thai, Droid Serif, esint10, eufm10, FontAwesome, FreeMono, FreeSans, FreeSerif, gargi, Garuda, Gentium, Gentium Basic, Gentium Book Basic, GentiumAlt, GFS Artemisia, GFS Baskerville, GFS BodoniClassic, GFS Complutum, GFS Didot, GFS Didot Classic, GFS Didot Rg, GFS Gazis, GFS Neohellenic Rg, GFS Olga, GFS Porson, GFS Solomos, GFS Theokritos, Gillius ADF, Gillius ADF Cd, Gillius ADF No2, Gillius ADF No2 Cd, Inconsolata, IPAexGothic, IPAexMincho, IPAGothic, IPAMincho, IPAPGothic, IPAPMincho, Junicode, KacstArt, KacstBook, KacstDecorative, KacstDigital, KacstFarsi, KacstLetter, KacstNaskh, KacstOffice, KacstOne, KacstPen, KacstPoster, KacstQurn, KacstScreen, KacstTitle, KacstTitleL, Kedage, Khmer OS, Khmer OS System, Kinnari, Latin Modern Math, Lato, Lato Black, Lato Hairline, Lato Light, Liberation Mono, Liberation Sans, Liberation Sans Narrow, Liberation Serif, Linux Biolinum Keyboard O, Linux Biolinum O, Linux Libertine Display O, Linux Libertine Initials O, Linux Libertine Mono O, Linux Libertine O, LKLUG, LM Mono 10, LM Mono 12, LM Mono 8, LM Mono 9, LM Mono Caps 10, LM Mono Light 10, LM Mono Light Cond 10, LM Mono Prop 10, LM Mono Prop Light 10, LM Mono Slanted 10, LM Roman 10, LM Roman 12, LM Roman 17, LM Roman 5, LM Roman 6, LM Roman 7, LM Roman 8, LM Roman 9, LM Roman Caps 10, LM Roman Demi 10, LM Roman Dunhill 10, LM Roman Slanted 10, LM Roman Slanted 12, LM Roman Slanted 17, LM Roman Slanted 8, LM Roman Slanted 9, LM Roman Unslanted 10, LM Sans 10, LM Sans 12, LM Sans 17, LM Sans 8, LM Sans 9, LM Sans Demi Cond 10, LM Sans Quot 8, Lobster Two, Lohit Bengali, Lohit Devanagari, Lohit Gujarati, Lohit Punjabi, Lohit Tamil, Loma, Mallige, Meera, mry_KacstQurn, msam10, msbm10, Mukti Narrow, Mukti Narrow, NanumBarunGothic, NanumGothic, NanumMyeongjo, Nimbus Mono L, Nimbus Roman No9 L, Nimbus Sans L, Norasi, OpenSymbol, Padauk, Padauk Book, Phetsarath OT, Pothana2000, Purisa, Rachana, Rekha, rsfs10, Saab, Sawasdee, Standard Symbols L, STIXGeneral, STIXIntegralsD, STIXIntegralsSm, STIXIntegralsUp, STIXIntegralsUpD, STIXIntegralsUpSm, STIXNonUnicode, STIXSizeFiveSym, STIXSizeFourSym, STIXSizeOneSym, STIXSizeThreeSym, STIXSizeTwoSym, STIXVariants, Symbol, TakaoPGothic, TeX Gyre Adventor, TeX Gyre Bonum, TeX Gyre Chorus, TeX Gyre Cursor, TeX Gyre Heros, TeX Gyre Heros Cn, TeX Gyre Pagella, TeX Gyre Schola, TeX Gyre Termes, TG Pagella Math, TG Termes Math, Tibetan Machine Uni, Tlwg Typist, Tlwg Typo, TlwgMono, TlwgTypewriter, Ubuntu, Ubuntu Condensed, Ubuntu Light, Ubuntu Medium, Ubuntu Mono, Umpush, URW Bookman L, URW Chancery L, URW Gothic L, URW Palladio L, utkal, Utopia, Vemana2000, Waree, wasy10 (via Flash)
Platform	3.44	10.85	Linux x86_64
User Agent	9.56	756.57	Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0
Touch Support	0.54	1.45	Max touchpoints: 0; TouchEvent supported: false; onTouchStart supported: false
Are Cookies Enabled?	0.16	1.12	Yes

Figure 2.2: Browser fingerprint as collected by Panopticlick project.

such as clearing cookies or using private browsing mode. Stateless tracking is even stealthier than evercookies since it does not leave any trace in the browser.

Eckersley also presented a mathematical treatment of browser fingerprinting. He used Shannon entropy [99] to quantify how identifying particular browser features are. Given a browser feature with values $\{x_1, ..., x_n\}$ and probability mass function $P(x)$, one can compute the entropy of the fingerprint distribution as follows:

$$H(X) = -\sum_{i=1}^{n} \mathrm{P}(x_i) \log_2 \mathrm{P}(x_i) \tag{2.1}$$

The entropy can be calculated for either the combination of all available browser features or for just a subset such as fonts or plugins. Ranking different components by their entropy, Eckersley measured plugins and system fonts as the two most identifying features with 15.4 and 13.9 bits of entropy, respectively. The study also found that the combination of all fingerprint features amounts to 18.1 bits of entropy. This means that, on average, a browser will share its fingerprint with only one in 286,777 ($2^{18.1}$) other browsers [35].

The entropy measurements from [35] are reproduced in Table 2.1.

Shannon entropy had been previously used to measure anonymity [33, 98]. One limitation of using Shannon entropy is that it is an average measure that does not take worst cases into account. In browser fingerprinting context, this means that many browsers can be unique while others have relatively large anonymity sets.

Nikiforakis et al. [78] analyzed the techniques and adoption of three fingerprinting companies, two of which had been identified in a 2012 survey paper on web tracking [65]. While Nikiforakis et al. studied the practices of these three companies, they did not attempt to discover other, unknown fingerprinters.

Other researchers have proposed the use of additional features that are not considered in Panopticlick. These include running performance benchmarks to differentiate between JavaScript engines [73] or distinguishing by the errors in web standard test suites [75].

Predating browser fingerprinting, researchers have devised ways to detect operating systems of Internet hosts based on their network behavior [4, 116]. Kohno et al.'s influential study was the first attempt to distinguish individual devices instead of a device class or operating system. Using passive network fingerprinting, the researchers could remotely fingerprint devices by their *clock skew* [55].

Table 2.1: Entropy measurement for each browser feature in the Panopticlick dataset. Table is reproduced from [35]

Browser feature	**Entropy (bits)**
Browser Plugins	15.4
System fonts	13.9
User agent	10.0
HTTP Accept headers	6.09
Screen	4.83
Timezone	3.04
Supercookies	2.12
Cookies enabled	0.353

The following sections present an overview of different browser fingerprinting techniques that we have studied during our doctoral research.

2.4.1 Font-based Fingerprinting

A robust detection methodology with low false positive rate is needed to survey the fingerprinting practices on the web.

Based on prior studies [35,78] and our review of the code of known fingerprinters, we consider *"font detection"* to be a good indicator of fingerprinting. The reasons for this choice are three-fold: (1) as Table 2.1 shows, fonts and plugins are the two most identifying features according to the Panopticlick study [35], (2) sites can query plugins for many reasons that are not related to tracking, while there are fewer such cases for querying the list of system fonts, (3) unlike browser plugins, fonts depend on the Operating System (OS) and can be used to link a user across different browsers running on the same device [25].

Although plugins such as Flash and Java can enumerate the installed fonts, web scripts cannot query the fonts directly. However, when Flash and Java are disabled, scripts can use a side-channel inference method to check if a given font is installed [78]. The technique works by printing the same string with different fonts and comparing the dimensions of the rendered string against dimensions measured with the fallback font. When the probed font does not exist, a fallback font is used to display the string. This allows scripts to infer whether or not the font is installed in the system.

Font	Rendered Text	Width x Height
Monospace	mmmmmmmmmmlli	563 x 87
Nonexistent-font	mmmmmmmmmmlli	563 x 87
Liberation Sans	mmmmmmmmmmlli	650 x 83
NanumGothic	mmmmmmmmmmlli	700 x 83
Purisa	mmmmmmmmmmlli	663 x 117
Sawasdee	mmmmmmmmmmlli	612 x 72

Figure 2.3: Javascript-based font probing uses a side-channel based on string dimensions.

Figure 2.3 shows the dimensions of the same string when printed with different fonts. The fist row shows the string printed with the fallback font, which is Monospace in this case. The font displayed in the second row is not installed in the system and thus the string is printed using the fallback font. The fonts in the remaining rows are installed, and lead to different string dimensions than the fallback font. Thus, the dimensions reveal whether or not the fonts are installed in the system.

Our contributions. In [19], we develop the *FPDetective*[3] framework to identify and analyze web-based device fingerprinting. Figure 2.4 outlines FPDetective's components and workflow. The crawler is the main component of FPDetective and it consists of automation libraries (Selenium [9] and CasperJS [87]) and two instrumented browsers, PhantomJS [50] and Chromium [15]. We modify the source code of the browsers to log events that might be related to fingerprinting such as loading an abnormal number of fonts, or accessing specific browser properties. To facilitate analysis, logs from the browsers are parsed and inserted into a database.

Instrumenting the browsers does not help with the analysis of Flash objects, since these objects are interpreted and executed by the Adobe Flash player library. Thus, to detect Flash-based fingerprinting, we analyze the ActionScript source code of the Flash objects. To extract the Flash source code, we use mitmproxy [30], an SSL-capable intercepting HTTP proxy to capture all the HTTP(S) traffic. We then process network captures to extract Flash objects

[3] Available on: `http://homes.esat.kuleuven.be/~gacar/fpdetective/`

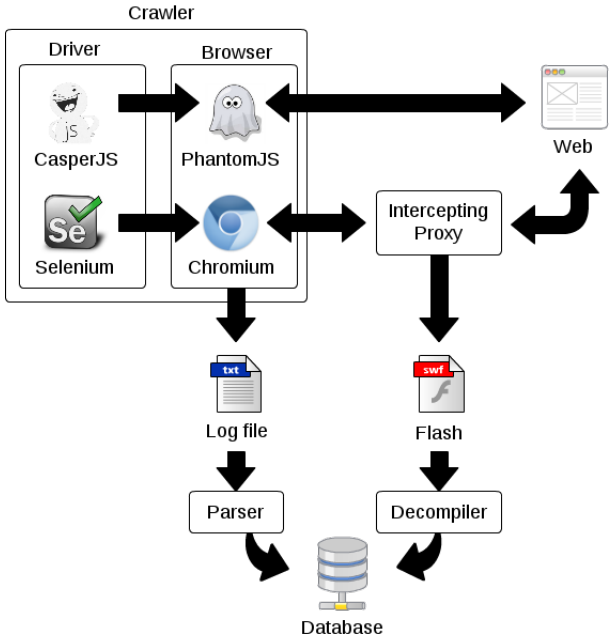


Figure 2.4: FPDetective Framework

that we identify by Flash-specific "magic numbers". Finally, we decompile Flash objects to get their source code.

We run crawls using several machines, but use a central database to combine and analyze the results. The stored data include JavaScript function calls, ActionScript API calls, HTTP requests and responses, and the list of loaded or requested fonts.

We use FPDetective to conduct a large-scale study of browser fingerprinting on the top million Alexa sites. Our study identifies 16 new fingerprinting providers and Flash objects (including commercial fingerprinting as well as in-house solutions), some of which are active in the top 500 websites.

We uncover previously unreported fingerprinting practices, such as attempting to evade detection by removing the fingerprinting script once it has executed, and collecting fingerprints through third-party widgets. Our findings highlight the rising popularity of fingerprinting and the need for more transparency, awareness and countermeasures with respect to these practices.

We evaluate existing countermeasures against fingerprinting: Tor Browser

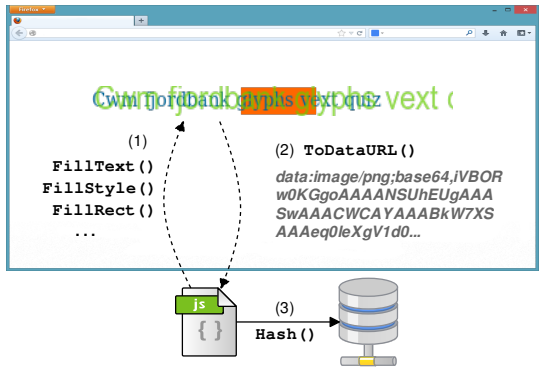


Figure 2.5: Canvas fingerprinting basic flow of operations

and Firegloves. We discover vulnerabilities in both tools that would allow a fingerprinter to bypass their protections. We also analyze the effect of the Do-Not-Track (DNT) header and find that it is ignored by the fingerprinting scripts studied in our paper.

Follow-up work. Following our study, Fifield and Egelman developed a novel fingerprinting technique that exploits the differences in the onscreen dimensions of font glyphs [41]. Unlike font-probing methods, where the fingerprinter compiles the list of system fonts, Fifield and Egelman's approach captures additional characteristics that also affect the dimensions of the rendered string. These include, for example, anti-aliasing, kerning settings, and the version of the installed font.

2.4.2 Canvas Fingerprinting

Canvas fingerprinting is a type of browser fingerprinting technique that was first presented by Mowery and Shacham in 2012 [74]. The authors showed that scripts can use the HTML5 Canvas API to exploit differences in the rendering of the same image.

Canvas fingerprinting works by drawing an invisible image onto the canvas and reading the rendered image data back. The exact rendering of the image depends on the operating system, installed fonts, font library, graphics card, graphics driver and the browser. When the drawn image contains text, differences in font rasterization such as anti-aliasing, hinting or sub-pixel smoothing may affect the exact rendering as well [74].

Figure 2.5 shows an overview of the basic steps of canvas fingerprinting. The script first draws text with the font and size of its choice and adds background colors (1). Then the script calls the Canvas API's `ToDataURL` method to get the pixel data in Base64 encoded form (2). Finally, the script takes the hash of the text-encoded pixel data, which serves as the fingerprint (3). This fingerprint may be combined with other high-entropy browser properties to increase identifiability [35]. Note that hashing in the last step is optional, and a fingerprinter may want to collect the raw pixel data. Also, the fingerprinter may try drawing different shapes, apply image effects or use text crafted to maximize the distinguishing power of the canvas fingerprint.

The entropy of canvas fingerprints has not been measured in a large-scale experiment like Panopticlick [35]. Mowery and Shacham collected data from 294 Mechanical Turk users and found that the distribution of canvas fingerprints in their dataset contains 5.73 bits of entropy. Since their experiment was too small for accurately measuring entropy, the authors had a further estimate of at least 10 bits, meaning that, on average, a browser will share its canvas fingerprint with one in 1,000 (2^{10}) other browsers.

Our contributions. Our study [18] surveys the use of canvas fingerprinting scripts in the top 100,000 Alexa sites. We modified the open source Firefox browser to log all the function calls that might be used for canvas fingerprinting. Specifically we logged `fillText`, `ToDataURL`, `strokeText`, `MozFetchAsStream`, `getImageData` and `ExtractData` methods of the Canvas API.

Using Firefox's `nsContentUtils::GetCurrentJSContext` and `nsJSUtils::GetCallingLocation` methods, we logged the URL of the calling script and the line number of the calling (initiator) code. This allows us to precisely attribute the fingerprinting attempts to correct scripts.

Crawling the homepages of the most popular 100,000 Alexa sites, we find canvas fingerprinting to be the most common fingerprinting method ever studied, with more than 5% prevalence.

Although the overwhelming majority (95%) of the scripts belong to a single provider (*addthis.com*), we discovered a total of 20 canvas fingerprinting provider domains, active on 5542 of the top 100,000 sites.

Analysis of the real-world scripts revealed that they went beyond the techniques suggested by the academic research community. For instance, AddThis's script checks support for drawing Unicode characters by printing the character U+1F603 (a smiling face with an open mouth). Further, it checks for canvas *globalCompositeOperation* support and uses the perfect pangram *"Cwm fjordbank glyphs vext quiz"* as the text string.

2.4.3 Battery Status API

Battery Status API is a W3C standard that enables web scripts to access the battery state of a mobile device or a laptop [58]. The API exposes the current battery level and predicted time to charge or discharge [58]. A potential use case for the API is that web applications can limit their energy consumption when the battery level is low.

Battery Status API does not require any permission to access the battery information. The API also does not require browsers to display a notification when it is used. This allows website and third-party scripts to access the battery information without user awareness.

Our contributions. In our study we observe that the battery level is reported by the Firefox browser on GNU/Linux with *double* precision, while just two significant digits are used on other platforms. Our study finds that it is possible to find the battery's capacity by exploiting high precision battery level readouts. The battery capacity, as well as its level, can function as short-term identifiers that facilitate tracking and can potentially be used for respawning [81].

We analyze the `UPower` source code to understand how it computes the battery level from the capacity, voltage and current charge. The resulting equations cannot be directly solved. However, the high precision readings, in combination with floating point representation errors, provide additional constraints. This allows us to brute-force candidate values for the battery's voltage and capacity.

To counter this fingerprinting vector, we propose a solution that reduces the Battery Status API's fingerprintable surface by rounding the level readings. This fix does not diminish the effective functionality of the API, since high precision readings are not required.

We filed a bug report for Mozilla Firefox to communicate the problem [80]. The fix was quickly implemented and deployed by Mozilla engineers in response to our report.

Further, our study triggered the update of the W3C recommendation that standardizes the Battery Status API. The recommendation was updated to mention the privacy risks we pointed out in our study.

The privacy concerns raised in our study and follow-up work [38] caused the browser manufacturers to remove the Battery Status API from the browsers and underlying rendering engines [2, 10, 89].

Follow-up work. In 2016, Englehardt and Narayanan conducted an online tracking survey covering the top 1 million sites. Their study found two scripts

seemingly using the Battery Status API for tracking purposes [38]. Spooren et al. investigated the potential of battery level measurements to augment active authentication [105]. Using *battery charge probability histograms*, they computed the likelihood of getting a particular battery level reading given an earlier reading. This enabled them to detect unexpected battery readings and flag suspicious authentication attempts.

2.4.4 Other Uses of Browser Fingerprinting

Although browser fingerprinting is generally associated with questionable tracking practices, researchers have also explored its potential to augment web authentication and security [20, 26, 83, 91, 104, 105, 107, 110].

In an online authentication setting, browser fingerprints may serve as an additional factor by indicating that users *have the device* that they used to login previously. This can help prevent account hijacking attempts where the user's password is stolen or guessed.

Fingerprinting can also help defend against session hijacking, where an attacker steals the authentication cookies of a victim to impersonate her. Spooren et al. note that in an effort to limit the burden of authentication, popular services such as Facebook and LinkedIn utilize long-lived sessions that may last several weeks [104]. This increases the vulnerability to an adversary who uses cross-site scripting or similar attacks to hijack an authenticated session.

For mobile devices, where browser fingerprinting is less likely to yield a unique fingerprint due to limited customization, fingerprints can be used for *negative authentication*, i.e., detecting potentially compromised sessions [104].

In contrast to these security use cases, fingerprinting can also be used by web-based malware to identify vulnerable browsers. Kolbitsch et al. note that modern web-based malware often targets specific browser configurations [56]. Analyzing real-world attack campaigns, malware researchers found examples of browser fingerprinting scripts that check whether the device is vulnerable [46] and whether it might be a honeypot [97].

2.5 Mobile tracking

Several studies have investigated the privacy implications of mobile apps, which have become ubiquitous in the last decade. An influential study by Enck et al. presented TaintDroid, a system-wide taint analysis system that allows tracking

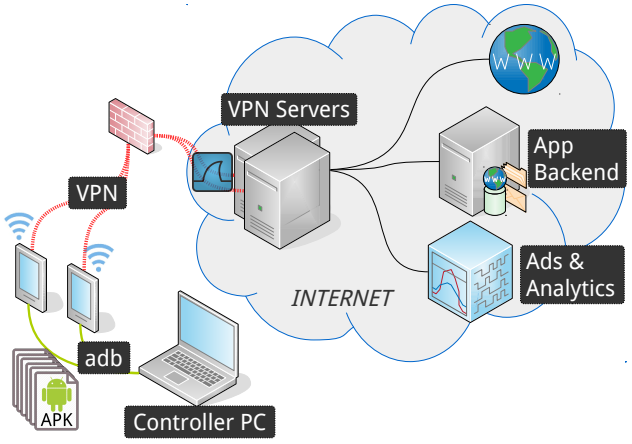


Figure 2.6: Our data collection setup consists of a Controller PC, two Android phones and two VPN servers for capturing the network traffic.

of sensitive information flows [37]. Analyzing the network traffic generated by the mobile apps, Xia et al. have shown that an adversary can attribute up to 50% of the app traffic to the real names of users [115].

Our contributions. The prior work mobile tracking have mainly focused on the sensitive information that is collected and transmitted to remote servers [37, 47, 51]. In contrast, our study quantifies the tracking capabilities of a passive network adversary who monitors the mobile app network traffic [111]. In particular, we present an automated analysis of 1260 Android apps and show how mobile apps enable surveillance by sending unique identifiers over unencrypted connections. Our study is motivated by Snowden documents, which revealed that the tracking identifiers sent over unencrypted channels are being used for surveillance purposes [8].

We follow the methodology of Englehardt et al. who have shown that third-party cookies sent over unencrypted connections can be used to link together 62-73% of a user's web browsing history [39]. We extend their work by showing how passive network fingerprinting based on TCP timestamps can be used to improve the linking of users' traffic. Following Englehardt et al., we measure the success of the adversary by the ratio of the network traffic he can link together for a given user.

We analyze 1260 Android applications from 42 app categories using the data collection framework shown in Figure 2.6. The framework includes a controller

PC, two Android smartphones and two Virtual Private Network (VPN) servers to capture the network traffic. The framework downloads and runs mobile applications, captures their network traffic and automatically detects identifiers that are sent in the clear.

Our results show that 57% of a user's unencrypted mobile traffic can be linked by a global adversary [111]. We also analyze *Adblock Plus for Android* [6] and *Disconnect Malvertising* [7], two tracking countermeasures available to privacy aware mobile users. Our analysis finds their effectiveness to be very limited against identifier leakage. Adblock Plus reduces the ratio of linkable traffic from 57% to 50%. Disconnect Malvertising performs better, reducing the linkable traffic to 38% from 57%.

Using HTTPS for the mobile app traffic can effectively protect against passive network adversaries. Further, a system-wide proxy or VPN app similar to HTTPS Everywhere browser extension can be developed to redirect insecure HTTP requests to HTTPS endpoints where possible.

2.6 Technical countermeasures against online tracking

In this section we present countermeasures available to users to defend against online tracking. Legal or regulatory countermeasures such as European Data Protection regime and consumer protection efforts by the FTC are left our as they are beyond the scope of our studies.

Do Not Track HTTP header. The Do-Not-Track (DNT) allows users to signal their preferences about tracking to websites. Do Not Track (DNT) is modeled after "Do Not Call Registry," which allow people to opt-out from receiving unsolicited telemarketing calls. DNT preference is conveyed in two ways: first, as an HTTP header field, and second as a JavaScript property in the `navigator` object.

DNT is standardized by the W3C under the name "Tracking Preference Expression" and it has already been adopted by most modern browsers [40].

Since DNT needs to be respected on the server-side, by the very companies who may profit from tracking, its effectiveness is called into question [60, 88].

Our contributions. The findings of our study on font-based browser fingerprinting suggest that DNT preferences are ignored by the fingerprinters in our dataset [19].

Browser settings. All modern browsers provide an option to disable cookies or disable only the third-party cookies. Comparing different browser settings such as disabling JavaScript, cookies and third-party cookies, Krishnamurthy et al. found that disabling third-party cookies may have significant privacy benefits without causing severe usability problems [59]. The same study found that although disabling JavaScript or cookies provide better tracking protection, their negative impact on the usability is also higher, potentially leading to refusal of service.

Krishnamurthy et al. studied the leakage of sensitive and identifying information to third parties on 120 websites. They found that 56% of the sites leaked private informations such as name, zip code or email to at least one third party [60]. Evaluating the effectiveness of existing online tracking countermeasures such as blocking cookies, disabling JavaScript and sending Do-Not-Track HTTP header, they found severe shortcomings of those protections. The limitations of the countermeasures ranged from not being able protect against certain forms of tracking to negative usability consequences.

In their study of the top million websites, Englehardt and Narayanan found Firefox's third-party cookie blocking feature very effective, as it reduced the average number of third parties per site from 17.7 to 12.6 [38].

Browser extensions. Ad-blocker extensions have been used since the late nineties [42]. According to PageFair, the number of devices that block ads reached 615 million by the end of 2016 [84].

Browser extensions use a combination of techniques to counter online tracking. They block HTTP requests based on blacklists of tracking, analytics or advertising domains. They also hide advertisement elements based on Cascaded Style Sheets (CSS) rules. Unlike other blacklist-based extensions, Privacy Badger detects trackers based on their behavior [43].

By comparing the effectiveness of several anti-tracking extensions, Ikram et al. found that NoScript and Ghostery block the most trackers within a labeled dataset of tracking scripts, 78% and 65% respectively. Privacy Badger blocked only the 37% of the scripts labeled as tracking [52].

As a response to the rise of ad blocking software, certain online websites started to use anti-adblocking scripts to deny access to visitors with adblockers. Nithyanand et al. found that 6.7% of websites in the Alexa Top 5000 use anti-adblocking scripts. News, blogs, and entertainment websites are found to be the categories that are more likely to include anti-adblocking scripts [79].

Firefox's Tracking Protection. In principle, Firefox's Tracking Protection is very similar to adblocking extensions [57]. It uses a blacklist to block requests

to tracking domains. However, Tracking Protection benefits from a better integration with the underlying browser. Testing Firefox's Tracking Protection on Alexa top 200 news sites, Kontaxis and Chew measured 67.5% reduction in the number of HTTP cookies and 44% median reduction in page load time [57].

Tor Browser. Tor is the most popular anonymous communications network with more than two million daily users [17]. Tor routes users' traffic through three relays to hide users' IP address from the websites or other communication endpoint [34]. Tor uses onion routing [45], in combination with encryption in layers to prevent any single Tor relay from linking the source and destination of a connection.

The overarching defense strategy of Tor Browser is to eliminate fingerprintable differences to make all Tor users look like the same. At the implementation level, Tor Browser uses a combination of defenses that include Private Browsing mode, disk write avoidance, value spoofing, feature removal, API reimplementation or virtualization [69].

Since the Tor Browser has a limited user base, even a partial fingerprint can be enough to identify Tor users to their non-Tor identities, breaching the anonymity provided by the network. Moreover, very complex nature of the browser code base makes it very hard to maintain the integrity and comprehensiveness of the defenses in the presence of constant code updates and the introduction of new browser APIs.

Font-based fingerprinting is a particularly important threat for Tor Browser, since the fonts depends on the operating system and can be used to link Tor users to their non-Tor identities. As a countermeasure, Tor Browser used to limit the number of fonts that can be queried or loaded in a document.

Our contributions. In [19], we discovered a vulnerability that allowed us to circumvent the Tor Browser's font limits to load an unlimited number of fonts. The Tor Browser developers swiftly fixed the problem prior to the publication of our study [19].

Follow-up work. Following the publication of our study [19], we found and reported another attack that circumvents Tor Browser's font limits by using multiple frames within a document [44]. Moreover, Fifield and Egelman published a font-based fingerprinting technique that use the differences in the onscreen dimensions of font glyphs [41]. Their fingerprinting technique was effective against Tor Browser. To address these weaknesses, the Tor Project started to ship a fixed set of fonts with the Tor Browser Bundle [1].

Although using the same set of fonts alleviates the problems associated with font-probing, the dimensions of a text can still be affected by the font rendering

settings such as hinting [21]. Research needs to be done to check whether these rendering differences can serve as an effective fingerprint for Tor users.

Chapter 3

Conclusion and Future Work

3.1 Conclusion

Our research has shown that advanced tracking mechanisms such as browser fingerprinting, evercookies and cookie syncing are actively used by third-party trackers on thousands of websites. We address the difficulty of detecting and analyzing advanced online tracking and enable similar line of work that relies on browser instrumentation.

Certain countermeasures are available against advanced tracking methods but they all suffer from limitations and weaknesses. Our line of research is critical in finding those weaknesses and help the developers to address them.

Our analysis of the Battery Status API has shown that seemingly innocuous battery information can serve as a tracking identifier. The W3C has updated the API standard to address the privacy risks as a response to our study.

We have presented a survey of leakage of personal information and shopping habits from online retail shops to payment providers. Our analysis has shown that 52% of the 881 retail sites shared product names, item numbers and descriptions with PayPal.

Finally, we have studied the extent of mass surveillance enabled by mobile application tracking. We have analyzed the network traffic of 1260 Android applications using a mobile app automation framework that we have developed, and found that 57% of a user's unencrypted mobile traffic can be linked together by a global adversary.

Advanced online tracking is a significant privacy issue that enables surreptitious profiling of individuals' online activities. We contribute to making the web a more transparent and privacy respecting space by conducting large scale studies of the online tracking technologies. We hope our findings will help having a more informed public and policy debate about otherwise stealthy technologies.

3.2 Future Work

The following future work can address the existing research gap and enable new lines of web privacy research.

Reproducible web measurement studies: The dynamic nature of the web makes it impossible to reproduce other researchers' work as the analyzed pages almost constantly change. A potential direction for future work can be to develop infrastructure and tools for reproducible analysis of online tracking. Chromium Project's Web Page Replay [13] and `mitmproxy`'s server-side replay feature [12] can be used to this end.

Record-and-replay capabilities of web traffic and sessions can be useful for multi-execution based analysis of web-based malware [56]. Similarly, information-flow security studies based on secure multi-execution such as FlowFox [32] can benefit from such capability.

Finally, longitudinal tracking studies similar to recently published [62] can reproduce the tracking practices more realistically using replayable web archives.

Using browser instrumentation for web security studies: The native-code level browser instrumentation such as the one accomplished in our work [19] can be adapted for the analysis of web based malware. The existing JavaScript malware analysis tools such as JSDetox [11] have limitations when it comes to emulating a real browser. Web-based malware with fingerprinting capabilities can easily bypass their protections. Using full-fledged instrumented browsers with low-level instrumentation can be used to overcome this problem.

Studying new web-enabled devices and IoT: New connected devices such as smart TVs and IoT devices can enable different set of tracking mechanisms that is worth studying. Although web tracking studies may offer methodological support in detecting long-term and unique identifiers, data collection and automation can be challenging for this line of research. A possible extension of this study could be to investigate cross-device tracking involving smart TVs.

Detecting Code Injection by Tor Exit Nodes: Winter et al.'s "Spoiled Onions" study has shown different ways Tor exit nodes may interfere with

the network traffic. They find several malicious or misconfigured exit nodes, that attempt to strip SSL connections or steal email credentials [114]. Although their study mentions a case where HTML code is injected by a malicious exit, it does not presents a decisive analysis due to limited data. To address this research gap, the nature and extent of privacy violating code injections by Tor exit nodes can be investigated. This study can contribute to Tor Project by flagging malicious exits who tamper with web pages.

Bibliography

[1] #13313 (Enable bundled fonts in Tor Browser) – Tor Bug Tracker & Wiki. https://trac.torproject.org/projects/tor/ticket/13313.

[2] Firefox — Notes (52.0) — Mozilla. https://www.mozilla.org/en-US/firefox/52.0/releasenotes/.

[3] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. https://tools.ietf.org/html/rfc5280.

[4] Nmap - Free Security Scanner For Network Exploration & Security Audits. http://www.nmap.org.

[5] 'Tor Stinks' presentation. http://www.theguardian.com/world/interactive/2013/oct/04/tor-stinks-nsa-presentation-document, 2013.

[6] About Adblock Plus for Android. https://adblockplus.org/android-about, 2015.

[7] Disconnect Malvertising for Android. https://disconnect.me/mobile/disconnect-malvertising/sideload, 2015.

[8] Mobile apps doubleheader: BADASS Angry Birds. http://www.spiegel.de/media/media-35670.pdf, 2015.

[9] Selenium - Web Browser Automation. http://docs.seleniumhq.org/, 2015.

[10] Bug 164213 – remove battery status api from the tree. https://bugs.webkit.org/show_bug.cgi?id=164213, 2016.

[11] Jsdetox | relentless coding. http://www.relentless-coding.com/projects/jsdetox/info, 2016.

[12] Server-side replay — mitmproxy 0.17.1 documentation. http://docs.mitmproxy.org/en/stable/features/serverreplay.html, 2016.

[13] Web page replay - webpagetest documentation. https://sites.google.com/a/webpagetest.org/docs/private-instances/web-page-replay, 2016.

[14] HTTP Public Key Pinning (HPKP) - HTTP | MDN. https://developer.mozilla.org/en-US/docs/Web/HTTP/Public_Key_Pinning, 2017.

[15] The Chromium Projects. https://www.chromium.org/, 2017.

[16] Toomey for Senate | Facebook for Business. https://www.facebook.com/business/success/toomey-for-senate, 2017.

[17] Users - Tor Metrics. https://metrics.torproject.org/userstats-relay-country.html, 2017.

[18] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The Web never forgets: Persistent tracking mechanisms in the wild. In *21st ACM Conference on Computer and Communications Security (CCS)*, pages 674–689. ACM, 2014.

[19] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. FPDetective: Dusting the Web for fingerprinters. In *20th ACM Conference on Computer and Communications Security (CCS)*, pages 1129–1140. ACM, 2013.

[20] Furkan Alaca and PC van Oorschot. Device fingerprinting for augmenting web authentication: classification and analysis of methods. In *Proceedings of the 32nd Annual Conference on Computer Security Applications*, pages 289–301. ACM, 2016.

[21] arthuredelstein. #16672 (Text rendering allows font fingerprinting) – Tor Bug Tracker & Wiki. https://trac.torproject.org/projects/tor/ticket/16672.

[22] 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.

[23] Tim Berners-Lee. Three challenges for the web, according to its inventor. http://webfoundation.org/2017/03/web-turns-28-letter/, 2017.

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

[25] Károly Boda, Ádám Máté Földes, Gábor György Gulyás, and Sándor Imre. User tracking on the web via cross-browser fingerprinting. In *Nordic Conference on Secure IT Systems*, pages 31–46. Springer, 2011.

[26] Elie Bursztein, Artem Malyshev, Tadek Pietraszek, and Kurt Thomas. Picasso: Lightweight device class fingerprinting for web clients. In *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 93–102. ACM, 2016.

[27] Ryan Sleevi Chris Evans, Chris Palmer. Public Key Pinning Extension for HTTP. https://tools.ietf.org/html/rfc7469, 2015.

[28] European Commission. Antitrust: Commission opens three investigations into suspected anticompetitive practices in e-commerce. http://europa.eu/rapid/press-release_IP-17-201_en.htm.

[29] Federal Trade Commission. Protecting Consumer Privacy in an Era of Rapid Change: Recommendations For Businesses and Policymakers. https://www.ftc.gov/reports/protecting-consumer-privacy-era-rapid-change-recommendations-businesses-policymakers, 2012.

[30] Aldo Cortesi. mitmproxy: a man-in-the-middle proxy. http://mitmproxy.org/.

[31] Lorrie Faith Cranor. Can users control online behavioral advertising effectively? *IEEE Security & Privacy*, 10(2):93–96, 2012.

[32] Willem De Groef, Dominique Devriese, Nick Nikiforakis, and Frank Piessens. Flowfox: a web browser with flexible and precise information flow control. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 748–759. ACM, 2012.

[33] Claudia Diaz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards measuring anonymity. In *International Workshop on Privacy Enhancing Technologies*, pages 54–68. Springer, 2002.

[34] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320. USENIX, 2004.

[35] Peter Eckersley. How unique is your web browser? In *Privacy Enhancing Technologies*, pages 1–18. Springer, 2010.

[36] Manuel Egele, Engin Kirda, and Christopher Kruegel. Mitigating drive-by download attacks: Challenges and open problems. In *iNetSec 2009–Open Research Problems in Network Security*, pages 52–62. Springer, 2009.

[37] William Enck, Landon P Cox, Peter Gilbert, and Patrick Mcdaniel. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. *OSDI'10 Proceedings of the 9th USENIX conference on Operating systems design and implementation*, 2010.

[38] Steven Englehardt and Arvind Narayanan. Online tracking: A 1-million-site measurement and analysis. In *Conference on Computer and Communications Security*. ACM, 2016.

[39] 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 *Proceedings of the 24th International Conference on World Wide Web*, pages 289–299, 2015.

[40] Roy T. Fielding and David Singer. Tracking Preference Expression (DNT). `https://www.w3.org/2011/tracking-protection/drafts/tracking-dnt.html`, 2016.

[41] David Fifield and Serge Egelman. Fingerprinting Web Users through Font Metrics. In *Financial Cryptography and Data Security (FC)*. Springer-Verlag, 2015.

[42] Laurie J. Flynn. THE MEDIA BUSINESS: ADVERTISING; Battle Begun on Internet Ad Blocking. `http://www.nytimes.com/1999/06/07/business/the-media-business-advertising-battle-begun-on-internet-ad-blocking.html?mtrref=query.nytimes.com&gwh=F568E9499F00F79E057B625FDE4FB7BE&gwt=pay`, 1999.

[43] Electronic Frontier Foundation. Privacy badger | electronic frontier foundation. `https://www.eff.org/privacybadger`, 2017.

[44] gacar. #5798 (Improve persistence and WebFont compatibility of font patch) – Tor Bug Tracker & Wiki. `https://trac.torproject.org/projects/tor/ticket/5798#comment:13`, 2013.

[45] David Goldschlag, Michael Reed, and Paul Syverson. Hiding routing information. In *Information Hiding*, pages 137–150, 1996.

[46] Derek Gooley. Top Exploit Kit Activity Roundup - Winter 2017 | Zscaler Blog. `https://www.zscaler.com/blogs/research/top-exploit-kit-activity-roundup-winter-2017`, 2017.

[47] MC Grace, Wu Zhou, X Jiang, and AR Sadeghi. Unsafe Exposure Analysis of Mobile In-App Advertisements. *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, 067(Section 2), 2012.

[48] Joshua Green and Sasha Issenberg. Inside the Trump Bunker, With Days to Go. `https://www.bloomberg.com/news/articles/2016-10-27/inside-the-trump-bunker-with-12-days-to-go`, 2016.

[49] Aniko Hannak, Gary Soeller, David Lazer, Alan Mislove, and Christo Wilson. Measuring price discrimination and steering on e-commerce web sites. In *Proceedings of the 2014 conference on internet measurement conference*, pages 305–318. ACM, 2014.

[50] Ariya Hidayat. PhantomJS | PhantomJS. `http://phantomjs.org/`, 2017.

[51] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. These aren't the droids you're looking for: Retrofitting Android to protect data from imperious applications. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 639–652. ACM, 2011.

[52] Muhammad Ikram, Hassan Jameel Asghar, Mohamed Ali Kaafar, Anirban Mahanti, and Balachandar Krishnamurthy. Towards seamless tracking-free web: Improved detection of trackers via one-class learning. *Proceedings on Privacy Enhancing Technologies*, 2017(1):79–99, 2017.

[53] Facebook IQ. Moments That Matter: Finding the Extraordinary in the Ordinary. `https://insights.fb.com/2015/06/09/moments-that-matter/`, 2015.

[54] Samy Kamkar. Evercookie - virtually irrevocable persistent cookies. `http://samy.pl/evercookie/`, Sep 2010.

[55] Tadayoshi Kohno, Andre Broido, and Kimberly C Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, 2005.

[56] Clemens Kolbitsch, Benjamin Livshits, Benjamin Zorn, and Christian Seifert. Rozzle: De-cloaking internet malware. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2012.

[57] Georgios Kontaxis and Monica Chew. Tracking Protection in Firefox For Privacy and Performance. In *Web 2.0 Workshop on Security and Privacy (W2SP)*. IEEE, 2015.

[58] Anssi Kostiainen and Mounir Lamouri. Battery Status API. `https://www.w3.org/TR/battery-status/`, 2012. Accessed: 24.4.14.

[59] Balachander Krishnamurthy, Delfina Malandrino, and Craig E Wills. Measuring privacy loss and the impact of privacy protection in web browsing. In *Proceedings of the 3rd symposium on Usable privacy and security*, pages 52–63. ACM, 2007.

[60] Balachander Krishnamurthy, Konstantin Naryshkin, and Craig Wills. Privacy leakage vs. protection measures: the growing disconnect. In *Proceedings of the Web 2.0 Security and Privacy Workshop*, volume 2, pages 1–10, 2011.

[61] Pedro Leon, Blase Ur, Richard Shay, Yang Wang, Rebecca Balebako, and Lorrie Cranor. Why johnny can't opt out: A usability evaluation of tools to limit online behavioral advertising. In *SIGCHI Conference on Human Factors in Computing Systems*, pages 589–598. ACM, 2012.

[62] 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 *25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX, August 2016. USENIX Association.

[63] Mary Madden and Lee Rainie. Americans' attitudes about privacy, security and surveillance. `http://www.pewinternet.org/2015/05/20/americans-attitudes-about-privacy-security-and-surveillance/`, 2015.

[64] Jonathan R. Mayer. Any person... a pamphleteer. Senior Thesis, Stanford University, 2009.

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

[66] Aleecia M McDonald and Lorrie Faith Cranor. Americans' attitudes about internet behavioral advertising practices. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 63–72. ACM, 2010.

[67] 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.

[68] William Melicher, Mahmood Sharif, Joshua Tan, Lujo Bauer, Mihai Christodorescu, and Pedro Giovanni Leon. (do not) track me sometimes:

Users' contextual preferences for web tracking. *Proceedings on Privacy Enhancing Technologies*, 2016(2):135–154, 2016.

[69] Steven Murdoch Mike Perry, Erinn Clark. The Design and Implementation of the Tor Browser [DRAFT]. `https://www.torproject.org/projects/torbrowser/design//#Implementation`, 2015.

[70] Jakub Mikians, László Gyarmati, Vijay Erramilli, and Nikolaos Laoutaris. Crowd-assisted search for price discrimination in e-commerce: First results. In *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*, pages 1–6. ACM, 2013.

[71] Lou Montulli. The irregular musings of lou montulli: The reasoning behind web cookies. `http://www.montulli-blog.com/2013/05/the-reasoning-behind-web-cookies.html`, 2013.

[72] David Z. Morris. Trump Digital Team Running "Three Major Voter Suppression Operations" | Fortune.com. `http://fortune.com/2016/10/30/trump-voter-supression-operations/`.

[73] Keaton Mowery, Dillon Bogenreif, Scott Yilek, and Hovav Shacham. Fingerprinting information in JavaScript implementations. In Helen Wang, editor, *Proceedings of W2SP 2011*. IEEE Computer Society, May 2011.

[74] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in HTML5. In *Web 2.0 Workshop on Security and Privacy (W2SP)*. IEEE, 2012.

[75] Martin Mulazzani, Philipp Reschl, Markus Huber, Manuel Leithner, Sebastian Schrittwieser, and Edgar R. Weippl. Fast and reliable browser identification with javascript engine fingerprinting. In *Web 2.0 Workshop on Security and Privacy (W2SP)*, May 2013.

[76] Arvind Narayanan. Price Discrimination is All Around You. `https://33bits.org/2011/06/02/price-discrimination-is-all-around-you/`, 2011.

[77] Nick Nikiforakis and Gunes Acar. Browse at your own risk. *IEEE Spectrum*, 51(8):30–35, 2014.

[78] 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 (SP), 2013 IEEE Symposium on*, pages 541–555. IEEE, 2013.

[79] Rishab Nithyanand, Sheharbano Khattak, Mobin Javed, Narseo Vallina-Rodriguez, Marjan Falahrastegar, Julia E Powles, Emiliano De Cristofaro, Hamed Haddadi, and Steven J Murdoch. Ad-blocking and counter blocking: A slice of the arms race. In *6th USENIX Workshop on Free and Open Communications on the Internet (FOCI 16)*. USENIX Association, 2016.

[80] Lukasz Olejnik. Bug 1124127 - Round Off Navigator Battery Level on Linux. `https://bugzilla.mozilla.org/show_bug.cgi?id=1124127`, 2015. Accessed: 30.2.15.

[81] Lukasz Olejnik, Gunes Acar, Claude Castelluccia, and Claudia Diaz. The leaking battery - A privacy analysis of the HTML5 battery status API. In *Data Privacy Management, and Security Assurance - 10th International Workshop, DPM 2015, and 4th International Workshop, QASA 2015, Vienna, Austria, September 21-22, 2015. Revised Selected Papers*, pages 254–263, 2015.

[82] Lukasz Olejnik, Tran Minh-Dung, and Claude Castelluccia. Selling Off Privacy at Auction. In *Annual Network and Distributed System Security Symposium (NDSS)*. IEEE, 2014.

[83] Erol Ozan. Password-free authentication for social networks. In *Computing and Communication Workshop and Conference (CCWC), 2017 IEEE 7th Annual*, pages 1–5. IEEE, 2017.

[84] PageFair. The state of the blocked web: 2017 global adblock report. `https://pagefair.com/downloads/2017/01/PageFair-2017-Adblock-Report.pdf`, 2017.

[85] Article 29 Working Party. Opinion 9/2014 on the application of Directive 2002/58/EC to device fingerprinting. `http://ec.europa.eu/justice/data-protection/article-29/documentation/opinion-recommendation/files/2014/wp224_en.pdf`, 2014.

[86] Adam Pasick. Facebook says it can sway elections after all—for a price — Quartz. `https://qz.com/922436/facebook-says-it-can-sway-elections-after-all-for-a-price/`, 2017.

[87] Nicolas Perriault and CasperJS Contributors. CasperJS, a navigation scripting and testing utility for PhantomJS and SlimerJS. `http://casperjs.org/`, 2016.

[88] Mike Perry. Do Not Beg: Moving Beyond DNT through Privacy by Design. In *W3C Workshop: Do Not Track and Beyond*. W3C, 2012.

[89] Chris Peterson. Bug 1313580 - remove web content access to battery api. https://bugzilla.mozilla.org/show_bug.cgi?id=1313580, 2016.

[90] Sören Preibusch, Thomas Peetz, Gunes Acar, and Bettina Berendt. Shopping for privacy: Purchase details leaked to paypal. *Electronic Commerce Research and Applications*, 15:52–64, 2016.

[91] Davy Preuveneers and Wouter Joosen. Smartauth: Dynamic context fingerprinting for continuous user authentication. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 2185–2191. ACM, 2015.

[92] Shayla Price. How Emotional Targeting Converts More Leads. https://blog.kissmetrics.com/emotional-targeting-converts-more-leads/, 2017.

[93] Lee Rainie, Sara Kiesler, Ruogu Kang, Mary Madden, Maeve Duggan, Stephanie Brown, and Laura Dabbish. Anonymity, privacy, and security online. *Pew Research Center*, 5, 2013.

[94] Edith Ramirez, Julie Brill, Maureen K Ohlhausen, Joshua D Wright, and Terrell McSweeny. Data Brokers–A Call for Transparency and Accountability. *Federal Trade Commission, Tech. Rep*, 2014.

[95] F. Roesner and T. Kohno und D. Wetherall. Detecting and defending against third-party tracking on the web. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI*, 2012.

[96] Jérôme Segura. Large Angler Malvertising Campaign Hits Top Publishers - Malwarebytes Labs | Malwarebytes Labs. https://blog.malwarebytes.com/threat-analysis/2016/03/large-angler-malvertising-campaign-hits-top-publishers/, 2016.

[97] Jérôme Segura and Eugene Aseev. Operation Fingerprint: A look into several Angler Exploit Kit malvertising campaigns. https://malwarebytes.app.box.com/v/operation-fingerprint, 2016.

[98] Andrei Serjantov and George Danezis. Towards an information theoretic metric for anonymity. In *International Workshop on Privacy Enhancing Technologies*, pages 41–53. Springer, 2002.

[99] Claude Elwood Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[100] Ryan Singel. Online tracking firm settles suit over undeletable cookies. https://www.wired.com/2010/12/zombie-cookie-settlement/, 2010.

[101] 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.

[102] Ashkan Soltani, Andrea Peterson, and Barton Gellman. NSA uses Google cookies to pinpoint targets for hacking. `https://www.washingtonpost.com/news/the-switch/wp/2013/12/10/nsa-uses-google-cookies-to-pinpoint-targets-for-hacking/`, 2013.

[103] Aditya K Sood and Richard J Enbody. Malvertising–exploiting web advertising. *Computer Fraud & Security*, 2011(4):11–16, 2011.

[104] Jan Spooren, Davy Preuveneers, and Wouter Joosen. Mobile device fingerprinting considered harmful for risk-based authentication. In *Proceedings of the Eighth European Workshop on System Security*. ACM, April 2015.

[105] Jan Spooren, Davy Preuveneers, and Wouter Joosen. Leveraging battery usage from mobile devices for active authentication. *Mobile Information Systems*, 2017.

[106] Joseph Turow, Michael Hennessy, and Nora A Draper. The tradeoff fallacy: How marketers are misrepresenting american consumers and opening them up to exploitation. 2015.

[107] 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.

[108] Blase Ur, Pedro Giovanni Leon, Lorrie Faith Cranor, Richard Shay, and Yang Wang. Smart, useful, scary, creepy: perceptions of online behavioral advertising. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 4:1–4:15, New York, NY, USA, 2012. ACM.

[109] Jennifer Valentino-DeVries, Jeremy Singer-Vine, and Ashkan Soltani. Websites Vary Prices, Deals Based on Users' Information. `https://www.wsj.com/articles/SB10001424127887323777204578189391813881534`, 2012.

[110] Tom Van Goethem, Wout Scheepers, Davy Preuveneers, and Wouter Joosen. Accelerometer-based device fingerprinting for multi-factor mobile authentication. In *International Symposium on Engineering Secure Software and Systems*, pages 106–121. Springer, 2016.

[111] Eline Vanrykel, Gunes Acar, Michael Herrmann, and Claudia Diaz. Leaky Birds: Exploiting Mobile Application Traffic for Surveillance. In *Financial Cryptography and Data Security - 20th International Conference, FC 2016*, Lecture Notes in Computer Science, pages 1–22. Springer-Verlag, 2016.

[112] W3C. Same Origin Policy - Web Security. `https://www.w3.org/Security/wiki/Same_Origin_Policy`, 2010.

[113] David Waterhouse. Video Ad Tech Company Unruly Brings Emotional Targeting To Programmatic Video Advertising. `https://unruly.co/news/article/2015/03/25/unruly-custom-audiences/`, 2015.

[114] Philipp Winter, Richard Köwer, Martin Mulazzani, Markus Huber, Sebastian Schrittwieser, Stefan Lindskog, and Edgar Weippl. Spoiled onions: Exposing malicious tor exit relays. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 304–331. Springer, 2014.

[115] Ning Xia, Han Hee Song, Yong Liao, and Marios Iliofotou. Mosaic: Quantifying Privacy Leakage in Mobile Networks. *SIGCOMM '13 Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, (ii):279–290, 2013.

[116] Michal Zalewski. p0f v3 (version 3.06b). `http://lcamtuf.coredump.cx/p0f3/`.

[117] Apostolis Zarras, Alexandros Kapravelos, Gianluca Stringhini, Thorsten Holz, Christopher Kruegel, and Giovanni Vigna. The dark alleys of madison avenue: Understanding malicious advertisements. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, pages 373–380. ACM, 2014.

[118] Yan Zhu. Weird New Tricks for Browser Fingerprinting. `https://zyan.scripts.mit.edu/presentations/toorcon2015.pdf`, 2015.

Part II

Publications

List of Publications

International Journals

1. Preibusch, S., Peetz, T., Acar, G., and Berendt, B. Shopping for privacy: Purchase details leaked to paypal. *Electronic Commerce Research and Applications 15* (2016), 52–64

    - See p. 177, this publication is an extended version of the FC 2015 paper.

International Conferences and Workshops with Proceedings

2. Vanrykel, E., Acar, G., Herrmann, M., and Diaz, C. Leaky Birds: Exploiting Mobile Application Traffic for Surveillance. In *Financial Cryptography and Data Security - 20th International Conference, FC 2016* (2016), Lecture Notes in Computer Science, Springer-Verlag, pp. 1–22

    - See p. 151

3. Olejnik, L., Acar, G., Castelluccia, C., and Diaz, C. The leaking battery - A privacy analysis of the HTML5 battery status API. In *Data Privacy Management, and Security Assurance - 10th International Workshop, DPM 2015, and 4th International Workshop, QASA 2015, Vienna, Austria, September 21-22, 2015. Revised Selected Papers* (2015), pp. 254–263

    - See p. 135

4. Preibusch, S., Peetz, T., Acar, G., and Berendt, B. Purchase details leaked to PayPal. In *Financial Cryptography and Data Security*

- *19th International Conference, FC 2015* (Puerto Rico, 2015), Lecture Notes in Computer Science, Springer-Verlag, p. 10

5. Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., and Diaz, C. The Web never forgets: Persistent tracking mechanisms in the wild. In *21st ACM Conference on Computer and Communications Security (CCS)* (2014), ACM, pp. 674–689

    - See p. 91

6. Juarez, M., Afroz, S., Acar, G., Diaz, C., and Greenstadt, R. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014), ACM, pp. 263–274

7. Acar, G., Juarez, M., Nikiforakis, N., Diaz, C., Gürses, S., Piessens, F., and Preneel, B. FPDetective: Dusting the Web for fingerprinters. In *20th ACM Conference on Computer and Communications Security (CCS)* (2013), ACM, pp. 1129–1140

    - See p. 59

Magazine Articles

8. Nikiforakis, N., and Acar, G. Browse at your own risk. *IEEE Spectrum 51*, 8 (2014), 30–35

    - Invited article, featured on the cover page of the IEEE Spectrum magazine. See p. 51.

Technical Reports

9. Vanrykel, E., Acar, G., Herrmann, M., and Diaz, C. Exploiting Unencrypted Mobile Application Traffic for Surveillance. Technical report, 2016

10. Acar, G., Alsenoy, B. V., Piessens, F., Diaz, C., and Preneel, B. Facebook Tracking Through Social Plug-ins. Technical report prepared for the Belgian Privacy Commission, 2015

Miscellaneous

11. Acar, G. Position paper: Obfuscation for and against device fingerprinting. In *Symposium on Obfuscation* (New York City NY USA, 2014), p. 5

Publication

Browse at your own risk

Publication Data

Nikiforakis, N., and Acar, G. Browse at your own risk. *IEEE Spectrum 51*, 8 (2014), 30–35

This article was featured on the cover of the IEEE Spectrum magazine. The running title of the article was "Browser Fingerprinting and the Online-Tracking Arms Race."

Contributions

- I co-wrote the article (roughly 20-25% of it) and studied news articles and standards to trace back the reactions to the introduction of cookies in the early days of the web.

Browse at your own risk

Nick Nikiforakis[1] and Gunes Acar[2]

[1] KU Leuven, Dept. of Computer Science, iMinds-DistriNet, Leuven, Belgium
nick.nikiforakis@cs.kuleuven.be
[2] KU Leuven ESAT/COSIC, iMinds, Leuven, Belgium
gunes.acar@esat.kuleuven.be

On the 5th of July, 1993, the magazine "The New Yorker" published a cartoon made by Peter Steiner which depicts two dogs sitting in front of a computer with the caption: "On the Internet, nobody knows you're a dog." This comic was understood to highlight the anonymity inherent on the Internet and how this anonymity could assist even canines, in keep their identities private.

Fast-forward 20 years, and now this is unfortunately no longer the case. Given the tracking technologies in use today, interested parties do not only know that you are a dog, but they also know the color of your fur, how often you visit the vet, and what is your favorite dog treat. Advertising agencies collaborate with websites and gather your browsing data, creating effectively a browsing profile for every user. These browsing profiles can be so specific, that they allow advertisers to target populations such as "young mothers with teenage children", "Acculturated Latinos with income more than 50K" or those interested in buying "allergy relief" products.

When this is combined with our volunteered self-revelation on social media, or offline purchases or whereabouts as recent plans by Facebook and Google suggest, the amount of information that other people have about us becomes too close for comfort.

In this article, we examine the history of tracking on the web and its evolution, paying particular attention to *fingerprinting*, a new type of tracking which does not require the, now widely recognized, *cookies*.

Origins of web tracking

Browser cookies were first introduced in Netscape Navigator in 1994 and quickly adopted by all other browsers, available at the time. Cookies are small pieces of text that is stored in user's browser by the visited websites. They are then made available to the website in subsequent visits, which allow websites to recognize returning visitors or keep a state about a visitor, such as the items in the shopping cart. Cookies can also be used to store login credentials. Sites like *facebook.com* can remember logged-in users, without requiring them to

provide their passwords with every new interaction.

Unfortunately, the stateful nature of cookies was quickly abused to allow companies to track users across sites. This can be better explained through an example. Suppose that a user browses to *travel.com*, whose homepage includes an advertising banner located on the web servers of *advertiser.com*. Therefore, as part of the process of rendering *travel.com*'s homepage, the user's browser will request the banner from advertiser.com. The web server of *advertiser.com*, uses this opportunity to not only send the banner but also to set a cookie on the user's machine. Later, when the user browses to other websites affiliated with advertiser.com, e.g., rental.com, the tracking website receives its previously-set cookies, recognizes the user, and creates a profile of the user's browsing habits. These *third-party* cookies and their implications started receiving attention from various media outlets and privacy organizations starting from mid-nineties. Over the years, people realized that the set of websites they visit, reveals much information about themselves, ranging from their gender and age, to their political affiliations and health status. The possession of such knowledge by advertising networks and other interested third parties, comes with potentially dire privacy consequences, given that users had no control and transparency over the processing and sharing of their data.

Reactions to Tracking

Cookies may be one of the most disputed technologies when it comes to privacy. Beginning from the mid-nineties, one can see news articles about cookies with headlines containing catchy expressions like "death of privacy" and many variations of big brother metaphor. For example a New York Times article from 1999 describes cookies as "comprehensive privacy invaders." Some of them also include reflections of the idea that privacy maybe an obstacle to online businesses. A Times article titled "Invasion Of Privacy" from 1997 quotes a government bureaucrat saying: "The future is in electronic commerce," all that's holding it up is "this privacy thing."

Even technical standardization efforts got their share of these discussions. A 1997 coalition letter to browser manufacturers signed by privacy organizations, expressed their support for the first cookie standard (RFC 2109), which states that *third-party* cookies should be blocked "to prevent possible security or privacy violations." But advertising companies pushed back harder, and finally none of the two popular browsers of that time followed the specification and allowed *third-party* cookies. While modern browsers give the ability to users to reject third-party cookies, even today, most of them come with this feature turned off by default (Safari being a notable exception).

In a 2012 study on Online Behavioral Advertising, the majority of participants

opposed it, describing it as both smart and creepy. While the majority of non-technical people still do not understand the exact workings of third-party tracking, the last years have witnessed a surge of more advanced self-help tools which, the somewhat knowledgeable user, can use to defend against web tracking.

In 2005, browsers started adding the "Private Browsing" mode in their browsers, to give users the option of visiting websites without leaving long-term cookies on their machines. Next to the efforts from the browser vendors, independent developers started producing privacy-preserving tools that users could install in their browsers. Today, the most popular Mozilla extension is AdBlock Plus which blocks ads and the third-party cookies that trackers use. Moreover, recent tools like Ghostery and Mozilla's Lightbeam allow the user to get a glimpse of the number of trackers on each website and how these trackers collaborate between seemingly unrelated websites. Finally, recent studies have shown that a large percentage of users delete their cookies on a regular basis, a fact that points to at least some understanding of cookies and their implications.

Recently opt-out websites have appeared, which users can visit and opt-out from tracking from a selection of parties, some of which are operated by advertising organizations and given as examples of industry self-regulation. Studies, however, have found that these sites have confusing interfaces and major usability flaws, which prevent users from correctly opting out. Moreover, even if the user succeeds in opting out, half of the companies kept their existing cookies and continued tracking, Stanford researchers found in 2011.

Adaptation of trackers

When users started deleting their cookies, the various tracking parties had to develop new ways of identifying these users. Most methods that came out of this process had one thing in common; they tried to hide the same tracking information that they used to place in cookies, in some other "corner" of the browser. One popular technique was the use of *Flash cookies*. Flash cookies are conceptually similar to normal cookies, but they are specific to the Flash plugin. In the past, a website could hide information in Flash cookies, that would survive the clearing of cookies. Moreover, Flash cookies were used to re-spawn other deleted cookies, by copying the information from Flash cookies back into normal cookies. Companies were able to use this practice for a number of years before researchers started publicizing their practices in 2008. Today, modern browsers give users the ability to delete all cookies from within the browser menus.

While most tracking techniques, today, still rely on the concept of "stuffing" information in a user's browser, there is one emerging family of techniques that

does not require storing information, namely web-based device fingerprinting. Web-based device fingerprinting is the process of collecting information about unique characteristics of the devices that users utilize to browse the web. Under the assumption that each user owns and operates her own device, identifying a device is equivalent to identifying the user behind it. Prior research by Peter Eckersley in 2010, showed that combining several benign browser attributes can lead to an almost "perfect" fingerprint, i.e., more than 94% of user devices have unique fingerprints. These attributes include the dimensions of a user's screen, her timezone, and the lists of browser plugins and system fonts.

Discovering the State of Practice

In our work, we studied the current fingerprinting ecosystem in order to, among others, identify the current providers of commercial fingerprinting products and services, how commercial solutions differ from Eckersley's fingerprinting methodology, and which websites fingerprint their users. It is important to identify the current big players in the fingerprinting ecosystem, because of their correlating power. More specifically, a user's fingerprint is usually stored in a database of "known devices" with contextual information and can be used instead of cookies, as a mean to identify the user in the future. Since these companies are used by multiple websites, they can correlate their data across sites, and thus provide insights about a user to a website, gathered by the rest of their clients, even if the former has never "seen" that user before. Insights can take the form of demographics, as well as a "threat score" which purportedly can alert the website of a malicious user.

We started our analysis by first identifying and studying the fingerprinting code of three popular fingerprinting providers. Through this process, we were able to create a taxonomy which we used to compare each company to the rest as well as to Eckersley's methodology. There, we discovered that not only commercial fingerprinting solutions are up to date with academic findings, but that they also include novel methods of their own. For instance, we discovered that one company was using a novel method of identifying the installed fonts on a user machine, without relying on the machine volunteering this information as Eckersley did. This finding is very interesting since it shows that fingerprinters adopt to the changing browser environment so that they can still obtain user-specific information. In addition to this novel font-probing technique, we also discovered the use of Flash as a way of identifying whether a users are using other networked computers to hide their original IP address, as well as the presence of intrusive fingerprinting plugins that are installed silently on a user's machine, by being bundled with certain applications that a user downloads and installs.

Using the information gathered from these three companies, we then designed

and implemented a program that autonomously browses the web and "senses" when a website is trying to fingerprint it. The purpose of this experiment was to identify more players in the fingerprinting ecosystem, that may be less popular than the originally-studied companies, but still collect device identifying information. Using our tool, we discovered 16 new instances of device fingerprinting, some of which were in-house solutions, and the rest offered by commercial companies as a product. Our results showed that web-based device fingerprinting is offered by many more companies than originally thought and is adopted by many popular websites, i.e, 145 of top ten thousand websites. We also found that more than 400 websites of the Internet's most popular one million have been using JavaScript-only fingerprinting to work with Flash-less devices such as iPhone or iPad. Moreover, our experiment revealed that, at this point in time, users keep on being fingerprinted even if they use have enabled the "Do Not Track" preference, in their browser.

Limitations of Current Defenses

At this point it should be evident that fingerprinting is not only actively practiced on the modern web, but also that it can have a negative effect on a user's privacy. The question that thus naturally arises is what are people doing about it.

As part of our research at KU Leuven, we examined various available tools which, among others, claim to beat fingerprinting. One popular way is installing browser extensions that allow users to change a series of values, customarily used to identify each browser. Using these extensions, Firefox users on Linux, can pretend to be Internet Explorer users on Microsoft Windows. Other extensions go further, reporting false dimensions for a user's screen size and limiting the probing of fonts.

Through our analysis of these extensions, we discovered that they were all trivially bypassable by any determined party. This is because modern browsers are huge pieces of software each with its own intricacies. These intricacies are discoverable and can thus give away the true nature of a browser, regardless of what the browser may claim to be.

These intricacies not only void the effects of these extensions but also bring the user to a position worse than his original one. Since each extension is slightly different than others, a tracker, after determining the true nature of a user's browser, may also be able to determine which extension the user is utilizing. If he succeeds, then the user, instead of, say, one in 10 million Firefox users, becomes one in 3,000 Firefox users with that particular extension installed. An analogous real-life example would be trying to change the make and model of your car, by replacing the stickers. Not only what you did is obvious for

someone looking beyond stickers, but now, in terms of anonymity, you are worse than when you begun, since the majority of people leave their stickers intact.

Future Directions

Our findings highlight the difficulty of pretending to be someone you are not, at least in the context of browsing environments, and show that easy solutions are no match for determined trackers. Researchers, including ourselves, are currently actively studying the problem trying to come-up with effective solutions.

A straightforward way to combat fingerprinting today is to stop the scripts from loading in the browser, similar to advertising blockers that many users currently utilize. By maintaining a blacklist of fingerprinting scripts, an anti-fingerprinting plugin could detect their loading and prohibit their execution. One issue that arises from this technique is that the blacklist must be constantly updated to keep up with new URLs and evasion techniques from the trackers. Another issue, is that we currently do not know whether the loading of fingerprinting scripts is necessary for the functionality of any single website. It is definitely possible for a website operator to refuse the loading of their site unless the fingerprinting scripts are present and operational.

Another, more systematic way of approaching the problem of fingerprinting, would be to create a setup in which many users share the same fingerprint. For instance, one can envision a browsing environment located on the cloud, where the user's browser simply becomes a terminal for communicating with this cloud browser. In this setup, many users with diverse browsing environments, will be hidden behind a single cloud- based, browsing environment. Granted, a tracker can determine that a cloud-based setup is being utilized, however, given a wide adoption of this service, any user of the service will be indiscernible from any other user.

Conclusion

Given that ads are the number one industry of the web, and that tracking is a crucial component of ads, we believe that web tracking in general and fingerprinting in specific are here to stay. Right now, as a community, we are still discovering the nature and magnitude of this privacy problem and only after we understand it, we can hope to properly address it.

Publication

FPDetective: Dusting the Web for Fingerprinters

Publication Data

Acar, G., Juarez, M., Nikiforakis, N., Diaz, C., Gürses, S., Piessens, F., and Preneel, B. FPDetective: Dusting the Web for fingerprinters. In *20th ACM Conference on Computer and Communications Security (CCS)* (2013), ACM, pp. 1129–1140

Contributions

- Principal author.

FPDetective: Dusting the Web for Fingerprinters

Gunes Acar[1], Marc Juarez[1,2], Nick Nikiforakis[3]
Claudia Diaz[1], Seda Gürses[1,4], Frank Piessens[3], and Bart Preneel[1]

[1] KU Leuven ESAT/COSIC, iMinds, Leuven, Belgium
{name.surname}@esat.kuleuven.be
[2] IIIA-CSIC, Bellaterra, Spain
mjuarez@iiia.csic.es
[3] KU Leuven, Dept. of Computer Science, iMinds-DistriNet, Leuven, Belgium
{nick.nikiforakis,frank.piessens}@cs.kuleuven.be
[4] New York University, Dept. of Media, Culture, and Communication, NY, USA

Abstract. In the modern web, the browser has emerged as the vehicle of choice, which users are to trust, customize, and use, to access a wealth of information and online services. However, recent studies show that the browser can also be used to invisibly fingerprint the user: a practice that may have serious privacy and security implications.

In this paper, we report on the design, implementation and deployment of FPDetective, a framework for the detection and analysis of web-based fingerprinters. Instead of relying on information about known fingerprinters or third-party-tracking blacklists, FPDetective focuses on the detection of the fingerprinting itself. By applying our framework with a focus on font detection practices, we were able to conduct a large scale analysis of the million most popular websites of the Internet, and discovered that the adoption of fingerprinting is much higher than previous studies had estimated. Moreover, we analyze two countermeasures that have been proposed to defend against fingerprinting and find weaknesses in them that might be exploited to bypass their protection. Finally, based on our findings, we discuss the current understanding of fingerprinting and how it is related to Personally Identifiable Information, showing that there needs to be a change in the way users, companies and legislators engage with fingerprinting.

1 Introduction

In 2010, Eckersley demonstrated that benign characteristics of a browser's environment, like the screen dimensions and list of installed fonts, could be combined to create a unique device-specific fingerprint [7]. Of the half million users who participated in Eckersley's experiment, 94.2% of those using Flash or Java had a unique device fingerprint, and could thus be identified and tracked without the need for stateful client-side technologies, such as browser or Flash cookies.

Fingerprinting user devices through the browser is an increasingly common practice used of advertising and anti-fraud companies. Stateless user tracking allows advertising companies to sidestep the limitations imposed by regulation on *cookies* in Europe and the United States. Moreover, with the advent of smartphones and tablets, fingerprinting allows advertisers to augment previously gathered user-data and track the user across devices. Anti-fraud companies advertise fingerprinting as a means to protect users and web applications against malevolent actors, for instance, by detecting the use of stolen credentials and identifying Sybil attacks [6]. Most of the time, these services are based on massive device reputation databases where device fingerprints are stored along with the device owners' web history and "reputation scores."

Device fingerprinting raises serious privacy concerns for everyday users. Its stateless nature makes it hard to detect (no cookies to inspect and delete) and even harder to opt-out. Moreover, fingerprinting works just as well in the "private-mode" of modern browsers, which cookie-conscious users may be utilizing to perform privacy-sensitive operations. In recent research, Nikiforakis et al. [18] analyzed the techniques and adoption of three fingerprinting companies, two of which had been identified by Mayer and Mitchell in a 2012 survey paper on web-tracking [13]. While the authors studied the practices of these three companies, they did not attempt to discover other fingerprinters and explore the methods adopted by these.

This paper aims to shed light on current device fingerprinting practices through three main contributions. The first is the design and implementation of *FPDetective*,[5] a framework for identifying and analyzing web-based device fingerprinting without relying on a list of known fingerprinters.

Second, we use FPDetective to conduct a large-scale study of web-based device fingerprinting in the top million Alexa sites. Using FPDetective, we were able to identify 16 new fingerprinting scripts and Flash objects (including commercial

[5]The FPDetective framework is available here: `http://homes.esat.kuleuven.be/~gacar/fpdetective/`

fingerprinting as well as in-house solutions), some of which are active in the top 500 websites, showing that fingerprinting is much more prevalent than previous studies estimated. Further, we uncovered previously unreported fingerprinting practices, such as attempting to evade detection by removing the fingerprinting script once the device has been fingerprinted, and collecting fingerprints through third-party widgets. Our findings highlight the rising prevalence of fingerprinting and, in turn, the need for more transparency, awareness and counter-measures with respect to these practices.

Finally, we evaluate the Tor Browser and Firegloves, two privacy-enhancing tools that have "fingerprinting-resistance" as a design goal. We discovered vulnerabilities in both tools that would allow a fingerprinter to identify users. This illustrates the difficulty of protecting against device fingerprinting. We also tested whether enabling the Do-Not-Track (DNT) header had any impact on the behavior of fingerprinting scripts, and found that it does not.

The rest of the paper is organized as follows. Section 2 reviews the state-of-the-art in device fingerprinting methods. The FPDetective framework is described in Section 3. Section 4 motivates the use of font detection to identify potential fingerprinters, and explains our data analysis methodology. Section 5 presents the experimental results of our study of the top million Alexa websites. We evaluate in Section 6 the Tor browser, Firegloves, and DNT as countermeasures to fingerprinting. We discuss in Section 7 the uses of fingerprinting and its relationship to privacy concerns. Finally, we offer our conclusions in Section 8.

2 Device Fingerprinting

A device *fingerprint* is a set of system attributes that, for each device, take a combination of values that is, with high likelihood, unique, and can thus function as a device identifier. These attributes include, for example, the device's screen size, the versions of installed software, and the list of installed fonts. Attributes that take more diverse values (e.g., the list of fonts) are more identifying than values shared by many devices (e.g., version of the operating system). Similarly, attributes with values that are more stable over time (i.e., that change only infrequently or very gradually) facilitate identification compared to those that change often and unpredictably.

Web-based device fingerprinting is the process of collecting sufficient information through the browser to perform stateless device identification. These fingerprints may then be used as identifiers for tracking the device in the web. By tracking, we refer to the linking of visits to (one or multiple) web pages as made by the same device.

There are many reasons why web applications may need device-related information, e.g., to correctly render content or to serve device compatible media. Thus, there are many APIs that enable applications to query these attributes. At the same time, these APIs can be used to learn enough attribute values to obtain a device fingerprint that is, for practical purposes, unique. When a user browses to a webpage that includes fingerprinting software, her device fingerprint may be collected and compared to a database of known devices. Using such techniques, known devices can be matched, and previously unknown devices can be added to the database. Depending on the use case, the database entry for each device is augmented with contextual and behavioral information with each visit of the user to a monitored webpage.

In this section we briefly introduce device fingerprinting methods. Some of the discussed methods have already been encountered in real-life fingerprinting code, while others have been shown to work but are not yet known to have been adopted by fingerprinters [18].

2.1 JavaScript-based

From its original inception in 1995 all the way to today, JavaScript has emerged as the de-facto client-side, programming language of the web. The expressiveness of JavaScript, combined with its "forgiving" nature when it comes to programming errors (such as missing semicolons) and, most importantly, its ubiquitous availability in modern browsers, has made it an indispensable tool of modern web sites. Virtually all non-static web pages use JavaScript and a 2010 study by Yahoo showed that only 1% of actual human visitors have JavaScript disabled [24].

JavaScript is used by programmers mainly to dynamically manipulate a page's DOM, enrich user experience through asynchronous requests and responses, and offload non-critical, server functionality to the client. Its privileged position inside the browser, however, also makes it a strong fingerprinting tool. The JavaScript-accessible browser resources that have, historically, been probed the most, are the following:

- `navigator`: The `navigator` object contains information about the browser vendor and specific browser version, the supported plugins and MIME types, as well as relatively coarse-grained information about the operating system and architecture on which the browser is executing.
- `screen`: The `screen` object contains information about the resolution of the user's monitor (height and width) and the color and pixel depth.

Mayer, in 2009, reported on an experiment where he fingerprinted 1328 web clients [12]. By hashing the concatenated contents of the `navigator`, `screen`, `navigator.plugins` and `navigator.mimeTypes`, Mayer was able to uniquely identify more than 96% of the browsers. A year later Eckersley, through the Panopticlick project[6], fingerprinted nearly half a million browsers and by extending the set of fingerprinted features with fonts, timezones and a browser's ACCEPT headers, was able to uniquely identify 94.2% of the visitors' browsing environments [7]. Eckersley also showed that the list of installed fonts is one of the most identifying features of a system. This list can be obtained either through JavaScript by measuring and then comparing the dimensions of text rendered with different fonts [18], or through browser plugins.

Other researchers have proposed the use of performance benchmarks for differentiating between JavaScript engines [15], errors in standard test-suites [17] and differences in the appearance of `canvas` elements created through JavaScript [16]. Furthermore, a user's browsing history, which can be recovered exploiting JavaScript's visited-link color feature [9], has also been shown to uniquely identify users [19]. In general, while these methods have not yet been encountered in deployed fingerprinting products [18], they could potentially be used to increase the accuracy of the gathered fingerprints.

2.2 Plugin-based

The latest version of HTML, HTML5, together with the advanced capabilities of JavaScript and Cascading Style Sheets, give web developers today the ability to create feature-rich web applications. This, however, was not the case with older versions of HTML, as their abilities to deliver interactive rich Internet applications, like games, video and music were limited. Third-party companies, like Adobe, developed plugins and platforms to create, deliver and render interactive multimedia content. The clear winner of this "plugin-war" was Adobe Flash, with Java being a distant second.

As with JavaScript, the adoption of popular third-party plugins gives fingerprinters the ability to extract numerous features. Eckersley used Java and Flash to obtain the list of fonts installed in a device [7], since font-enumeration API calls are made available by the Flash and Java plugins. In addition to font extraction, commercial fingerprinting companies use Flash to circumvent HTTP proxies set up by the user and get more fine-grained information about the device, such as the specific operating system kernel version, or the presence of multiple-monitor setups [18].

[6] `https://panopticlick.eff.org/`

2.3 Extension-based

The modular nature of modern browsers has allowed developers to create extensions that add new functionality, remove undesired features, and modify others. Unlike plugins, extensions are not enumerable through JavaScript and thus can only be detected by their possible side-effects. For instance, Mowery et al. [15] showed that it is possible to deduce custom whitelists from the popular NoScript plugin, simply by requesting scripts from domains and later inspecting whether the scripts successfully executed, by searching for predetermined JavaScript objects in the global address space. The deduced whitelists can be used as an extra fingerprint feature. Nikiforakis et al. [18] showed that user-agent-spoofing extensions can also be discovered due to inconsistencies in the reported browsing environment when each extension is active.

2.4 Header-based & Server-side

Yen et al. [23] performed a fingerprinting study, similar to Eckersley's, by analyzing month-long logs of Bing and Hotmail. The authors found that IP addresses and user-agent headers may help in tracking users with high precision and recall.

Predating fingerprinting at the browser-level, researchers had shown that it is possible to not only remotely learn the operating system of a particular host on the Internet [1, 25], but also to fingerprint multiple physical devices hidden behind NATs through their clockskew, by analyzing the TCP timestamps of network packets [11].

While these techniques may be less accurate compared to the aforementioned in-browser fingerprinting methods, their pure server-side nature makes their detection very difficult, if not impossible.

3 FPDetective Framework

In this section we describe FPDetective, a framework for detecting web-based device fingerprinting. FPDetective is designed as a flexible, general purpose framework that can be used to conduct further web privacy studies. FPDetective is freely available and can be downloaded from `http://homes.esat.kuleuven.be/~gacar/fpdetective`.

Figure 1 outlines FPDetective's components and workflow. The main component of FPDetective is a crawler, whose purpose is to visit websites and collect data about events that might be related to fingerprinting, such as the loading of fonts, or accessing specific browser properties. These logs are parsed and committed to a central database in a relational structure. In order to detect Flash-based fingerprinting, all browser traffic is directed through an intercepting proxy that logs all the HTTP(S) traffic between the browser and the web server. These network dumps are parsed to extract Flash objects, that are then decompiled using a free, third-party decompiler and stored in the database.

In our analysis of the data collected by FPDetective we focus on font detection. We would like to emphasize that this choice was made in order to facilitate the analysis of the data, and the data gathered by FPDetective can be analyzed differently, based on other fingerprinting classification criteria. Further, the FPDetective framework can easily be adapted for use in Web privacy studies unrelated to fingerprinting. The framework is developed with modularity in mind using Python, C++, JavaScript and MySQL programming/scripting languages. Researchers can customize the framework to carry out different experiments by replacing the script that FPDetective executes when it visits the sites. We warmly welcome other researchers to provide their comments, contribute to the project, or fork their own software out of FPDetective.

The remainder of this section describes each component of FPDetective in more detail.

3.1 Components

Crawler: The crawler features two instrumented browsers, PhantomJS[7] and Chromium[8]. We chose PhantomJS to collect data related to JavaScript-based fingerprinting for its minimal use of resources. We used Chromium to investigate Flash-based fingerprinting, since PhantomJS does not have plugin support and thus cannot run Flash objects. CasperJS[9] and Selenium[10] were used to drive the browsers to websites and navigate through the pages.

To build instrumented versions of the browsers, we modified parts of the WebKit source code, which is the rendering engine used by both Chromium and PhantomJS. It should be noted, however, that during the course of our study, Chromium Project announced that they leave WebKit for a new rendering engine called Blink, which is again based on WebKit [2]. We preferred to work

[7] `http://phantomjs.org/`
[8] `http://www.chromium.org/`
[9] `http://casperjs.org/`
[10] `http://docs.seleniumhq.org/`

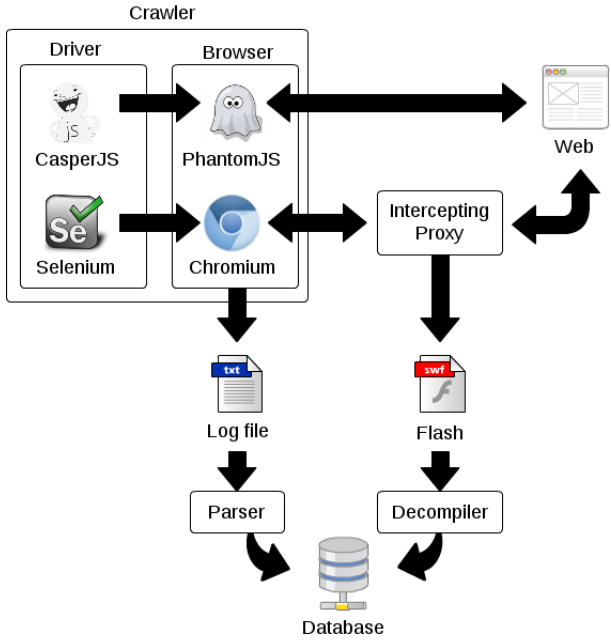


Figure 1: FPDetective Framework

at the native code level instead of developing browser extensions or JavaScript patches for several reasons: to detect events that are not JavaScript-based (especially those related to fonts); to detect the origin of events more precisely; and to defend against JavaScript attacks that block or circumvent extensions and getter methods[11].

The modifications allow the crawler to intercept and log accesses to the following browser and device properties, which could be used for fingerprinting:

- access to the following `navigator` properties and methods: `userAgent`, `appCodeName`, `product`, `productSub`, `vendor`, `vendorSub`, `onLine`, `appVersion`, `language`, `plugins`, `mimeTypes`, `cookieEnabled()`, `javaEnabled()`
- access to `navigator.plugins`: `name`, `filename`, `description`, `length`
- access to `navigator.mimeTypes`: `enabledPlugin`, `description`, `suffixes`, `type`

[11] `http://code.google.com/p/chromium/issues/detail?id=55084`

- access to `window.screen` properties: `horizontalDPI`, `verticalDPI`, `height`, `width`, `colorDepth`, `pixelDepth`, `availLeft`, `availTop`, `availHeight`, `availWidth`
- access to `offsetWidth` and `offsetHeight` properties and `getBoundingRect` method of HTML elements.
- font load attempts by intercepting calls to `CSSFontFace::getFontData` and `CSSFontSelector::getFontData` methods

Parser: The parser is used to extract relevant data from the logs generated by the crawler, and to store them in the database. It also tags sites with a label if a known fingerprinting script is found in the HTTP requests made for this visit.

Intercepting Proxy: In order to obtain Flash files for static analysis, we redirected traffic through mitmproxy [4], an SSL-capable intercepting proxy. We used the mitmdump module to log all the HTTP traffic passing through the proxy, and the libmproxy library to parse and extract Flash files based on content sniffing. More specifically, we detected Flash files through Flash-specific "magic numbers" appearing the first bytes of content since the `Content-Type` HTTP header is not always reliable.

Decompiler: We used the JPEXS Free Flash Decompiler[12] to decompile Flash files and obtain the ActionScript source code. The source code is then searched for fingerprinting related function calls (e.g. `enumerateFonts` and `getFontList`) to obtain a binary occurrence vector. The complete list of methods and properties searched in the decompiled source code is available in Appendix A.2.

Central Database: We ran crawls using several machines, but used a central database to store, combine, and analyze the results of different crawls with minimal effort. The stored data include the set of JavaScript function calls, the list of HTTP requests and responses, and the list of loaded or requested fonts. For the Flash experiments, we also stored a binary vector that represents the existence of ActionScript API calls that might be related to fingerprinting.

3.2 Performance

By using the Dromaeo JavaScript performance test suite[13] we compared our modified Chromium browser against the Chromium browser available from Ubuntu's repositories. Although the versions of Chromium and WebKit were different for the two browsers (Chromium 25 vs 28 and WebKit 537.22 vs

[12] `http://www.free-decompiler.com/flash/`
[13] `http://dromaeo.com/`

537.36) we still believe that the results can provide a rough approximation of the performance of our tool. The resulting scores from the Dromaeo test suites were 3,148.63 runs/s for original Chromium and 3,025.86 runs/s for our modified browser. The difference between their aggregate performance is about 4%, where error intervals were measured as 1.24% and 1.98%.

The most crucial differences were between the `getHeight` and `getWidth` methods of the Prototype JavaScript library[14] (9.76 runs/s vs 214.1 runs/s and 9.79 runs/s vs 244.3 runs/s) which is probably due to the fact that we log accesses to `offsetHeight` and `offsetWidth` properties. But within JQuery's[15] `height` and `width` methods, differences were quite low (0.84 runs/s vs 2.27 runs/s and 0.97 runs/s vs 2.27 runs/s) hinting that this might also be due to the libraries' implementation details[16].

We were able to run 200 parallel PhantomJS instances on a customer grade computer with quad core CPU and 8GB of RAM. With these settings we were able to complete a homepage crawl of the Alexa top million sites in a period of four days.

For the experiments with the Chromium browser, we could run approximately 20 crawlers in parallel, since each crawler was running an instance of Chromium and mitmproxy, as well as was using the Flash decompiler whenever a Flash object was received. Crawling the top 20,000 Alexa sites required about one day using a single computer with the characteristics given above.

4 Font-based analysis of fingerprinting

In order to carry out a large scale study of fingerprinting, we combined automated and manual analysis in our experiments. In the automated step we use font probing and font enumeration as a criteria for identifying candidate sites that possibly include a fingerprinting script or object. This approach is motivated in Section 4.1 and explained in more detail in Section 4.2. We then examine manually the candidates to refine the classification using additional criteria and establish whether the pages contain fingerprinting code.

[14] `http://prototypejs.org/`

[15] `https://jquery.org/`

[16] The full comparison of the performance results can be seen here: `http://dromaeo.com/?id=197597,197598`.

4.1 Motivation

By inspecting the code of known fingerprinters, and taking into account the findings of previous studies [7, 18], we distill *"font detection"* through enumeration or probing as a necessary ingredient of device fingerprinting. Furthermore, we consider font detection to be a good indicator of fingerprinting, as getting the list of system fonts has fewer use cases than requesting information needed for browser feature detection.

The use of font detection for fingerprinting also offers the following advantages:

- according to Panopticlick study, fonts are the second most identifying attribute of a device, having 17.1 bits of entropy when unsorted;
- although the list of installed browser plugins has slightly more entropy, fonts are OS-dependent and thus enable linking different browsers running on the same device;
- the Tor Browser, which is the most widely used tool with counter-fingerprinting features, is shipped with a fixed browser configuration in which plugins are disabled. Fonts are thus the most identifying feature in this context, and the best candidate to successfully identify and track anonymous Tor users.

4.2 Methodology

We follow a two-step analysis. The first step consists of an automated analysis of font detection and the second of a manual analysis of scripts and decompiled Flash source code. Font detection plays a central role in our analysis of the dataset collected by FPDetective: it is used to identify likely fingerprinting candidates in an automated fashion. We consider as likely candidates the websites that request an abnormally large number of fonts, or that have font enumeration calls in the decompiled ActionScript source code. In our analysis we classified as candidates pages that include scripts that load more than 30 fonts, or Flash files that contain font enumeration calls.

Our analysis methodology is different for the identification of Flash and JavaScript-based fingerprinters, given that font detection can be carried out by direct enumeration through Flash, and by indirect probing through JavaScript.

In the case of Flash, a dynamic analysis of font enumeration is hard due to the proprietary nature of the Flash plugin player and the use of internal font caches. At the same time, because of the byte-code nature of Flash objects, they

can be straightforwardly decompiled to their original ActionScript source code. We thus decompiled the Flash objects into their corresponding ActionScript source code, selected the ones that had font enumeration calls, and manually analyzed them to understand what they do with the list of system fonts they collect. We counted a Flash object as a fingerprinter when it enumerates system fonts, collects information about the device capabilities and sends the collected fingerprint back to a remote server either by opening a socket connection or using JavaScript asynchronous calls. Furthermore we eliminated many false positives by checking Flash file URLs and domains with WHOIS lookups. For example, multimedia players that include font enumeration to be used on screen displays were eliminated by using background information.

On the other hand, due to the increased use of code obfuscation and code minimization techniques, it is, in general, very challenging to accurately perform a static analysis of JavaScript source code. We thus opted for a dynamic analysis approach, in which we intercept and record JavaScript font-probing events with the FPDetective's instrumented browsers.

As mentioned earlier, next to font-probing events, FPDetective also collects data related to other events that might be associated with fingerprinting, such as browser plugin enumeration and screen-size detection. The second part of our JavaScript-based fingerprinting analysis consisted of manually studying the identified candidates, taking into account this additional information. Specifically we checked for evidence of browser feature enumeration, including iterating over `name`, `filename`, `description` and `length` properties of browser plugins, the `enabledPlugin`, `description`, `suffixes`, and `type` properties of `mimeType` objects, and `navigator` properties such as `userAgent`, `appCodeName`, `product`, and `productSub`; `screen` object properties such as dimensions, color and pixel depth information; and properties that reveal installed toolbars such as `availTop` and `availLeft`. We classify a JavaScript file as a fingerprinter when it loads more than 30 system fonts, enumerates plugins or mimeTypes, detects screen and navigator properties, and sends the collected data back to a remote server. As we do for the Flash objects, we incorporated background information about JavaScript domain names to eliminate false positives.

We verified the correct functioning of the framework by crawling test pages that request a known set of fonts and checking them against the ones returned by FPDetective.

5 Experiments and Results

In the following experiments we used the FPDetective framework to crawl the top Alexa websites while searching for instances of web-based device fingerprinting. Our experiments were separated into the ones geared towards the discovery of JavaScript-based fingerprinting attempts and the remaining ones towards the discovery of Flash-based fingerprinting.

For each type of experiment, we first automatically searched for font-detection attempts and used that for selecting candidates for manual analysis. In the JavaScript experiments, we tried to find JavaScript-based font probing attempts, for which we used the number of requested fonts as a measure. In order to filter out websites that load high numbers of fonts, but do not use them for probing, we checked if websites measure the width and height of displayed text by using the number of calls to the `offsetWidth` and `offsetHeight` properties of the corresponding HTML elements.

For the Flash experiments we crawled the sites with Chromium and intercepted Flash objects with mitmproxy. We then decompiled the discovered Flash files and searched for a list of ActionScript API calls (see Appendix A.2 for the full list) that might be relevant to fingerprinting. Subsequently, we generated a binary vector that represents the occurrence of each function call in the Flash object and used that to select objects for manual analysis.

5.1 JavaScript-Based Font probing

In this set of experiments, we crawled the top Alexa websites with FPDetective to find out the extent of the JavaScript-based font probing. In the first experiment we visited the homepages of the top Alexa 1 million websites where we waited for 10 seconds, in order to allow for the loading of remote content. In the second experiment, we visited 100,000 websites and clicked 25 homepage links from the same domain. We waited 5 seconds after each click and 10 seconds after each page load to allow for resources to load.

By analyzing the sites that are sorted by FPDetective as likely candidates of fingerprinting, we found 13 instances of JavaScript-based font-probing scripts, on a total of 404 websites. In order to ensure the accuracy of our classification, we augmented the automated, dynamic analysis with manual analysis of the source

			Number of sites using JS-based fingerprinting		
			1M	100K	
Fingerprinting Provider	**Num Fonts**	**Top Rank**	**In homepage**	**In homepage**	**In inner pages**
BlueCava	231/167/62	1,390	250	24	24
Perferencement	153	49,979	51	6	6
CoinBase	206	497	28	4	4
MaxMind	94	498	24	5	5
Inside graph	355	98,786	18	1	1
SiteBlackBox	389	1,687	14	10	10
Analytics-engine	98	36,161	6	-	-
Myfreecams	71	422	3	1	1
Mindshare Tech.	487	109,798	3	-	-
Cdn.net	297	501,583	3	-	-
AFK Media	503	199,319	2	-	-
Anonymizer	80	118,504	1	-	-
Analyticsengine	93	522,447	1	-	-
			404	51	51

Table 1: Prevalence of Fingerprinting with JavaScript Based Font Probing on Top 1M Alexa sites

code and background information on the companies that own the domains from where scripts are served. Specifically, we checked whether the script (or another script served from the same domain) collects other high-entropy browser properties such as plugins or mimeTypes and if the company is involved in products and services that might be related to fingerprinting, e.g., device identification, analytics and anti-fraud.

Table 1 shows the results of these experiments in detail. The discovered scripts probed as many as 503 fonts and, popularity-wise, there were three websites in the top 500 Alexa sites that made use of a font-probing script.

BlueCava was discovered on the homepages of 250 Alexa websites, making it, by far, the most popular font-probing script. Moreover, it is the only one of the discovered font-probing scripts that queries different sets of fonts based on the device's operating system: 231 fonts for Microsoft Windows, 167 for Mac OS

and 62 for other operating systems.

While analyzing the candidate scripts, we came across some interesting practices, such as a script that first dynamically injects itself into the page and then, after collecting the device fingerprint, removes itself from the page. This was one of the cases that motivated us to develop our tools for dynamic analysis and is discussed in more detail in Section 7.2.

Most of, but not all, the scripts that we found are served from a domain registered for an analytics and/or anti-fraud company. An intriguing case is that of Anonymizer[17], a paid anonymization service that fingerprints every visitor accessing their homepage and has a set of fingerprinting scripts that include function names such as `submitDnsInfoViaAjax`, `getClockSkew`, `getJsFontList` and `connectViaSocket`.

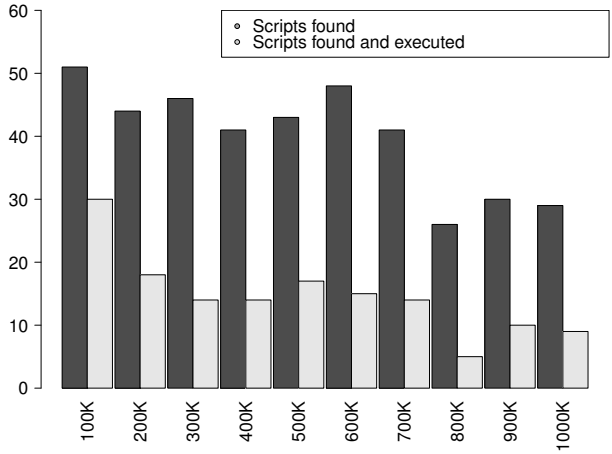


Figure 2: JavaScript-based font probing scripts on homepages of Top 1 Million Alexa sites

We also found that including third party scripts is not the only form of distribution, e.g., the font probing script from CoinBase is found as an embedded button that visitors might click to donate BitCoins to site owner.

We compared the sites that fingerprint users on the homepage, to sites that only fingerprint users on the inner pages. We did not observe any differences between these two, as evidenced by the two last columns of Table 1. This could mean that either we could not crawl the inner pages that include fingerprinting

[17] `https://www.anonymizer.com/`

Table 2: Flash Fingerprinting objects found on Top 10K Alexa websites

Fingerprinting Provider	No. of Sites	Top Rank	Flash Cookies	Proxy Detection	HW and OS Profiling	Interaction with JS
BB Elements	69	903	✓			✓
Piano Media	12	3,532	✓		✓	✓
Bluecava	6	1,390	✓			✓
ThreatMetrix	6	2,793		✓	✓	
Alipay	1	83	✓		✓	✓
meb.gov.tr (Turkish Ministry of Education)	1	2,206	✓		✓	✓

scripts not present in the homepage, or sites prefer to fingerprint users on their homepages.

The histogram in Figure 2 represents the distribution of sites using JavaScript-based font probing among the Top 1 Million Alexa sites. The histogram is divided in intervals of 100,000 sites (according to their Alexa rank) which are further divided into two bins. The darker bin refers to the total number of sites in that popularity range that served a fingerprinting script. These scripts are counted by matching previously discovered script URLs or regular expressions to URLs of HTTP requests made while visiting the site. Since we found that not all scripts probe a large number of fonts every time they are loaded, we included a second (lighter) bin to represent sites that both served a fingerprinting script and probed for fonts.

5.2 Flash Based Font Enumeration

The main objective of this experiment is to investigate the extent of Flash based fingerprinting techniques on the Web. We crawled the top 10 thousand Alexa websites using FPDetective and, for each site, we visited the homepage and waited for 10 seconds to allow for resources to load completely.

We automatically decompiled all the Flash files caught by the intercepting proxy and checked the presence of functions that might be used for fingerprinting. We manually analyzed the files that include functions for known Flash-based fingerprinting techniques [18], such as querying the `Capabilities` class to

collect information about the operating system and device, or opening a socket (by using `XMLSocket`) to bypass system wide proxy settings.

We also checked for the presence of function calls that might be used to send the collected information to a remote server (e.g., `sendAndLoad` or `URLLoader`) or use of `ExternalInterface.Call` or `addCallback` functions for opening JavaScript interfaces to either call JavaScript functions from the Flash file or allow page scripts to access Flash functions.

Finally, we took into account whether the collected information is sent to a remote server or used internally, by the Flash object. This helped us to filter out potential false positives, such as Flash video players that enumerate system fonts, in order to check if a font is available for use in player's on-screen display.

Table 2 shows our results for Flash-based fingerprinting. Being found on 69 sites, BB Elements, which offers tools for managing ad campaigns, is the most prevalent Flash-based fingerprinter. Piano Media is mostly found on newspaper sites and claims to employ fingerprinting for paywall enforcement. Paywalls, in this case, are used to ensure that users do not access paid content, such as newspaper or magazine articles, without subscription or payment. Usually, paywalls are combined with a limited number of free accesses, e.g., an online newspaper can offer ten free articles per user per day.

The website of the Turkish Ministry of Education (MEB) is unique in both being the only `.gov` website that employs web-based device fingerprinting and in the wealth of collected attributes from the device's hardware components. The Flash file served from this site extracts and sends detailed information about system mouse, keyboard, accelerometer, multi-touch capability, microphone, camera as well as system fonts. It should be noted that this site serves an e-School system to manage grades and other records of millions of students in Turkey. ThreatMetrix is the only discovered company which uses proxy-piercing to reveal a user's original IP address and whether she uses a proxy.

In Table 2 we can see the fingerprinters that have been discovered along with some of the functions that were called from the Flash files. We only included in the table the Flash files that have functions for font enumeration. That does not include, for instance, 50 Flash objects from one of the companies previously studied [18].

Given that we discovered several previously unknown fingerprinting scripts by extending our crawl space to 1 million for the JavaScript experiment, we plan to make a similar study for the Flash based fingerprinters.

5.3 Testing FPDetective with Fontconfig

Fontconfig is a Linux library that configures and manages the access to system fonts. In this experiment we used debugging functionality of fontconfig to test our framework's accuracy.

We crawled the Alexa Top 10,000 sites and compared the set of font requests made by each web site to ensure that FPDetective is not missing or overcounting font events. The comparison of the two measurements indicated that we neither miss nor overcount the font load events.

5.4 Prevalence of Fingerprinting

With FPDetective we found 404 sites in the Alexa top million pages that fingerprint visitors on their homepages using JavaScript-based font probing. These scripts are served by 13 different fingerprinting providers, of which only a few had been identified in prior research.

Although 404 out of 1,000,000 can be thought of as a very low frequency, we would like to note that the results presented here should be taken as lower bounds, as our crawlers cannot reach pages that are placed after forms including CAPTCHAs or similar obstacles. Moreover, Flash-based fingerprinting was present on the homepages of 95 out of the top 10,000 sites, indicating that Flash-based fingerprinting is more prevalent. This is possibly because of its extended capabilities for font enumeration, proxy detection and its widespread browser support.

6 Evaluation of Fingerprinting Countermeasures

In this section we briefly analyze two tools that can be employed to counter fingerprinting: the Tor Browser and the FireGloves Firefox extension. We also evaluate whether the Do Not Track header is being respected by fingerprinters.

6.1 Tor Browser

The Tor Browser is part of the software bundle that is used to access the Tor anonymity network [5], a popular service that is currently used daily by more than 800,000 people to anonymously browse the web. Tor relays communications over three routers located in different parts of the world. The communications

are encrypted in layers (onion routing [8]) to prevent any single Tor router from linking the source and destination of a data stream.

The Tor Browser incorporates strong defenses to counter the fingerprinting techniques described by the Panopticlick study, as its design goals include enabling user anonymity and preventing the linkability of browsing sessions. Given that the Tor Browser has a limited user base compared to web users in general, even a partial fingerprint might be enough to uniquely distinguish a Tor user. Thus, there is a need to eliminate fingerprintable differences among Tor Browsers, so that Tor users remain indistinguishable from each other. In fact, *cross-origin fingerprinting unlinkability* is listed as a privacy requirement for the Tor Browser[18]. For this reason the Tor Browser is shipped with fixed settings that provide almost no browser properties that could be exploited to extract distinguishing features.

As described in Section 4, fonts are operating-system-dependent, and thus a good candidate attribute for distinguishing and recognizing users. To limit font-based fingerprinting, the Tor Browser caps the number of fonts that a page can request and load. However, for usability reasons, `@font-face` rules are exempted from these limits. Upon inspection of the Tor Browser source code, we discovered that the local fonts loaded by `@font-face` CSS rules are also exempted from the Tor Browser's font-per-document cap, and that it is possible to load an unlimited number of system fonts using the `local()` value of the `@font-face` rule's `src` descriptor.

Furthermore, if a font is not installed locally, it may be requested from the `src/url` property of the font-face rule. This is effectively communicating the lack of this font to the server without using JavaScript. Note also that one local font-face rule can be used to report the `status(not-found)` of more than one font by chaining font-face `local()` properties, for example:

```
@font-face { font-family: Font1;
             src: local("Font2"), local("Font3"),
                  url("Font1-2-3-NotFound.ttf");
           }
```

We immediately communicated the vulnerability to the Tor Bug Tracker and the issue is fixed in the upcoming (2.4) version with a patch that disables the use of the `local()` property[19].

[18] `https://www.torproject.org/projects/torbrowser/design/#fingerprinting-linkability`

[19] `https://trac.torproject.org/projects/tor/ticket/8455#comment:3`

6.2 Firegloves

Firegloves [3] is a proof-of-concept browser extension for Mozilla Firefox that was created for research purposes. In order to confuse fingerprinting scripts, Firegloves returns randomized values when queried for certain attributes, like the screen resolution, the platform on which the browser is running and the browser's vendor and version. Additionally, Firegloves limits the number of fonts that a single browser tab can load and reports false dimension values for the `offsetWidth` and `offsetHeight` properties of HTML elements to evade JavaScript-based font detection.

We evaluated the effectiveness of Firegloves' as a countermeasure to fingerprinting, and discovered several shortcomings. For instance, instead of relying on `offsetWidth` and `offsetHeight` values, we could easily use the width and the height of the rectangle object returned by `getBoundingClientRect` method, which returns the text's dimensions, even more precisely than the original methods[20]. This enabled us to detect the same list of fonts as we would without the Firegloves extension installed. Surprisingly, our probe for fonts was not limited by the claimed cap on the number of fonts per tab. This might be due to a bug, or to changes in the Firefox extension system that have been introduced after FireGloves, which is not currently being maintained, was first developed.

Although Firegloves spoofs the browser's user-agent and platform to pretend to be a Mozilla Firefox version 6 running on a Windows operating system, the `navigator.oscpu` is left unmodified, revealing the true platform. Moreover, Firegloves did not remove any of the new methods introduced in later versions of Mozilla Firefox and available in the `navigator` object, such as `navigator.mozCameras` and `navigator.doNotTrack`. Finally, since Firegloves cannot change the APIs available to Flash, a Flash application can still discover the real operating system and the real screen resolution.

Overall, while Firegloves is trying to protect users from fingerprinting, its detectable presence on the users' browsing environments may actually make them more uniquely identifiable: since Firegloves is installed by only $1,750$ users, its successful detection makes the user much more unique than if it was not present at all.

Our findings are in line with prior results showing that user-agent-spoofing extensions can be straightforwardly discovered and bypassed [18]. These findings illustrate the difficulty of effectively protecting users against fingerprinting, and indicate that counter-measures short of perfect may result in a net loss of privacy for their adopters – as their devices become more easily fingerprintable through

[20] `https://developer.mozilla.org/en-US/docs/DOM/element.offsetWidth`

these.

6.3 Do Not Track

The Do Not Track (DNT) HTTP header field allows users to signal their tracking preferences to websites. DNT is currently being standardized by the W3C under the name "Tracking Preference Expression" and it has already been adopted by most modern browsers[21].

We set the DNT header to 1 in the PhantomJS browser and visited the websites identified as performing fingerprinting in our previous experiments. For all of these pages, we obtained the same results with respect to the number of fonts probed and other browser properties accessed, suggesting that DNT preferences are ignored by fingerprinters.

7 Discussion

7.1 Uses of Fingerprinting

Using FPDetective, we were able to identify companies that are engaging in what we call web based device fingerprinting. While it is not possible to infer the purposes for which fingerprinting is being put to use by these companies, we take a moment to reflect on their fingerprinting related practices.

The majority of the companies serving the fingerprinting scripts explicitly state on their websites, in press releases, and in various social media channels that they are successfully deploying (device) fingerprinting. In contrast, the websites that employ the services of these companies rarely disclose in their privacy policies that they are fingerprinting their users' devices, let alone mention that they are gathering the information that FPDetective revealed them to be collecting. In some cases, we detected fingerprinting scripts that were embedded in ad banners. It is unclear whether first parties serving these ad banners are aware of the existence of these embedded fingerprinting scripts.

Companies express that they deploy device fingerprinting in the context of a variety of web services. The spectrum of use cases include fraud detection, protection against account hijacking, anti-bot and anti-scraping services, enterprise security management, protection against DDOS attacks, real-time targeted marketing, campaign measurement, reaching customers across devices,

[21] `http://www.w3.org/TR/2013/WD-tracking-dnt-20130430/`

and limiting number of access to services. While most companies will specialize either in 'fraud detection and web security' or 'marketing applications', MaxMind is an example of a company that uses their data collection to provide services for both uses. MaxMind also stands out with their very explicit documentation of their personal and non-personal data collection, their processing activities and their commitment to make their practices transparent.

Further material from companies describing the use cases suggest that the data models in the databases also vary. Fraud detection companies often speak of "device reputation databases" with profiles for "billions" of devices that are rich enough to provide "intelligence" to companies about the security risks of these devices. The variety in mentioned uses ranging from "adjusting security policies based on the device a person is using" to "identifying and blocking botnets that easily switch IP addresses" suggests that these are rich device-centered databases. On the other hand, marketing companies express that they are able to identify user-behavior across websites and devices, suggesting that they feed fingerprinting data into "customer"-centered database systems that are heavy on analytics, and can link different devices to a single user across websites.

In-house applications seem to be mainly concerned with whether the same user is using multiple devices to access the same service, or limiting the number of times a specific service is accessed, e.g., when filling out surveys. Such applications seem to limit data collection using fingerprinting to the given website and are less concerned with analytics and intelligence applications.

A perplexing case among in-house applications is that of Anonymizer Inc., a company that presents itself as "the global leader in online privacy, anonymity, and identity protection solutions for over 17 years."[22] The script is served from the domain privacytool.org (owned by Anonymizer Inc.), a site where users can test whether they are anonymous online. The privacytool.org site clearly explains that to perform the test a Java applet will run on the user's computer, describes the information that will be gathered, and explicitly states that *"Data obtained from the browser like lists of plug-ins or fonts can be used to identify your computer."*[23] Users must click a link placed below this information in order to run the applet.

We found, however, that fingerprinting scripts from privacytool.org are also present in the homepage of anonymizer.com, another site owned by Anonymizer Inc. that, paradoxically, offers anonymity as a service. The anonymizer.com privacy policy states: *"When you navigate our Web site, Anonymizer will gather certain information such as your Internet Protocol address, browser type, browser language, and the date and time of your visit. We may place a cookie*

[22] `https://www.anonymizer.com/company/`
[23] `http://privacytool.org/AnonymityChecker/index.jsp`

on your computer [...]" The policy includes further information about cookies but, in contrast to the privacytool.org notice, it does not mention at any point the execution of fingerprinting scripts, or that it collects information such as the list of installed fonts, DNS server information, or the real IP address if the user is connected through a proxy. Finally, note that while privacytool.org offers informed choice to its users, who may voluntarily execute the script, the fingerprinting scripts that run in the anonymizer.com homepage are invisible to users and run by default.

7.2 Visibility of Fingerprinting

The majority of web users have difficulties in grasping what cookies are, whether they are enabled, their threats to their privacy and how to manage them [14]. This situation is worse in the case of third-party and covert cookies, which have been found to essentially be invisible to end-users. Arguably, user perception of web based device fingerprinting is comparably, if not more, invisible to the users.

In all the cases we encountered, there were no visible effects of fingerprinting, and the users were not informed that they were being fingerprinted. Thus, the only way for users to discover that their devices are being fingerprinted is to manually examine the source of the page and all the embedded JavaScript and Flash objects.

In one distinct case, while verifying the results of FPDetective, we discovered that a company which sells fingerprinting products for anti-bot and anti-scraping services, was deleting the fingerprinting script from the page's DOM after the script had executed and collected the fingerprint. Thus, the only way to identify this fingerprinter is to breakpoint through the execution of JavaScript code and witness the loading and unloading of the fingerprinting code. This requires an in-depth understanding of how JavaScript is executed and the principles of debugging programs, which the vast majority of users are not likely to possess. Moreover, this anti-debugging technique is reminiscent of the techniques employed by JavaScript malware when trying to evade detection by analysts and high-interaction honeypots [10].

7.3 Is fingerprinting a matter of privacy?

Our findings suggest that this issue may require further technical and legal attention with respect to privacy. Yet, most of the companies whose

fingerprinting activities were detected by FPDetective expressly distance their practices from any consequences they might have for people's privacy.

A number of companies argue that they do *not* collect PII (Personally Identifiable Information) in the process of distinguishing "the good, the bad and the ugly" on the web. While fingerprinting may not require PII, the use cases described by fingerprinting companies on their webpages suggest that they use device information to track, profile, and shape the future web experience of the tracked users (as well as bots). For example, MaxMind offers online retailers a service to check on their customers' fraud scores based on 31 "non-PII" attributes, including IP address, shipping address, non-salted hashes of email addresses and passwords, and credit card BIN number. The computation of the customers' score is based on things like the 'riskiness of the country of origin', 'proxy use', 'free webmail use', and 'bank checks'. These are matters closely related to privacy concerns expressed by users about uninformed monitoring of web usage, constraints on informational self-determination, and discrimination [21].

Further, by focusing on "device" identification and, especially in fraud detection cases, claiming that they are concerned only about "bots", companies express that tracking "persons" is not the object of their interest. In this worldview, fingerprinting is nothing more than (security) scripts collecting data based on socially invisible interactions that are irrelevant to individual privacy. Further, especially in the case of fraud detection, companies often argue that fingerprinting is implemented for the protection of the end-users' quality of service. These two framings, i.e., "fingerprinting is all about devices" and "we track these devices for user convenience", make it very difficult to demand a response to the privacy issues that may be raised with respect to device fingerprinting and the use of the databases populated using fingerprinting. As such fingerprinting practices proliferate, device IDs may come to "represent" users in databases, instead of PII. Hence, classical conceptions of PII may not be sufficient to grasp the social and ethical concerns associated fingerprinting and related databases. We hope that this paper, by virtue of making web based fingerprinting more visible, will contribute to better understanding what privacy issues may be at stake and to challenging the framing of web based device fingerprints as non-PII.

In the context of the US, it may be worth discussing whether a static list of attributes that count as PII is sufficient to draw a reasonable boundary on which "personal data" should be subject to protection. Device fingerprinting underlines that data is identifiable based on context; in other words, identifiability may result from processing seemingly non-identifiable information [22]. If we accept this argument, then attention needs to be paid to risks associated with: linking of the reputation and device fingerprint databases to individuals; undesirable and unacceptable uses of these datasets for determining "the good, the bad,

and the ugly" of the web; the security of these datasets; and the opacity of fingerprinting practices to the general public, as well as to individual device owners.

It is also currently unclear whether there is a responsibility to inform the owners that their devices are being fingerprinted, and if so, who has the responsibility to inform the users. For example, the privacy policy of `articlesbase.com` explicitly indicates all the "non-personal information" that the site collects. However, in this rather detailed and readable list, the site does not state that they are fingerprinting the user's device or that they are probing fonts. One could argue, since the font probing scripts are not served by `articlesbase.com` but by `siteblackbox.com`, that it is the responsibility of the latter to inform the users. However, in their documentation of the articlesbase case [20], no references are made to fingerprinting of users and we were unable to locate a privacy policy on the website. Similar issues are likely to arise when it comes to honoring DNT preferences.

8 Conclusion

User tracking is becoming pervasive as advertisers and tracking companies seek to refine their targeting, detect fraud, or offer new services. While most of today's tracking is done through third-party cookies, prior research has shown that browser and system attributes can be used to uniquely identify devices through fingerprints. Even though these fingerprints are less accurate than stateful identifiers such as cookies, their main advantage is that device fingerprinting is harder to detect and to defend against.

In this paper we presented FPDetective, a fingerprinting-detection framework that identifies web based fingerprinters. Using FPDetective, we performed a large-scale crawl of the Internet's most popular websites, and showed that the adoption of fingerprinting is significantly higher than what previous research estimated. Among others, we identified large commercial companies involved in fingerprinting, a complete disregard towards the DNT header, and the use of anti-debugging techniques, most commonly associated with JavaScript malware. Moreover, we showed that dedicated fingerprinters can bypass existing privacy-protecting technologies.

Overall, our findings demonstrate that web fingerprinting is a real and growing issue, deserving the attention of both policymakers and the research community. We hope that our framework, which is freely available to other researchers and can easily be extended to conduct further studies, will contribute to addressing

this issue by providing a means to shed light on web fingerprinting practices and techniques.

9 Acknowledgements

The authors would like thank the anonymous reviewers, Carmela Troncoso, Ashkan Soltani, Nessim Kisserli and Tom van Cutsem for their valuable comments; Vicenç Torra for enabling the collaboration; and Danny De Cock for his hardware support. For KU Leuven, this research was supported by the projects IWT SBO SPION, iMinds CoMobile, FWO G.0360.11N, FWO G.0686.11N, GOA TENSE (GOA/11/007), and EU FP7 NESSoS, WebSand and Strews; the B-CCENTRE; and the Research Fund KU Leuven. For IIIA-CSIC, this research was supported by the projects EU FP7 grant agreement number 262608, the Spanish MEC projects ARES (CONSOLIDER INGENIO 2010 CSD2007-00004), eAEGIS (TSI2007-65406-C03-02), and COPRIVACY (TIN2011-27076-C03-03). For NYU, this research was supported by the Intel Science and Technology Center - Social Computing. Marc Juarez was partially funded by the LLP Erasmus Programme of the Commission of the European Communities during his stay at KU Leuven.

References

[1] Nmap - Free Security Scanner For Network Exploration & Security Audits. `http://www.nmap.org`.

[2] Adam Barth. Blink: A rendering engine for the Chromium project. `http://blog.chromium.org/2013/04/blink-rendering-engine-for-chromium.html`.

[3] Karoly Boda. Firegloves. `http://fingerprint.pet-portal.eu/?menu=6`.

[4] Aldo Cortesi. mitmproxy: a man-in-the-middle proxy. `http://mitmproxy.org/`.

[5] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320. USENIX, 2004.

[6] John R Douceur. The sybil attack. In *Peer-to-peer Systems*, pages 251–260. Springer, 2002.

[7] Peter Eckersley. How unique is your web browser? In *Privacy Enhancing Technologies*, pages 1–18. Springer, 2010.

[8] David Goldschlag, Michael Reed, and Paul Syverson. Hiding routing information. In *Information Hiding*, pages 137–150, 1996.

[9] Dongseok Jang, Ranjit Jhala, Sorin Lerner, and Hovav Shacham. An empirical study of privacy-violating information flows in JavaScript Web applications. In *Proceedings of CCS 2010*, pages 270–283, October 2010.

[10] Alexandros Kapravelos, Marco Cova, Christopher Kruegel, and Giovanni Vigna. Escape from monkey island: Evading high-interaction honeyclients. In *Proceedings of the 8th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, pages 124–143, 2011.

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

[12] Jonathan R. Mayer. Any person... a pamphleteer. Senior Thesis, Stanford University, 2009.

[13] Jonathan R. Mayer and John C. Mitchell. Third-party web tracking: Policy and technology. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 413–427, 2012.

[14] Anthony D Miyazaki. Online privacy and the disclosure of cookie use: Effects on consumer trust and anticipated patronage. *Journal of Public Policy & Marketing*, 27(1):19–33, 2008.

[15] Keaton Mowery, Dillon Bogenreif, Scott Yilek, and Hovav Shacham. Fingerprinting information in JavaScript implementations. In Helen Wang, editor, *Proceedings of W2SP 2011*. IEEE Computer Society, May 2011.

[16] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in HTML5. In Matt Fredrikson, editor, *Proceedings of W2SP 2012*. IEEE Computer Society, May 2012.

[17] Martin Mulazzani, Philipp Reschl, Markus Huber, Manuel Leithner, Sebastian Schrittwieser, and Edgar R. Weippl. Fast and reliable browser identification with javascript engine fingerprinting. In *Web 2.0 Workshop on Security and Privacy (W2SP)*, May 2013.

[18] Nick Nikiforakis, Alexandros Kapravelos, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. Cookieless monster: Exploring the ecosystem of web-based device fingerprinting. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 541–555, 2013.

[19] Łukasz Olejnik, Claude Castelluccia, and Artur Janc. Why Johnny Can't Browse in Peace: On the Uniqueness of Web Browsing History Patterns. In *the 5th workshop on Hot Topics in Privacy Enhancing Technologies (HOTPETS 2012)*.

[20] SiteBlackBox. Case studies: Articlesbase. `http://www.siteblackbox.com/case-studies/articles-base.php`.

[21] Blase Ur, Pedro Giovanni Leon, Lorrie Faith Cranor, Richard Shay, and Yang Wang. Smart, useful, scary, creepy: perceptions of online behavioral advertising. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 4:1–4:15, New York, NY, USA, 2012. ACM.

[22] Irma Van Der Ploeg. *Keys To Privacy. Translations of "the privacy problem" in Information Technologies*, pages 15–36. Maastricht: Shaker, 2005.

[23] Ting-Fang Yen, Yinglian Xie, Fang Yu, Roger Peng Yu, and Martin Abadi. Host fingerprinting and tracking on the web:privacy and security implications. In *Proceddings of the 19th Annual Network and Distributed System Security Symposium (NDSS)*, 2012.

[24] Nicholas C. Zakas. How many users have JavaScript disabled? `http://developer.yahoo.com/blogs/ydn/many-users-javascript-disabled-14121.html`.

[25] Michal Zalewski. p0f v3 (version 3.06b). `http://lcamtuf.coredump.cx/p0f3/`.

A Appendices

A.1 List of Fingerprinting URLs

Table 3 shows the location of each fingerprinting script discovered, separated by the respective fingerprinting companies.

A.2 ActionScript Calls

The following list enumerates the ActionScript calls that FPDetective searches for in the decompiled Flash files.

- `enumerateFonts` (only with argument true)

FingerPrinter	Script URL
BlueCava	http://ds.bluecava.com/v50/AC/BCAC5.js
Perferencement	http://tags.master-perf-tools.com/V20test/tagv22.pkmin.js
CoinBase	https://coinbase.com/assets/application-773afba0b6ee06b45ba4363a99637610.js
MaxMind	http://device.maxmind.com/js/device.js
Inside graph	http://inside-graph.com/ig.js
SiteBlackBox	(No fixed URL)
Analytics-engine	http://sl4.analytics-engine.net/detector/fp.js
Myfreecams	http://www.myfreecams.com/mfc2/lib/o-mfccore.js
Mindshare Tech.	pomegranate.js (No fixed domain)
Cdn.net	http://cdn-net.com/cc.js
AFK Media	http://gmyze.com/0.4.1.1/js/fingerprint.js
Anonymizer	https://www.privacytool.org/AnonymityChecker/js/fontdetect.js
Analyticsengine	http://dpp750yjcl65g.cloudfront.net/analyticsengine/util/fingerprint.compiled.js
BBelements	http://go.eu.bbelements.com/flash/bbnaut.swf
Piano Media	http://mp.pianomedia.eu/bucket/novosense.swf
Bluecava	http://lookup.bluecava.com/flash/guids[2|3].swf
ThreatMetrix	https://h.online-metrix.net/fp/fp.swf?org_id=...&session_id=... *
Alipay	http://img.alipay.com/common/um/lsa.swf
MEB	http://meb.gov.tr/KZneA1akxW/502758.swf

Table 3: URLs of Fingerprinting JavaScript and Flash Files
*: Also served from other domains.

- `getFontList`
- all `Capabilities` class properties and methods, including
  - `version`
  - `manufacturer`
  - `serverString`
  - `language`
- `screenDPI`
- `screenResolutionX`
- `screenResolutionY`
- `getTimezoneOffset`
- `getLocal`
- `XMLSocket`
- `Math.min, Math.max`

- `ExternalInterface.call`
- `ExternalInterface.addCallback`
- `sendAndLoad`
- `URLLoader`
- `navigateToURL`
- `loadMovie`
- `createUID`
- `getUrl`
- `allowDomain`
- `allowInsecureDomain`
- `loadPolicyFile`
- `URLRequest`
- `LoadVars`
- `md5, sha256, sha384, sha512`

Publication

The Web Never Forgets: Persistent Tracking Mechanisms in the Wild

Publication Data

Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., and Diaz, C. The Web never forgets: Persistent tracking mechanisms in the wild. In *21st ACM Conference on Computer and Communications Security (CCS)* (2014), ACM, pp. 674–689

Contributions

- Principal author.

The Web Never Forgets: Persistent Tracking Mechanisms in the Wild

Gunes Acar[1], Christian Eubank[2], Steven Englehardt[2], Marc Juarez[1], Arvind Narayanan[2], and Claudia Diaz[1]

[1] KU Leuven ESAT/COSIC, iMinds, Leuven, Belgium
{name.surname}@esat.kuleuven.be
[2] Princeton University
{cge,ste,arvindn}@cs.princeton.edu

Abstract. We present the first large-scale studies of three advanced web tracking mechanisms — canvas fingerprinting, evercookies and use of "cookie syncing" in conjunction with evercookies. Canvas fingerprinting, a recently developed form of browser fingerprinting, has not previously been reported in the wild; our results show that over 5% of the top 100,000 websites employ it. We then present the first automated study of evercookies and respawning and the discovery of a new evercookie vector, IndexedDB. Turning to cookie syncing, we present novel techniques for detection and analysing ID flows and we quantify the amplification of privacy-intrusive tracking practices due to cookie syncing.

Our evaluation of the defensive techniques used by privacy-aware users finds that there exist subtle pitfalls — such as failing to clear state on multiple browsers at once — in which a single lapse in judgement can shatter privacy defenses. This suggests that even sophisticated users face great difficulties in evading tracking techniques.

1 Introduction

A 1999 New York Times article called cookies comprehensive privacy invaders and described them as "surveillance files that many marketers implant in the personal computers of people." Ten years later, the stealth and sophistication of tracking techniques had advanced to the point that Edward Felten wrote "If

You're Going to Track Me, Please Use Cookies" [18]. Indeed, online tracking has often been described as an "arms race" [47], and in this work we study the latest advances in that race.

The tracking mechanisms we study are advanced in that they are hard to control, hard to detect and resilient to blocking or removing. *Canvas fingerprinting* uses the browser's Canvas API to draw invisible images and extract a persistent, long-term fingerprint without the user's knowledge. There doesn't appear to be a way to automatically block canvas fingerprinting without false positives that block legitimate functionality; even a partial fix requires a browser source-code patch [40]. *Evercookies* actively circumvent users' deliberate attempts to start with a fresh profile by abusing different browser storage mechanisms to restore removed cookies. *Cookie syncing*, a workaround to the Same-Origin Policy, allows different trackers to share user identifiers with each other. Besides being hard to detect, cookie syncing enables back-end server-to-server data merges hidden from public view.

Our goal is to improve transparency of web tracking in general and advanced tracking techniques in particular. We hope that our techniques and results will lead to better defenses, increased accountability for companies deploying exotic tracking techniques and an invigorated and informed public and regulatory debate on increasingly persistent tracking techniques.

While conducting our measurements, we aimed to automate all possible data collection and analysis steps. This improved the scalability of our crawlers and allowed us to analyze 100,000 sites for fingerprinting experiments, as well as significantly improve upon the scale and sophistication of the prior work on respawning, evercookies and cookie syncing.

1.1 Contributions

First study of real-world canvas fingerprinting practices. We present the results of previously unreported canvas fingerprinting scripts as found on the top 100,000 Alexa sites. We find canvas fingerprinting to be the most common fingerprinting method ever studied, with more than 5% prevalence. Analysis of the real-world scripts revealed that they went beyond the techniques suggested by the academic research community (Section 3).

Automated analysis of evercookies and respawning. We describe an automated detection method for evercookies and cookie respawning. Applying this analysis, we detected respawning by Flash cookies on 10 of the 200 most popular sites and found 33 different Flash cookies were used to respawn over 175 HTTP cookies on 107 of the top 10,000 sites. We also uncover a new

evercookie vector, IndexedDB, which was never found in the wild before (Section 4). Remarkably, respawning has already led to a lawsuit and a $500,000 settlement [14], and yet it is quite prevalent on the web.

Cookie syncing privacy analysis. We find instances of syncing of respawned IDs in the wild, i.e., an ID respawned by one domain is passed to another domain. Respawning enables trackers to link a user's browsing logs before cookie clearing to browsing logs after cookie clearing. In our measurements, approximately 1.4% of a user's browser history can be linked this way in the wild. However, the figure jumps to at least 11% when these respawned cookies are subsequently synced. Cookie syncing also allows trackers to merge records on individual users, although this merging cannot be observed via the browser. Our measurements in Section 5 show that in the model of back-end merging we study, the number of trackers that can obtain a sizable fraction (40%) of a user's browsing history increases from 0.3% to 22.1%.

Novel techniques. In performing the above experiments, we developed and utilized novel analysis and data collection techniques that can be used in similar web privacy studies.

- Using the *strace* debugging tool for low-level monitoring of the browser and the Flash plugin player (Section 4.2).
- A set of criteria for distinguishing and extracting pseudonymous identifiers from traditional storage vectors, such as cookies, as well as other vectors such as Flash storage. By extracting known IDs, we can track them as they spread to multiple domains through cookie syncing.

Making the code and the data public. We intend to publicly release all the code we developed for our experiments and all collected data, including (i) our crawling infrastructure, (ii) modules for analysing browser profile data and (iii) crawl databases collected in the course of this study.

1.2 Implications

The thrust of our results is that the three advanced tracking mechanisms we studied are present in the wild and some of them are rather prevalent. As we elaborate on in Section 6.1, they are hard to block, especially without loss of content or functionality, and once some tracking has happened, it is hard to start from a truly clean profile. A frequent argument in online privacy debates is that individuals should "take control" of their own privacy online. Our results suggest that even sophisticated users may not be able to do so without significant trade-offs.

We show that cookie syncing can greatly amplify privacy breaches through server-to-server communication. While web privacy measurement has helped illuminate many privacy breaches online, server-to-server communication is not directly observable. All of this argues that greater oversight over online tracking is becoming ever more necessary.

Our results only apply to desktop browsing. Studying similar tracking mechanisms on mobile platforms requires distinct methodologies and infrastructure and is left to future work.

2 Background and Related Work

The tracking mechanisms studied in this paper can be differentiated from their conventional counterparts by their potential to circumvent users' tracking preferences, being hard to discover and resilient to removal. We selected three of the most prominent persistent tracking techniques — canvas fingerprinting, evercookies and cookie syncing — based on the lack of adequate or comprehensive empirical measurements of these mechanisms in the wild. We now give a brief overview of these techniques.

Canvas fingerprinting: Canvas fingerprinting is a type of browser or device fingerprinting technique that was first presented in a paper by Mowery and Shacham in 2012 [32]. The authors found that by using the Canvas API of modern browsers, an adversary can exploit subtle differences in the rendering of the same text or WebGL scenes to extract a consistent fingerprint that can easily be obtained in a fraction of a second without user's awareness.

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 [32]. 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.

The entropy available in canvas fingerprints has never been measured in a large-scale published study like Panopticlick [16]. Mowery and Shacham collected canvas fingerprints from 294 Mechanical Turk users and computed 5.73 bits of entropy for their dataset. Since this experiment was significantly limited for

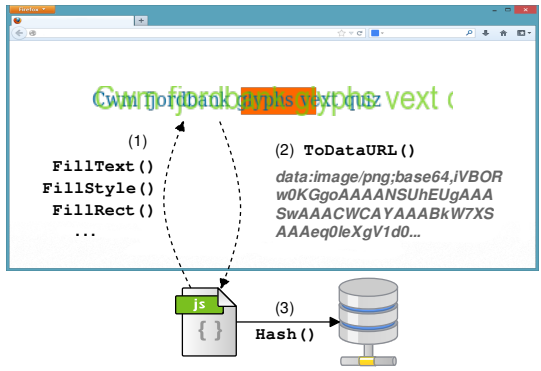


Figure 1: Canvas fingerprinting basic flow of operations

measuring the canvas fingerprinting entropy, they had a further estimate of at least 10 bits, meaning one in a thousand users share the same fingerprint [32].

Figure 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 basically 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 [16].

Evercookies and respawning: A 2009 study by Soltani et al. showed the abuse of Flash cookies for regenerating previously removed HTTP cookies, a technique referred to as "respawning" [43]. 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 [7].

In 2010, Samy Kamkar demonstrated the "Evercookie," a resilient tracking mechanism that utilizes multiple storage vectors including Flash cookies, localStorage, sessionStorage and ETags [21]. 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. Instead of just respawning HTTP cookies by Flash cookies, his script would check the cleared vectors in the background and respawn from any storage that persists.

Figure 2: 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.

Figure 2 depicts the stages of respawning by Local Shared Objects (LSOs), also known as Flash cookies. Whenever a user visits a site that uses evercookies, the site issues an ID and stores it in multiple storage mechanisms, including cookies, LSOs and localStorage. In Figure 2a, the value *123* is stored in both HTTP and Flash cookies. When the user removes her HTTP cookie (Figure 2b), 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 2c).

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. Domain A, for instance, could pass 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 sharing cookie values, given the restriction that sites can't read each other cookies, in order to better facilitate targeting and real-time bidding [4].

In general, we consider the domains involved in cookie syncing to be third parties — that is, they appear on the first-party sites that a user explicitly chooses to visit. Although some sites such as `facebook.com` appear both in a first and third-party context, this distinction is usually quite clear.

The authors of [38] 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. In the present work, we study the implications of the fact that trackers that share an ID through syncing are in position to merge their database entries corresponding to a particular user, thereby reconstructing a larger fraction of the user's browsing patterns.

2.1 Related work

While HTTP cookies continue to be the most common method of third-party online tracking [41], a variety of more intrusive tracking mechanisms have been demonstrated, refined and deployed over the last few years. In response, various defenses have been developed, and a number of studies have presented measurements of the state of tracking. While advertising companies have claimed that tracking is essential for the web economy to function [42], a line of research papers have proposed and prototyped solutions to carry out behavioral advertising without tracking.

Fingerprinting, novel mechanisms. Researchers have presented novel browser fingerprinting mechanisms such as those based on performance metrics [31], the JavaScript engine [33] , the rendering engine [50], clock skew [23], WebGL and canvas fingerprinting [32]. Most of those studies followed the path opened by the influential Panopticlick study [16], which demonstrated the potentials of browser fingerprinting for online tracking.

Measurement studies. Web privacy measurement is a burgeoning field; an influential early work is [25] and prominent recent work includes [29, 41]. Mayer and Mitchell made a comprehensive survey of tracking in combination with the policy that surrounds it, and developed a tool for similar web privacy measurement studies [29]. Roesner et al. analyzed different tracking methods and suggested a taxonomy for third-party tracking [41].

Other papers have looked at various aspects of web privacy, including PII leakage [26], mobile web tracking [17], JavaScript inclusions [35], targeted advertisements [27], and the effectiveness of blocking tools [28].

Two studies measured the prevalence of different fingerprinting mechanisms and evaluated existing countermeasures [6, 37]. 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 [37]. 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 [6].

In comparison to these studies, we focus on canvas fingerprinting, which, to the best of our knowledge, has never been reported to be found in the wild and is much harder to block.

Several studies have looked at the use of Flash cookies (LSOs) and, in particular, the use of Flash cookies to respawn HTTP cookies [7, 30, 43]. Soltani et al. uncovered the first use of respawning by Flash cookies [43], and in a follow-up study, Ayenson et al. found the first use of cache ETags and localStorage

for respawning [7]. McDonald and Cranor analyzed the landing pages of 100 popular websites, plus 500 randomly-selected websites, and found two cases of respawning in the top 100 websites and no respawning in the randomly selected 500 sites [30]. 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 [44]. The main difference between our study and the papers mentioned here is that we automated respawning detection as explained in Section 4, and this allowed us to analyze orders of magnitude more sites.

Olejnik et al. studied *cookie syncing* (which they call cookie matching) [38]. They found that over 100 cookie syncing events happen on the top 100 sites. In comparison to their work, our study of cookie syncing (i) is large-scale, covering 3,000 sites, (ii) is based on crawling rather than crowd-sourcing, allowing easier comparative measurements over time and (iii) presents a global view, in that we go beyond detecting individual sync events and are able to capture and analyze the propagation of IDs through the tracking ecosystem. Further, we study how cookie syncing interacts with respawning, leading to more persistent tracking and widening the effects of these two vulnerabilities taken individually.

Program analysis of JavaScript (i.e., static analysis and dynamic analysis) is a common technique in web security [46]. A few studies have used such techniques for blocking or measuring web trackers. Orr et al. use static analysis to detect and block JavaScript-loaded ads [39]. Tran et al. use dynamic taint analysis to detect various privacy-invasive behaviors [48]. Acar et al. use behavioral analysis to detect fingerprinting scripts that employ font probing [6].

Defenses. Besson et al. [10] examined the theoretical boundaries of fingerprinting defenses using Quantified Information Flow. Following a more practical approach, Nikiforakis and others developed a defense called PriVaricator to prevent linkability from fingerprinters by randomizing browser features such as plugins [36]. Finally, Unger et al. [50], studied the potentials of browser fingerprinting as a defense mechanism against HTTP(S) session hijacking.

In Section 6.1 we discuss how existing privacy tools defend against the advanced tracking mechanisms we study.

Behavioral targeting without tracking. Several papers have addressed the question of whether all this tracking is in fact necessary — they proposed ways to achieve the purported goals of third-party tracking, primarily targeted advertising, without server-side profiles. In *Adnostic*, the browser continually updates a behavioral profile of the user based on browsing activity, and targeting is done locally [14]. *PrivAd* has a similar model, but includes a trusted party

that attempts to anonymize the client [20]. *RePriv* has the more general goal of enabling personalization via interest profiling in the browser [19]. Bilenko et al. propose a model in which the user's profile and recent browsing history is stored in a cookie [11]. Other work on similar lines includes [8, 34, 49].

3 Canvas Fingerprinting

Canvas fingerprinting works by drawing text onto canvas and reading the rendered image data back. In the following experiments we used an instrumented Firefox browser that we built by modifying the source code and logged all the function calls that might be used for canvas fingerprinting.

3.1 Methodology and Data collection

Our methodology can be divided into two main steps. In the first, we identified the ways we can detect canvas fingerprinting, developed a crawler based on an instrumented browser and ran exploratory crawls. This stage allowed us to develop a formal and automated method based on the early findings. In the second step, we applied the analysis method we distilled from the early findings and nearly fully automated the detection of canvas fingerprinting.

Mowery and Shacham used `fillText` and `ToDataURL` methods to draw text and read image data respectively [32]. We logged the return value of `ToDataURL` and, in order to find out the strings drawn onto the canvas, we logged the arguments of `fillText` and `strokeText` methods[3].

We logged the URL of the caller script and the line number of the calling (initiator) code using Firefox's `nsContentUtils::GetCurrentJSContext` and `nsJSUtils::GetCallingLocation` methods. This allowed us to precisely attribute the fingerprinting attempt to the responsible script and the code segment. All function call logs were parsed and combined in a SQLite database that allowed us to efficiently analyze the crawl data. For each visit, we also added cookies, localStorage items, cache metadata, HTTP request/response headers and request bodies to the SQLite database. We used mitmproxy [4] to capture

[3] In addition to these three methods we intercepted calls to `MozFetchAsStream`, `getImageData` and `ExtractData` methods which can be used to extract canvas image data. But we did not put effort into recording the extracted image data for three reasons: they were not used in the original canvas fingerprinting paper [32], they are less convenient for fingerprinting (requires extra steps), and we did not find any script that uses these methods and fingerprints other browser properties in the initial experiments.

[4] `http://mitmproxy.org/`

HTTP data and parsed data accumulated in the profile folder for other data such as cookies, localStorage and cache data. The aggregated data were used in the early stage analysis for canvas fingerprinting and evercookie detection, which is explained in Section 4.2. Our browser modifications for Firefox consist of mere 33 lines of code, spread across four files and the performance overhead of the modifications is minimal.

We crawled the home pages of the top 100,000 Alexa sites with the instrumented Firefox browser between 1-5 May 2014. We used Selenium [5] to drive browsers to sites and ran multiple Firefox instances in parallel to reduce the crawl time. Implementing some basic optimizations and a naive load limiting check, we were able to run up to 30 browsers in parallel on a 4-core 8GB desktop machine running GNU/Linux operating system. The modified browsers were run in a *chroot jail* to limit the effects of the host operating system.

False positive removal The Canvas API is used by many benign scripts to draw images, create animations or store content for games. During our crawls we found interesting use cases, such as generating dynamic *favicons*, creating tag clouds, and checking font smoothing support. By examining the distinctive features of false positives and the fingerprinting scripts found in the initial experiments, we distilled the following conditions for filtering out false positives:

- There should be both `ToDataURL` and `fillText` (or `strokeText`) method calls and both calls should come *from the same URL.*
- The canvas image(s) read by the script should contain more than one color and its(their) aggregate size should be *greater than 16x16 pixels.*
- The image should *not* be requested *in a lossy compression format* such as JPEG.

Checking the origin of the script for both read and write access helped us to remove scripts that use canvas for only generating images but not reading them or vice versa. Although it is possible that two scripts from the same domain can divide the work to circumvent our detection method, we accepted that as a limitation.

Enforcing a 16x16 pixel size limit allowed us to filter out scripts that read too few pixels to efficiently extract the canvas fingerprint. Although there are 2^{8192} possible color combinations for a 16x16 pixel image[5], operating systems or font

[5] $2^{\text{colordepth}^{w \times h}}$, $2^{32^{16 \times 16}} = 2^{8192}$ for the RGBA color space, which uses 24 bits for the colors (RGB) and 8 bits for the alpha channel. See, `http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html#pixel-manipulation`

Fingerprinting script	**No. of including sites**	**Text drawn into the canvas**
ct1.addthis.com/static/r07/core130.js	5282	Cwm fjordbank glyphs vext quiz,😃
i.ligatus.com/script/fingerprint.min.js	115	http://valve.github.io
src.kitcode.net/fp2.js	68	http://valve.github.io
admicro1.vcmedia.vn/fingerprint/figp.js	31	http://admicro.vn/
amazonaws.com/af-bdaz/bquery.js	26	Centillion
.shorte.st/js/packed/smeadvert-	14	http://valve.github.io
stat.ringier.cz/js/fingerprint.min.js	4	http://valve.github.io
cya2.net/js/STAT/89946.js	3	ABCDEFGHIJKLMNOPQRS TUVWXYZ abcdefghijklmnop qrstuvwxyz0123456789+/
*.revtrax.com/RevTrax/js/fp/fp.min.jsp	3	http://valve.github.io
pof.com	2	http://www.plentyoffish.com
*.rackcdn.com/mongoose.fp.js	2	http://api.gonorthleads.com
9 others*	9	*(Various)*
TOTAL	5559 ***(5542 unique[1])***	-

Table 1: Canvas fingerprinting domains found on Top Alexa 100K sites.
*: Some URLs are truncated or omitted for brevity. See Appendix for the complete list of URLs.
1: Some sites include canvas fingerprinting scripts from more than one domain.

libraries only apply anti-aliasing (which is an important source of diversity for canvas fingerprinting) to text larger than a minimum font size.[6]

The final check was to filter out cases where canvas image data is requested in a lossy compression format. Under a lossy compression scheme, the returned image may lose the subtle differences that are essential for fingerprinting.

Applying these checks, we reduced the false positive ratio to zero for the 100,000 crawl, upon which we perform our primary analysis. We used static analysis to make sure the scripts we flagged as canvas fingerprinting were also collecting other high-entropy browser properties such as plugins, navigator features and screen dimensions. It should be noted that in other pilot crawls (beyond 100K), we witnessed some false positives that our conditions failed to remove. Also, we believe that a determined tracker may potentially circumvent our detection steps using more advanced but less reliable attacks such as pixel stealing using SVG filters [45] or CSS shaders [24].

3.2 Results

Table 1 shows the prevalence of the canvas fingerprinting scripts found during the home page crawl of the Top Alexa 100,000 sites. We found that more than 5.5% of crawled sites actively ran canvas fingerprinting scripts on their home pages. Although the overwhelming majority (95%) of the scripts belong to a single provider (*addthis.com*), we discovered a total of 20 canvas fingerprinting provider domains, active on 5542 of the top 100,000 sites[7]. Of these, 11 provider domains, encompassing 5532 sites, are third parties. Based on these providers' websites, they appear to be companies that deploy fingerprinting as part of some other service rather than offering fingerprinting directly as a service to first parties. We found that the other nine provider domains (active on 10 sites) are in-house fingerprinting scripts deployed by first parties. Note that our crawl in this paper was limited to home pages. A deeper crawl covering internal pages of the crawled sites could find a higher percentage of fingerprinting.

The 5.5% prevalence is much higher than what other fingerprinting measurement studies had previously found (0.4% [37], 0.4%, 1.5% [6]), although these studies may not be directly comparable due to the differences in methodology and data collection. Also note that canvas fingerprinting was first used by AddThis

[6] `https://wiki.ubuntu.com/Fonts#Font_Smoothing`

[7] We discarded some cases where the canvas fingerprinting script is served from a content delivery network (CDN) and additional analysis was needed to distinguish between different providers serving from the same (CDN) domain. Including these cases would only change the number of unique sites with canvas fingerprinting to 5552 (from 5542).

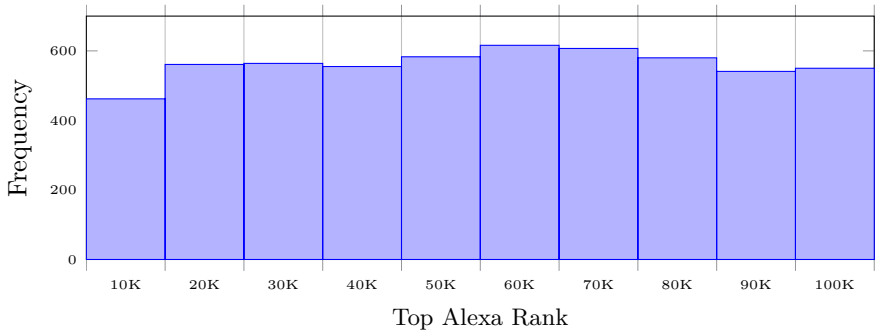


Figure 3: Frequency of canvas fingerprinting scripts on the home pages of Top Alexa 100K sites.

Rank interval	% of sites with canvas fingerprinting scripts
[1, 1K)	1.80
[1K, 10K)	4.93
[10K, 100K]	5.73

Table 2: Percentage of sites that include canvas fingerprinting scripts on the homepage, found in top 100K Alexa sites divided in intervals of variable length. Websites in the 1 to 1K rank interval are 2.5 times less likely to embed a canvas fingerprinting script than a site within 1K-10K interval.

between January 15 to February 1st, 2014, [8] which was after all the mentioned studies.

Below rank 10,000, the prevalence of canvas fingerprinting is close to uniform. However, we found that the top 1,000 sites are 2.5 times less likely to have included canvas fingerprinting scripts than the ones within the 1,000-10,000 range.

Note that the URL *http://valve.github.io*, printed by many scripts onto the canvas, belongs to the developer of an open source fingerprinting library[9]. Furthermore, all scripts except one use the same colors for the text and background shape. This similarity is possibly due to the use of the publicly available open source fingerprinting library *fingerprintjs* [51]. Figure 4 shows

[8] The date was determined using `http://httparchive.org/`

[9] See, `https://github.com/Valve/fingerprintjs/blob/v0.5.3/fingerprint.js#L250`

five different canvas images used by different canvas fingerprinting scripts. The images are generated by intercepting the canvas pixel data extracted by the scripts listed in Table 1.

Figure 4: Different images printed to canvas by fingerprinting scripts. Note that the phrase *"Cwm fjordbank glyphs vext quiz"* in the top image is a *perfect pangram*, that is, it contains all the letters of the English alphabet only once to maximize diversity of the outcomes with the shortest possible string.

Manually analyzing AddThis's script, we found that it goes beyond the ideas previously discussed by researchers and adds new tests to extract more entropy from the canvas image. Specifically, we found that in addition to the techniques outlined in Mowery and Shacham's canvas fingerprinting paper [32] AddThis scripts perform the following tests:

- Drawing the text twice with different colors and the default fallback font by using a fake font name, starting with *"no-real-font-"*.
- Using the perfect pangram[10] *"Cwm fjordbank glyphs vext quiz"* as the text string
- Checking support for drawing Unicode by printing the character *U+1F603 a smiling face with an open mouth.*
- Checking for canvas *globalCompositeOperation* support.
- Drawing two rectangles and checking if a specific point is in the path by the *isPointInPath* method.

By requesting a non-existent font, the first test tries to employ the browser's default fallback font. This may be used to distinguish between different browsers

[10] http://en.wikipedia.org/wiki/List_of_pangrams#Perfect_pangrams_in_English_.2826_letters.29

and operating systems. Using a perfect pangram, which includes a single instance of each letter of the English alphabet, the script enumerates all the possible letter forms using the shortest string. The last three tests may be trying to uncover the browser's support for the canvas features that are not equally supported. For instance, we found that the Opera browser cannot draw the requested Unicode character, *U+1F603*.

Another interesting canvas fingerprinting sample was the script served from the *admicro.vcmedia.vn* domain. By inspecting the source code, we found that the script checks the existence of 1126 fonts using JavaScript font probing.

Overall, it is interesting to see that commercial tracking companies are advancing the fingerprinting technology beyond the privacy/security literature. By collecting fingerprints from millions of users and correlating this with cookie based identification, the popular third party trackers such as AddThis are in the best position to both measure how identifying browser features are and develop methods for monitoring and matching changing fingerprints. Note that according to a recent ComScore report, AddThis "solutions" reaches 97.2% of the total Internet population in the United States and get 103 billion monthly page views.[11]

4 Evercookies

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 [7, 14, 43]. 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 both HTTP cookies and IndexedDB.

4.1 Detecting User IDs

Given that not all instances of the various potential storage vectors are used to track users, detecting evercookies hinges on determining whether a given string

[11] `http://www.businesswire.com/news/home/20131113005901/en/comScore-Ranks-AddThis-1-Distributed-Content-United`

can serve as a user ID. In order to detect persistent IDs in a given storage vector, we leveraged data from two simultaneous crawls on separate machines and applied the following set rule set for determining which elements are identifying. We present the rules with respect to HTTP cookies but note that they are applicable to other storage locations of a similar format.

- Eliminate cookies that expire within a month of being placed. These are too transient to track a user over time.
- Parse cookie value strings using common delimiters (e.g. : and &). This extracts potentially identifying strings from non-essential data.
- Eliminate parsed fields which don't remain constant throughout an individual crawl. Identifiers are likely to be unchanging.
- Compare instances of matching parsed cookie fields (for cookies with the same domain and name) between two unrelated crawls on different machines.
  - Eliminate fields which are not the same length.
  - Eliminate fields which are more than 33% similar according to the Ratcliff-Obershelp algorithm [12]. These are unlikely to contain sufficient entropy.

The presented method provides a strict and conservative detection of identifiers that we believe (through manual inspection) to have a very low false positive rate. We anticipate several sources of false negatives, for example ID strings that are obfuscated or embedded in longer strings using non-standard delimiters or ID strings that happen to have a high similarity. Similarly, an adversarial tracker could continually change an identifier or cookie sync short-lived identifiers, but keep a mapping on the back end to enable long-term tracking. Therefore, the results of this analysis provide a lower bound on the presence of evercookie storage vectors and on the level of cookie syncing.

4.2 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[12]. We developed a procedure to automate the detection of respawning by Flash cookies employing

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

the method discussed in Section 4.1 to detect IDs and using GNU/Linux's *strace* [22] debugging tool to log access to Flash cookies.

Compared to earlier respawning studies [7, 30, 43], the method employed in this paper 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, 700 sites). Note that, similar to [30], we only visited the home pages, whereas [7, 43] visited 10 internal links on each website. Another methodological difference is that we maintained the Flash cookies when visiting different websites, whereas [7, 43] used a virtual machine to prevent contamination. Last, [30] 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 and saved the accumulated HTTP and Flash cookies ($Crawl_1$). We then made three 10,000 site crawls, 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 are 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. We used the 100K crawl described in the canvas fingerprinting experiments for this purpose.

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 $Crawl_4$.

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 3 shows that on six of the top 100 sites, Flash cookies are 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 is found to use Flash based fingerprinting [6]. 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). The domain with the second highest number of respawns was `kiks.yandex.ru` which restored 11 cookies on 11 sites in each crawl ($Crawl_{2,3}$).

IndexedDB as Evercookie While running crawls for canvas fingerprinting experiments, we looked for sites that store data in the IndexedDB storage vector. Specifically, we checked the *storage/persistent* directory of the Firefox profile. A very small number of sites, only 20 out of 100K, were found to use the IndexedDB storage vector. Analyzing the IndexedDB file from the respawning crawl ($Crawl2$) described above, we found that a script from the `weibo.com` domain stored an item in the IndexedDB that exactly matched the content of the Flash cookie named `simg.sinajs.cn/stonecc_suppercookie.sol`. This Flash cookie was used to respawn HTTP cookies on Chinese microblogging site weibo.com and its associated web portal sina.com.cn. To the best of our knowledge, this is the first report of IndexedDB as an evercookie vector. A more thorough study of respawning based on IndexedDB is left for future study.

4.3 HTTP cookies respawning Flash cookies

We ran a sequential crawl of the Top 3,000 Alexa sites and saved the accumulated HTTP and Flash cookies. We extracted IDs from this crawl's HTTP cookies using the method described in Section 4.1. We then made an additional

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 3: Top-ranked websites found to include respawning based on Flash cookies. 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.

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 4.1. This enables us to ensure that the IDs are indeed found in the Flash objects of both crawls, aren't common to unrelated crawls, and exist as IDs on the original machine. Using this method, we detected 11 different unique IDs common between the three storage locations.

These 11 IDs correspond to 14 first-party domains, a summary of which is provided by Table 8 in the Appendix. We primarily observe respawning from JavaScript originating from two third-parties: `www.iovation.com`, a fraud detection company that is specialized in device fingerprinting, and `www.postaffiliatepro.com`, creators of affiliate tracking software (that runs in the first-party context). We also observe three instances of what appears to be in-house respawning scripts from three brands: Twitch Interactive (`twitch.tv` and `justin.tv`), `casino.com`, and `xlovecam.com`.

5 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 evercookies. First, pairs of domains who both know the same IDs via synchronization can use these IDs to merge their tracking databases on the back end. Second, respawned cookies may contain IDs that are widely shared due to prior sync events, enabling trackers to link a user's browsing histories from before and after clearing browsing state.

In this section, we present our method for detecting syncs, present an overview of the synchronization landscape and examine the threats of back-end database merges and history-linking for users who clear state.

5.1 Detecting cookie synchronization

Using the techniques outlined in Section 4.1, we identified cookies containing values likely to be user IDs. In order to learn which domains know a given ID through synchronization, we examined cookie value strings and HTTP traffic.

If a domain owns a cookie containing an ID, clearly the domain knows that ID. In fact, a telltale sign of cookie syncing is multiple domains owning cookies containing the same ID. Likewise, if an ID appears anywhere in a domain's URL string (e.g. in the URL parameters), then that domain also knows the ID. Note that a given tracker may simply ignore an ID received during a sync, but as we will demonstrate in Section 5.3, trackers opting to store IDs have the ability to gain user data through history merging.

The domains involved in HTTP traffic can be divided into (*referrer*, *requested URL*, *location*) tuples in which the location domain is non-empty only for HTTP response redirects. The rules for ID passing are as follows:

- If an ID appears in a requested URL, the requested domain learns the ID.
- If an ID appears in the referrer URL, the requested domain and location domain (if it exists) learn the ID.
- If an ID appears in the location URL, the requested domain learns the ID.

We cannot assume that the referrer learns a synced ID appearing in the requested URL or location URL string [38]. In particular, third-party JavaScript executing a sync on a first-party site will cause the first-party to show up as the referrer, even though it may not even be aware of the ID sync. Although we can determine the directionality of ID syncs in the cases of redirects, the fraction of flows in which we could determine both the sender and receiver was small. Hence, when examining cookie synchronization, we focused on which domains knew a given ID, rather than attempting to reconstruct the paths of ID flows.

5.2 Basic results

Before examining the privacy threats that can stem from cookie synchronization, we first provide an overview of cookie syncing activities that occur when browsing under different privacy settings. We ran multiple crawls of the top 3,000 Alexa domains on Amazon EC2[13] instances using three different Firefox privacy settings: allowing all cookies (i.e. no privacy-protective measures), allowing all cookies but enabling Do Not Track, and blocking third-party cookies. 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 is consistent with studies such as Balebako et al. [9] — they find that, due to lack of industry enforcement, Do Not Track provides little practical protection against trackers. We therefore omit further measurement and analysis of the effect of Do Not Track in this section.

Table 4 shows high-level statistics for illustrative crawls under the two third-party cookie settings. We say that an ID is involved in synchronization if it is known by at least two domains. Cookies and domains are involved in synchronization if they contain or know such an ID, respectively. The statistics displayed aggregate 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.

Appendix A.2 shows a summary of the top 10 parties involved in cookie synchronization under both cookie policies. Observe that although some parties are involved in less syncing under the stricter cookie policy, many of the top parties receive the same number of IDs. Overall, disabling third-party cookies reduces the number of synced IDs and parties involved in syncing by nearly a factor of two. While this reduction appears promising from a privacy standpoint, in the next section we will see that even with this much sparser amount of data,

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

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: Comparison of high-level cookie syncing statistics when allowing and disallowing third-party cookies (top 3,000 Alexa domains). 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.

database merges could enable domains to reconstruct a large portion of a user's browsing history.

Included in Appendix A.3 is a summary of the top 10 most shared IDs under both cookie policies. For a specific example, consider the most shared ID which all third party cookies are allowed, which was originally created by `turn.com`. This ID is created and placed in a cookie on the first page visit that includes Turn as a third-party. On the next page visit, Turn makes GET requests to 25 unique hostnames with a referrer of the form `http://cdn.turn.com/server/ddc.htm?uid=<unique_id>`... that contains its ID. These 25 parties gain knowledge of Turn's ID, as well as their own tracking cookies, in the process. Similar sharing occurs as the user continues to browse, eventually leading to 43 total domains. With third-party cookies disabled, the top shared IDs come from a disjoint set of parties, largely composed of syncs which share a first party cookie with several third-party sites.

5.3 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, assuming no collaboration among third-party trackers, only a handful of trackers are in position to track a sizeable fraction of an individual's browsing history. As per Olejnik et al [38], if a visited first party appears as the referrer in a request to another domain, we assume the second domain knows about this visit. For a crawl of 3,000 sites when allowing all cookies, only two of the 730 trackers could recover more than 40% of a user's history and only 11 could recover more than 10%. When disabling third-party cookies, the corresponding numbers are two and six, respectively. These results are consistent with earlier findings in Roesner et al [41].

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 5 shows the proportion of a user's 3,000-site browsing history a domain can recover, in decreasing sorted order, if a user enables all cookies. The figure when blocking third-party cookies (also Figure 5) takes a identical shape but is steeper because it only includes roughly 60% as many parties.

Observe that after introducing the ability for a site to merge records directly with one other tracker, the known proportion of a user's 3,000-site history dramatically increased for a large number of sites. When third-party cookies are allowed, 101 domains can reconstruct over 50% of a user's history and 161 could recover over 40%. Even when these cookies are blocked, 44 domains could recover over 40% of a user's history.

Not much is known about how prevalent back-end database merges are. In terms of incentives, a pair of trackers may enter into a mutually beneficial arrangement to increase their respective coverage of users' browsing histories, or a large tracker may act as a data broker and sell user histories for a fee.

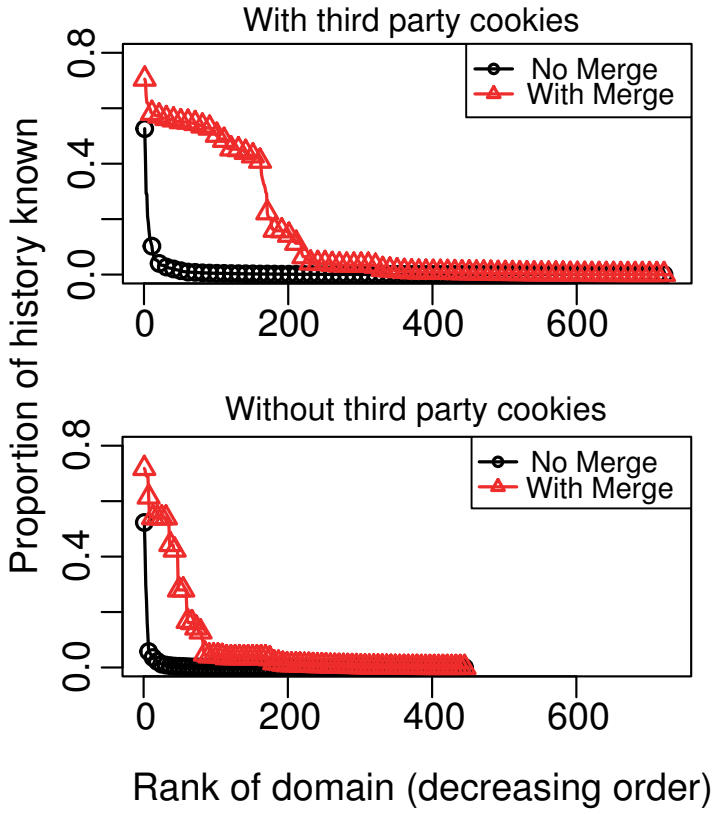


Figure 5: Proportions of user history known when allowing and blocking third party cookies under the two different merging schemes. Note that since the x-axis is sorted by the proportion of a user's history that a domain can recover, the domains may appear in different orders for the different models.

5.4 Respawning and syncing

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 clearing 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 two 3,000 site crawls on two EC2 instances, A and 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 between the two crawls on machine B either through Flash or through other means[14]. 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 JavaScript-based browser fingerprinting. We conjecture that these IDs were respawned through some form of passive, server-side fingerprinting[15].

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 is not in a position to observe a large fraction of a user's history — it only appears on a single first party domain (`casino.com`). In fact, the largest observed 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.

[14]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.

[15]Note that a document from one of these respawning domains, `merchenta.com` mentions 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`.

In particular, we observed `adnxs.com` on approximately 11% of first party sites across the two crawls. Thus `adnxs.com` now has 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 do so. This scenario enables at least 11% of a user's history to be tracked over time.

Our measurements in this section illustrate the potential for cookie respawning and syncing event on a single site by a small tracker to enable a large proportion of a user's history to be tracked by more prolific third parties.

6 Discussion

After presenting an evaluation of advanced tracking techniques, we now discuss the potential defenses against these methods and the implications of our study for privacy-conscious users.

6.1 Mitigation

A blunt way to defend against tracking is to simply block third-party content. This is the approach taken by tools such as AdBlock Plus[16] and Ghostery.[17] The user may also disable evercookie storage vectors such as Flash cookies [3], but to the best of our knowledge, tracking vectors such as localStorage, IndexedDB and canvas cannot be disabled, often due to the fact that doing so would break core functionality.

Canvas fingerprinting: The initial canvas fingerprinting study discusses possible countermeasures such as adding noise to the pixel data or trying to produce same pixel results for every system. Finding some barriers to all these options, the paper concludes that asking user permission for each canvas read attempt may be the only effective solution. Indeed, this is precisely the technique adopted in the Tor Browser, the only software that we found to successfully protect against canvas fingerprinting. Specifically, the Tor Browser returns an empty image from all the canvas functions that can be used to read image data [13]. The user is then shown a dialog where she may permit trusted sites to access the canvas. We confirmed the validity of this approach when visiting a site we built which performs browser fingerprinting.

As for more traditional fingerprinting techniques, the Tor browser again appears to be the only effective tool. With the exception of a recent Mozilla effort to

[16] `https://adblockplus.org`
[17] `http://www.ghostery.com`

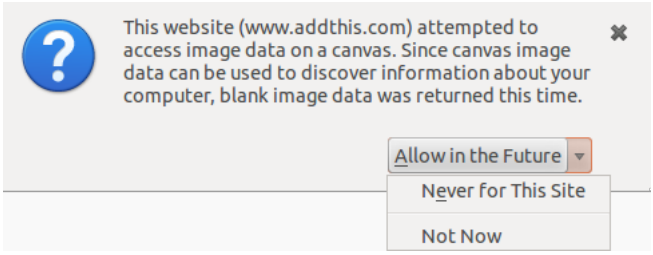


Figure 6: The Tor Browser's notification dialog for canvas read attempts. The empty image is returned to thwart canvas fingerprinting.

limit plugin enumeration [2], browser manufacturers have not attempted to build in defenses against fingerprinting. We note that they are in a position to facilitate such defenses by providing APIs or settings or tools that can be used to develop countermeasures.

Finally, academic studies on mitigating browser fingerprinting are promising but still far from providing practically implementable and comprehensive countermeasures that address all the attack possibilities [10, 36].

Evercookies: The straightforward way to defend against evercookies is to clear all possible storage locations. The long list of items removed by the Tor Browser when a user switches to a new identity provides a hint of what can be stored in unexpected corners of the browser: "searchbox and findbox text, HTTP auth, SSL state, OCSP state, site-specific content preferences (including HSTS state), content and image cache, offline cache, Cookies, DOM storage, DOM local storage, the safe browsing key, and the Google wifi geolocation token..." [40].

The user interfaces provided by popular browsers for managing browsing information are often fragmented, incomplete, or esoteric. For instance, Firefox's Clear Recent History interface does not clear localStorage if the user doesn't select "Everything" as the time range of removal[18]and there is no unified interface for checking what is stored in localStorage and IndexedDB. Similarly, Offline Website Data (AppCache and Cache) can only be checked by visiting a separate `about:cache` page.

Even if the user manages to clear all storage vectors, the fact that Flash storage is not isolated[19] between browsers which use the Adobe Flash plugin (e.g. Firefox, Chromium, and Internet Explorer) still creates an opportunity for respawning. Consider the common scenario of a multi-user environment where

[18]Bug 527667 `https://bugzilla.mozilla.org/show_bug.cgi?id=527667`
[19]Confirmed through manual analysis

Alice uses browser A and Bob uses browser B, without any OS-level separation of user accounts. Assume that Alice is privacy-conscious and clears browser state frequently, but Bob does not. Consider an ID on Browser A is shared between Browser A's Flash Cookies and HTTP Cookies. When Bob browses, X may be respawned as an HTTP cookie in browser B. In Section 4.2, we showed that this behavior occurs in the wild. Now when Alice completely clears the state of Browser A, the ID X will be removed from common flash storage and Browser A's HTTP storage. Crucially, however, when Bob browses again, it could be respawned from B's HTTP storage to common flash storage and later when Alice browses again, back to A's HTTP storage. We showed in Section 4.3 that HTTP-to-Flash respawning occurs in the wild as well. Thus the only way to defend against this attack in a multi-browser environment is to clear state on all browsers simultaneously. As a proof-of-concept, we manually tested the first-party domains on which we observe HTTP-to-Flash respawning (Appendix Table 8) and we found this exact scenario occurs on both `casino.com` and `xlovecam.com`.

Cookie syncing: We're not aware of any tools that specifically block cookie syncing. The bluntest approach, of course, is to simply block third-party cookie placement and HTTP traffic. EFF's newly released tool Privacy Badger[20] uses heuristics to block third-party cookies with the goal of preventing third-party tracking, erring on the side of false positives (i.e., blocking too many cookies). The Tor Browser Bundle (TBB) prevents cross-site cookie tracking by disabling all third-party cookies, and not storing any persistent data such as cookies, cache or localStorage. A more targeted solution would be to block third-party traffic containing strings that are cookie values, but this approach will likely suffer from false negatives. However, even a perfect blocking tool is flawed if it is not used immediately from a completely fresh browsing state. For instance, if a user browses for a short amount of time before installing such a tool, trackers may have already placed and synced cookies — enabling them to merge data in the back-end. If these IDs are maintained through a hard-to-block technique such as canvas fingerprinting, the trackers can still follow a user as he browses and link their records through these previously-established syncing relationships even if all future syncs are blocked.

6.2 The effect of opt-out

In order to study the effect of ad-industry opt-out tools on the tracking mechanisms we study, we opted-out from all the listed companies on the Network Advertising Initiative (NAI)[21] and European Interactive Digital Advertising

[20] `https://www.eff.org/privacybadger`
[21] `http://www.networkadvertising.org/choices/`

Alliance (EDAA)[22] opt-out pages.

Canvas fingerprinting: For each canvas fingerprinting script we visited two sites that included this script. We did not observe any website that stopped collecting canvas fingerprint due to opt-out.[23] This was despite the fact that AddThis was listed on the NAI opt-out page and Ligatus (second most popular canvas fingerprinter) was listed on EDAA's page.

We also tried opting-out by on AddThis' own Data Collection Opt-Out website[24], which again, did not stop AddThis's script collecting the canvas fingerprint.

Respawning: We did not observe any change in cookie respawning from HTTP to Flash cookies. This is expected as the parties involved are not participants in the advertising opt-out initiatives.

Cookie syncing: The use of opt-out cookies reduces the number of IDs involved in cookie synchronization by 30%. However, we see only a 5% reduction in the number of parties involved in synchronization. This reduction is comparatively smaller than the reduction seen when the browser is set to block third-party cookies. The composition of the top parties involved in synchronization is nearly the same as in the first-party cookie only case seen in Appendix A.2. In Section 5.3 we show how, even under the larger reduction in sync activity afforded by blocking all third-party cookies, it is 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 do not promise to stop tracking when a user opts out, but only 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.

6.3 Implications

Let us consider the level of user effort and sophistication required for effective mitigation. First, users must be very careful in their use of existing tools, such as clearing state on all browsers at once or installing blocking tools before cookie syncing has occurred. Second, users must accept usability drawbacks such as the prompt for Canvas API access. Third, there are also trade-offs in functionality and content availability. Finally, the rapid pace at which

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

[23] We observed that two of the 20 fingerprinting scripts (revtrax.com and vcmedia.vn) were missing on the sites we found them before, though we checked to ensure that this was not related to opt-out.

[24] `http://www.addthis.com/privacy/opt-out`

new tracking techniques are developed and deployed implies that users must constantly install and update new defensive tools. It is doubtful that even privacy-conscious and technologically-savvy users can adopt and maintain the necessary privacy tools without ever experiencing a single misstep.

Evercookies were at the center of fierce debates when Soltani et al. reported their findings [43] a few years ago. Although this resulted in a lawsuit and a $500,000 settlement [14], we find an increasing number of websites using these tracking technologies as well as significant advances in the technologies themselves.

The World Wide Web Consortium (W3C) standards documents that describe three new storage APIs (localStorage, IndexedDB and WebStorage APIs) have the same boilerplate warning about the tracking potentials of these mechanisms[25] and mention the necessity of an interface to communicate the evercookie risk. Perhaps a fruitful future direction for standards bodies is to consider privacy issues at the design stage, acknowledging that without such a proactive effort, tracking techniques are likely to have the upper hand over defenses. W3C's draft specification "Fingerprinting Guidance for Web Specification Authors" is a notable effort in this direction, for providing a guideline to Web specification authors about privacy risks of browser fingerprinting [15].

6.4 A Path Forward

Blocking tools are currently the primary solution to third-party tracking for the informed user. We believe that these tools can be greatly improved by a back-end consisting of regular web-scale crawls. Crawlers can incorporate sophisticated rules to detect unwanted tracking, as we have shown, whereas it would be difficult to deploy these directly into browser tools. Accordingly, we plan to further scale our crawling infrastructure, while continuing to release results in a machine-readable format.

Crawler-supported blocking tools could also benefit from machine learning and crowd-sourcing (instead of rules hand-coded by experts) for minimizing false positives and negatives. For example, we have produced an initial classification of canvas fingerprinting scripts on 100,000 sites, but there are surely many more such scripts in the web's long tail, which suggests that a semi-supervised learning approach could be effective. The resulting classifier would label scripts that access the canvas API as canvas fingerprinters or non-canvas-

[25] `http://www.w3.org/TR/webstorage/#user-tracking`, `http://www.w3.org/TR/IndexedDB/#user-tracking`, `http://www.w3.org/TR/webdatabase/#user-tracking`

fingerprinters. Turning to crowdsourcing, a browser tool could default to blocking all canvas write/read attempts, but slowly incorporate user feedback about broken functionality to train a model for identifying true fingerprinting attempts. Of course, these two approaches can be combined.

Finally, publishers have little insight into the types of tracking occurring on their own sites. The tools that we and others have built can be re-purposed to provide transparency not just to end-users but also allow publishers an in-depth look into how trackers collect data from their sites, where the data flows, and how it is used. This will allow them to discriminate between advertising or analytics providers on the basis of privacy practices.[26] If combined with public pressure to hold *first parties* accountable for online tracking and not just third parties, it can move online tracking in a more transparent and privacy-friendly direction.

7 Conclusion

We present a large-scale study of tracking mechanisms that misuse browser features to circumvent users' tracking preferences. We employed innovative measurement methods to reveal their prevalence and sophistication in the wild. Current options for users to mitigate these threats are limited, in part due to the difficulty of distinguishing unwanted tracking from benign behavior. In the long run, a viable approach to online privacy must go beyond add-ons and browser extensions. These technical efforts can be buttressed by regulatory oversight. In addition, privacy-friendly browser vendors who have hitherto attempted to take a neutral stance should consider integrating defenses more deeply into the browser.

8 Acknowledgements

The authors would like to thank Joseph Bonneau, Edward Felten, Georg Koppen, Lukasz Olejnik, Mike Perry, Vitaly Shmatikov, Roland Illig, and Matthew Wright for valuable feedback, Dillon Reisman and Pete Zimmerman for helping develop some of the infrastructure used, Oscar Reparaz for chroot tips and

[26] In fact, there is a fledgling commercial market for such tools [1], but they are not very sophisticated.

Junjun Chen for earlier work on cookie syncing that helped our understanding of the practice. For KU Leuven, this work was partially funded by the projects IWT SBO SPION, FWO G.0360.11N, FWO G.0686.11N, and the KU Leuven BOF OT project ZKC6370 OT/13/070.

References

[1] Privacychoice - get a free privacy scan of your site. `http://privacychoice.org/assessment`.

[2] Bug 757726 - disallow enumeration of navigator.plugins. `https://bugzilla.mozilla.org/show_bug.cgi?id=757726`, May 2012.

[3] Manage, disable Local Shared Objects | Flash Player. `http://helpx.adobe.com/flash-player/kb/disable-local-shared-objects-flash.html`, 2014.

[4] Doubleclick ad exchange real-time bidding protocol: Cookie matching. `https://developers.google.com/ad-exchange/rtb/cookie-guide`, February 2014.

[5] Selenium - Web Browser Automation. `http://docs.seleniumhq.org/`, 2015.

[6] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. FPDetective: Dusting the Web for fingerprinters. In *20th ACM Conference on Computer and Communications Security (CCS)*, pages 1129–1140. ACM, 2013.

[7] 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.

[8] Michael Backes, Aniket Kate, Matteo Maffei, and Kim Pecina. Obliviad: Provably secure and practical online behavioral advertising. In *IEEE Security and Privacy (S&P)*, pages 257–271. IEEE, 2012.

[9] 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 (W2SP)*. IEEE, 2012.

[10] Frédéric Besson, Nataliia Bielova, Thomas Jensen, et al. Enforcing Browser Anonymity with Quantitative Information Flow. 2014.

[11] Mikhail Bilenko, Matthew Richardson, and Janice Y Tsai. Targeted, not tracked: Client-side solutions for privacy-friendly behavioral advertising. In *Privacy Enhancing Technologies (PETS)*. Springer, 2011.

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

[13] Kathleen Brade. gitweb.torproject.org - torbrowser.git/blob - src/current-patches/firefox/0019-add-canvas-image-extraction-prompt.patch. `https://gitweb.torproject.org/torbrowser.git/blob/HEAD:/src/current-patches/firefox/0019-Add-canvas-image-extraction-prompt.patch`, November 2012.

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

[15] Nick Doty. Fingerprinting Guidance for Web Specification Authors. `http://w3c.github.io/fingerprinting-guidance/`, 2014.

[16] Peter Eckersley. How unique is your web browser? In *Privacy Enhancing Technologies*, pages 1–18. Springer, 2010.

[17] Christian Eubank, Marcela Melara, Diego Perez-Botero, and Arvind Narayanan. Shining the floodlights on mobile web tracking - a privacy survey. In *"Web 2.0 Security and Privacy"*, May 2013.

[18] Edward W. Felten. If You're Going to Track Me, Please Use Cookies. `https://freedom-to-tinker.com/blog/felten/if-youre-going-track-me-please-use-cookies/`, 2009.

[19] Matthew Fredrikson and Benjamin Livshits. Repriv: Re-imagining content personalization and in-browser privacy. In *IEEE Security and Privacy (S&P)*, pages 131–146. IEEE, 2011.

[20] Saikat Guha, Bin Cheng, and Paul Francis. Privad: practical privacy in online advertising. In *USENIX Conference on Networked Systems Design and Implementation*, pages 169–182. USENIX Association, 2011.

[21] Samy Kamkar. Evercookie - virtually irrevocable persistent cookies. `http://samy.pl/evercookie/`, Sep 2010.

[22] Michael Kerrisk. strace(1) - linux manual page. `http://man7.org/linux/man-pages/man1/strace.1.html`, May 2014.

[23] Tadayoshi Kohno, Andre Broido, and Kimberly C Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, 2005.

[24] Robert Kotcher, Yutong Pei, Pranjal Jumde, and Collin Jackson. Cross-origin pixel stealing: timing attacks using CSS filters. In *ACM Conference on Computer and Communications Security (CCS)*, pages 1055–1062. ACM, 2013.

[25] 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.

[26] Balachander Krishnamurthy and Craig E Wills. On the leakage of personally identifiable information via online social networks. In *ACM Workshop on Online Social Networks*, pages 7–12. ACM, 2009.

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

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

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

[30] 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.

[31] 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.

[32] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in HTML5. In *Web 2.0 Workshop on Security and Privacy (W2SP)*. IEEE, 2012.

[33] 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.

[34] Arvind Narayanan, Jonathan Mayer, and Subodh Iyengar. Tracking Not Required: Behavioral Targeting. `http://33bits.org/2012/06/11/tracking-not-required-behavioral-targeting/`, 2012.

[35] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. You are what you include: Large-scale evaluation of remote javascript inclusions. In *ACM Conference on Computer and Communications Security (CCS)*, pages 736–747. ACM, 2012.

[36] Nick Nikiforakis, Wouter Joosen, and Benjamin Livshits. PriVaricator: Deceiving Fingerprinters with Little White Lies. Available at `http://research.microsoft.com/en-us/um/people/livshits/papers%5Ctr%5Cprivaricator.pdf`.

[37] 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 (SP), 2013 IEEE Symposium on*, pages 541–555. IEEE, 2013.

[38] Lukasz Olejnik, Tran Minh-Dung, and Claude Castelluccia. Selling Off Privacy at Auction. In *Annual Network and Distributed System Security Symposium (NDSS)*. IEEE, 2014.

[39] Caitlin R Orr, Arun Chauhan, Minaxi Gupta, Christopher J Frisz, and Christopher W Dunn. An approach for identifying JavaScript-loaded advertisements through static program analysis. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, pages 1–12. ACM, 2012.

[40] Mike Perry, Erinn Clark, and Steven Murdoch. The design and implementation of the Tor browser [draft]. `https://www.torproject.org/projects/torbrowser/design`, 2013.

[41] F. Roesner and T. Kohno und D. Wetherall. Detecting and defending against third-party tracking on the web. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI*, 2012.

[42] Natasha Singer. Do Not Track? Advertisers Say 'Don't Tread on Us'. `http://www.nytimes.com/2012/10/14/technology/do-not-track-movement-is-drawing-advertisers-fire.html`, 2012.

[43] 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.

[44] Ove Sorensen. Zombie-cookies: Case studies and mitigation. In *Internet Technology and Secured Transactions (ICITST)*, pages 321–326. IEEE, 2013.

[45] Paul Stone. Pixel perfect timing attacks with HTML5. *Context Information Security (White Paper)*, 2013.

[46] Ankur Taly, John C Mitchell, Mark S Miller, J Nagra, et al. Automated analysis of security-critical javascript apis. In *IEEE Security and Privacy (S&P)*, pages 363–378. IEEE, 2011.

[47] James Temple. Stale Cookies: How companies are tracking you online today. `http://blog.sfgate.com/techchron/2013/10/02/stale-cookies-how-companies-are-tracking-you-online-today/`, 2013.

[48] Minh Tran, Xinshu Dong, Zhenkai Liang, and Xuxian Jiang. Tracking the trackers: Fast and scalable dynamic analysis of web content for privacy violations. In *Applied Cryptography and Network Security*, pages 418–435. Springer, 2012.

[49] Minh-Dung Tran, Gergely Acs, and Claude Castelluccia. Retargeting without tracking. *arXiv preprint arXiv:1404.4533*, 2014.

[50] 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.

[51] Valentin Vasilyev. Valve/fingerprintjs. `https://github.com/Valve/fingerprintjs`, 2012.

A Appendices

A.1 Flash cookies with the most respawns

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

A.2 Top parties involved in Cookie Sync

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 6: Number of IDs known by the Top 10 parties involved in cookie sync under both the policy of allowing all cookies and blocking third-party cookies.

A.3 Top IDs involved in Cookie Sync

All Cookies Allowed		No 3P Cookies	
ID Creator	**# D.**	**ID Creator**	**# D.**
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 7: Number of domains which have knowledge of unique IDs created by each listed domain. ID creator determined manually by first placement of cookie (* the relationship was unclear from HTTP/cookie logs).

A.4 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.js https://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 8: Summary of HTTP respawning. "Source of Respawn" describes whether or not the tracking occurs in the first-party or third-party context and lists the entity responsible for writing the script. * Interestingly fling.com has the ID passed from the third-party context and saved in the first-party context

A.5 List of Canvas Fingerprinting Scripts

Domain	URL of the Fingerprinting Script
addthis.com	http://ct1.addthis.com/static/r07/core130.js, http://ct1.addthis.com/static/r07/sh157.html# and 16 others
ligatus.com	http://i.ligatus.com//script/fingerprint.min.js
kitcode.net	http://src.kitcode.net/fp2.js
vcmedia.vn	http://admicro1.vcmedia.vn/fingerprint/figp.js
amazonaws.com[1]	https://s3-ap-southeast-1.amazonaws.com/af-bdaz/bquery.js
shorte.st	http://static.shorte.st/js/packed/smeadvert-intermediate...
ringier.cz	http://stat.ringier.cz/js/fingerprint.min.js
cya2.net	http://cya2.net/js/STAT/89946.js?ver=adl&cid=T...
revtrax.com	http://images.revtrax.com/RevTrax/js/fp/fp.min.jsp
pof.com	http://www.pof.com/
rackcdn.com[2]	https://c44ed9b5e....ssl.cf2.rackcdn.com/mongoose.fp.js
hediyera.com	http://www.hediyera.com/js/dota/dota.js
meinkauf.at	http://www.meinkauf.at/assets/application-....js
freevoipdeal.com	http://www.freevoipdeal.com/en/asset/js/...
voipbuster.com	http://www.voipbuster.com/en/asset/js/...
nonoh.net	http://www.nonoh.net/asset/js/e4cf90bfdfa29f5fd61050d14a11f0a1
49winners.com	http://49winners.com/js/49w3/fingerprint.js?v=1.1
freecall.com	http://www.freecall.com/asset/js/f4ccb1cb0e4128b6d4b08f9eb2c8deb4
domainsigma.com	http://static.domainsigma.com/static/public/js/common.9b6f343c.js
insnw.net[3]	http://dollarshaveclub-002.insnw.net/assets/dsc/dsc.fingerprint...

Table 9: URLs of Canvas Fingerprinting JavaScript. The URL parts snipped for brevity are denoted by ...
1: s3-ap-southeast-1.amazonaws.com (sends the collected fingerprint to adsfactor.net domain).
2: 44ed9b5ebea0e0739cdcbf3c0901f34702b963a7ca35c5bc1c.ssl.cf2.rackcdn.com (sends the collected fingerprint to api.gonorthleads.com).
3:dollarshaveclub002.insnw.net

Publication

The Leaking Battery: A Privacy Analysis of the HTML5 Battery Status API

Publication Data

Olejnik, L., Acar, G., Castelluccia, C., and Diaz, C. The leaking battery - A privacy analysis of the HTML5 battery status API. In *Data Privacy Management, and Security Assurance - 10th International Workshop, DPM 2015, and 4th International Workshop, QASA 2015, Vienna, Austria, September 21-22, 2015. Revised Selected Papers* (2015), pp. 254–263

Contributions

- I contributed to the development of the floating-point based attack. I collected battery related data from different devices. I wrote scripts to analyze the collected data.

The Leaking Battery

A Privacy Analysis of the HTML5 Battery Status API

Łukasz Olejnik[1], Gunes Acar[2], Claude Castelluccia[1], and Claudia Diaz[2]

[1] INRIA Privatics, Grenoble, France
[2] KU Leuven, ESAT/COSIC and iMinds, Leuven, Belgium

Abstract. We highlight privacy risks associated with the HTML5 Battery Status API. We put special focus on its implementation in the Firefox browser. Our study shows that websites can discover the capacity of users' batteries by exploiting the high precision readouts provided by Firefox on Linux. The capacity of the battery, as well as its level, expose a fingerprintable surface that can be used to track web users in short time intervals.

Our analysis shows that the risk is much higher for old or used batteries with reduced capacities, as the battery capacity may potentially serve as a tracking identifier. The fingerprintable surface of the API could be drastically reduced without any loss in the API's functionality by reducing the precision of the readings. We propose minor modifications to Battery Status API and its implementation in the Firefox browser to address the privacy issues presented in the study. Our bug report for Firefox was accepted and a fix is deployed.

1 Introduction

HTML5 Battery Status API enables websites to access the battery state of a mobile device or a laptop. Using the API, websites can check the battery level of a device and use this information to switch between energy-saving or high-performance modes. All the information exposed by the Battery Status API is available without users' permission or awareness.

The *"Security and privacy considerations"* section of the W3C specification that describes the Battery Status API, states the following: *"The information disclosed has minimal impact on privacy or fingerprinting, and therefore is exposed without permission grants"* [14]. Our findings, however, show that

the API, as implemented by the Firefox browser on GNU/Linux operating system, enables fingerprinting and tracking of devices with batteries in short time intervals.

As of June 2015, Firefox, Chrome and Opera are the only three browsers that supported the Battery Status API [3]. Although the potential privacy problems of the Battery Status API were discussed by Mozilla and Tor Browser developers as early as in 2012 [1, 2, 22], neither the API, nor the Firefox implementation, has undergone a major revision. We hope to draw attention to this privacy issue by demonstrating the ways to abuse the API for fingerprinting and tracking.

We present an analysis of Battery Status API as implemented by Firefox on GNU/Linux. Our analysis indicate that seemingly innocuous information provided by the Battery Status API can serve as a tracking identifier when implemented incorrectly.

The core contributions of this work are:

1. *We present a new device fingerprinting vector based on the Battery Status API.* We show that the Firefox's implementation of the Battery Status API allows the discovery of battery's capacity, provides short-term identifiers that facilitates tracking and potentially can be used for reinstantiating identifiers (*respawning*).
2. *We propose a solution that reduces the Battery Status API's fingerprintable surface* by rounding the level readings provided by the API. Our fix does not cause any loss in the effective functionality of the API. We filed a bug report for Mozilla Firefox to communicate the problem and the proposed solution [20]. The fix was quickly implemented and deployed by Mozilla engineers in response to our bug report.

2 Related work

The Panopticlick [9] study by Eckersley demonstrated the feasibility of browser fingerprinting for online tracking by measuring the entropy present in the browser properties such as screen size, list of system fonts and browser plugins. Other researchers demonstrated the many ways browsers can be fingerprinted using different properties, such as clock skew [13], font metrics [10], network protocol characteristics [7], JavaScript engine performance [16], WebGL and canvas rendering [17].

Recently, studies measured the prevalence of the browser fingerprinting on the Web [4,5,19], suggesting that questionable practices such as proxy circumvention

or stealthy techniques to exercise browser fingerprinting are commonly used by the websites.

In a similar vein, researchers studied *zombie cookie* (or *evercookie*) which is another tracking mechanism that can be used to reconstruct tracking identifiers - even if the user decides to clear her history [12] - with the use of Flash cookies [21], ETags [6] and other vectors.

A recent work, independent from ours, includes a very short note about the possible use of Battery API as a potential privacy risk vector [18]. The problem is not further described or analyzed, and the authors only mention potential risks due to monitoring of charging and discharging rates. In essence, our analysis is more extensive and detailed. Moreover, we describe a clear risk in relation to Firefox browser and study it in detail.

3 Background

3.1 Battery Status API

World Wide Web Consortium's (W3C) Battery Status API allows the reading of battery status data. Among the offered information are the current battery level and predicted time to charge or discharge. The respective properties `level`, `chargingTime` and `dischargingTime` can be accessed in JavaScript by first calling the `navigator.getBattery()` method [3] to get a `BatteryManager` object which then exposes these properties.

The API does not require user permission to read the battery information, any website or third-party scripts included on them, can use the API. The API also does not require browsers to notify users when the battery information is accessed. That allows website and third-party scripts to access the battery information transparently - without users' awareness.

The Battery Status API also provides JavaScript event handlers that allow the monitoring of updates to battery status. The API defines the `level` property as a double-precision floating-point number, taking values between 0 (depleted) and 1.0 (full) [14].

[3] Firefox does not implement `navigator.getBattery()` method, instead, it exposes a `navigator.battery` object.

3.2 Power information under Linux

In our exploratory survey of the Battery Status API implementations, we observed that the battery level reported by the Firefox browser on GNU/Linux was presented to Web scripts with *double* precision. An example battery level value observed in our study was 0.9301929625425652. We found that on Windows, Mac OS X and Android, the battery level reported by Firefox has just two significant digits (e.g. 0.32).

Analyzing the Firefox source code, we found out that the battery level is read from *UPower*, a Linux tool allowing the access to the UPower daemon [11]. The UPower daemon provides access to comprehensive power-management data about the device. Specifically, it enables the access to detailed information about the battery status such as capacity, level, voltage and provides estimates about the discharge and charge times.

Analyzing the UPower source code (`linux/up-device-supply.c`) to understand how it computes the battery level, we compiled the following equations:

$$BatteryLevel = 0.01 \times Percentage \tag{1a}$$

$$Percentage = 100.0 \times \frac{Energy}{EnergyFull} \tag{1b}$$

$$Energy = \frac{ChargeNow}{1,000,000} \times DesignVoltage \tag{1c}$$

$$EnergyFull = \frac{ChargeFull}{1,000,000} \times DesignVoltage \tag{1d}$$

The *Energy* is the current amount of energy present in the battery and measured in *watt-hours*. *EnergyFull* is also measured in *watt-hours* and represents the maximum possible amount of energy that can be stored in the battery. The *ChargeNow* and *ChargeFull* are measured in μAh and represent the current and maximum charge capacities of the battery respectively. Note that, due to the aging of the battery, *EnergyFull* tend to be lower than the design capacity of the battery, moreover, it can also change after a discharge, followed by a full charge – possibly for calibration purposes. Although many batteries share the same design capacities (e.g. 48.84 Wh or 62.16 Wh), as they age in time, their capacities may be reduced in different amounts, resulting in a diverse number of possible *EnergyFull* values, which are internally stored with four decimal places (e.g. 42.1678).

Since Firefox browser under Linux is accessing the UPower-provided data, it reads the *Percentage* value in 64 bit double precision floating point format and multiplies it by 0.01 to obtain the battery *level* as shown in 1(a). The *level* value is then exposed to website scripts through the Battery Status API in double precision.

As noted above, the *EnergyFull* value may change, as the battery capacity degrades. The UPower daemon updates the current capacity by comparing the *EnergyFull* to the latest value stored when the battery is fully charged.

4 Tracking with the Battery Status API

We measure the extent to which it is possible to link (and track) a device with battery using the battery level and charge/discharge time readouts. We observe how it could be leveraged for fingerprinting and tracking across sites. Moreover, we present a method to recover the battery's effective capacity (*EnergyFull*) using the precise battery level readouts provided by Firefox on Linux.

4.1 Tracking across sites

In this section, we discuss several potential fingerprinting and tracking scenarios. A third-party script that is present across multiple websites can link users' visits in a short time interval by exploiting the battery information provided to Web scripts. In order to do that, scripts can use the values of battery level, dischargingTime and chargingTime. The readings will be consistent on each of the sites, because of the fact that the update intervals (and their times) are identical. This could enable the third-party script to link these concurrent visits. Moreover, in case the user leaves these sites but then, shortly afterwards, visits another site with the same third-party script, the readings would likely be utilized to help in linking the current visit with the preceding ones.

Below we analyze more specific cases.

Frequency of battery status changes

We analyzed the update rates under different computing loads (such as watching a movie, simply browsing the Web, etc).

We tested the rate of these changes by setting up a simple page and registering JavaScript event handlers for battery status changes; we monitored JavaScript

readouts of level and dischargingTime, as well as the timestamps of these events. We analyzed the collected data for relative time differences between level, chargeTime and dischargeTime changes. The results indicated that for about 30 s, battery status may serve as a static identifier, allowing (e.g.) a third-party script to link visits from the same computer in short time intervals.

Number of possible identifiers

In our test setting, the lowest indication of dischargeTime we observed was 355 (in seconds), and highest 40277 s. Assuming all the values spanning a range $(355, 40277)$ are possible, this gives 39922 numbers. We can also assume that users seeing a near-drained battery generally connect their notebooks to AC power. Assuming users start to charge their devices when the battery level is 0.1, this leaves 90 available battery level states (0.11 to 1.0). The number of potential levels denoted by a tuple $(level, dischargeTime)$ would then be a simple multiplication 90×39922 and the final number of possible states would be 3592980, which only accounts for the discharging state. Using the information about the battery charge ($chargingTime$) could effectively double the number of possible states. The probability of a $(level, dischargeTime)$ collision (between different users, and assuming a uniform distribution) is therefore low and for a short time frame this would effectively be a unique identifier.

However, we emphasize that the dischargeTime levels can be subject to frequent changes, in response to change in the users' computer use patterns. This means that, in practice, the risk of long-term tracking with this information may be negligible. Moreover, depending on the battery level, some chargeTime or dischargeTime values may not be observed in practice [4]. Yet, the available combinations could be used to distinguish users behind a NAT (Network Address Translation). In such a setting, the computers may have similar fingerprints [9] and often identical public IP addresses. The readouts from the battery may allow distinguishing these users.

Reconstructing user identifiers in short-time intervals

Users who try to re-visit a website with a new identity may use browsers' private mode or clear cookies and other client side identifiers. When consecutive visits are made within a short interval, the website can link users' new and old identities by exploiting battery level and charge/discharge times. The website can then reinstantiate users' cookies and other client side identifiers, a method

[4] For instance, 355 s dischargeTime may be too short for a full battery or, 40277 s dischargeTime may be too long for a battery with level 0.1.

known as *respawning* [21]. Note that, although this method of exploiting battery data as a linking identifier would only work for short time intervals, it may be used against power users who can not only clear their cookies but can go to great lenghts to clear their evercookies.

5 Detecting battery capacity

In addition to using battery level and charge/discharge times for linking visits in short time intervals, Battery Status API can be used to infer the current battery capacity (*EnergyFull*) of a device if it allows high precision level readouts. In this section, we analyze the possibility of fingerprinting a device by exploiting high precision battery level readouts provided by the Firefox on Linux operating system.

We found that using the 64-bit double precision floating point battery level readouts from Firefox on Linux, it is possible to discover the value of *EnergyFull*, which signifies the actual battery capacity. We emphasize that our method only works for UPower and Firefox on Linux, and during our study we encountered some computers for which we cannot recover the capacity with our method. This can be due to the differences in how processors handle floating point calculations [5] or measurement errors in UPower.

The attack works by using the equations *1a-1d* by reading the battery level and finding candidate *Energy*, *EnergyFull* and *Voltage* levels which may give this floating point number reading. In order to do this, attacker may either brute-force the candidate values by testing all possible values or precompute a lookup table.

5.1 Test method

Assuming a uniform space of EnergyFull values (X, Y), we tested all the hypothetical level readouts to detect the possible identifiers. It is obvious that, for a given level reading, several possibilities for EnergyFull level may exist. However, if the attacker has access to multiple battery level readouts, the number of collisions becomes significantly smaller. We analyzed the number of potential EnergyFull candidates as a function of the battery level readouts.

In other words, we computed the number of collisions for one battery level readout, $State_1$. For each such possible readout level, we simulated another

[5] See, for example, [8, 15] on the "floating-point determinism problem."

different readout level (different than the one in preceding state), $State_2$ (battery levels in $State_1$ and $State_2$ are different). We compared the candidate EnergyFull values in $State_1$ and $State_2$ and intersecting the sets of possible EnergyFull levels, we effectively decreased the number of candidate EnergyFull values. The number of EnergyFull candidates for a total of 1559 battery level readings are displayed on Figure 1. Figure 2, on the other hand, shows the reduction of EnergyFull candidates when a script is able read the battery level multiple times from the same device. The figure is based on 1559 battery level readings collected from a laptop running Ubuntu 12.04 operating system. We highlight that such analysis is made possible due to the fixed space a floating-point value can represent, and relatively limited capacities of batteries used in practice [6].

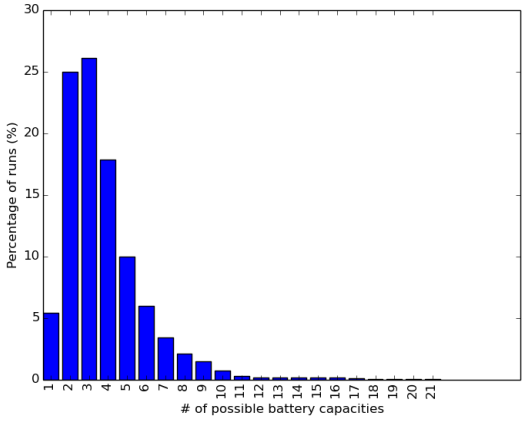


Figure 1: Distribution of number of candidate battery EnergyFull values for a total of 1559 battery level readings (runs). In 5% of the cases the attacker can detect the battery capacity with just one reading.

6 Defense

In the following subsections we outline possible defenses against the exploitation of the Battery Status API for fingerprinting and tracking.

[6] Observe that, possible capacities in this calculations include the reduced battery capacities (e.g. not limited to battery capacities on the market). Still, we could find the candidate capacities on a off-the-shelf computer without a significant computation overhead. We believe, an adversary with moderate storage resources can easily build a lookup table to further reduce the computation time.

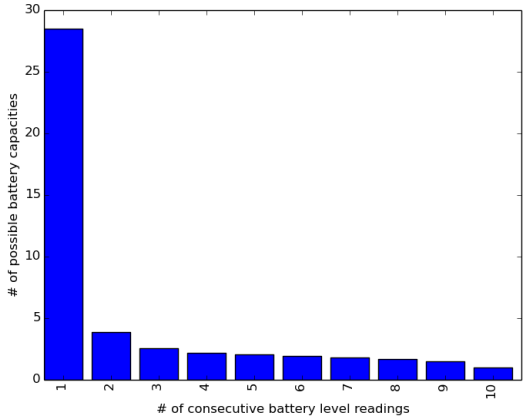


Figure 2: Average number of candidate battery EnergyFull values as a function of consecutive battery level readings. Attacker can significantly reduce the number of candidate battery capacities if he can read the battery level multiple times.

6.1 Limiting the precision of level readouts

In order to limit the tracking and fingerprinting potential of the Battery Status API, the implementations should avoid providing high-precision values. By simply rounding the `level` value of the battery, the threat would be minimized, without losing any functionality of the API. This comment especially applies to platforms where the OS provides high-precision read-outs about the battery.

We filed an appropriate bug report to Firefox implementation, pointing out the inconsistency of level reporting across different platforms [20]. The fix was implemented and deployed as of June 2015.

Moreover, we believe the Battery Status API could mention the risk of exposing high precision readouts in the “Security and privacy considerations” section of the standard. We plan to communicate the results of the study to the editors of the API.

6.2 Asking for user permission to access the Battery Status API

We also discussed potential scenarios where even the reduced precision of the level readout and charge/discharge times could constitute a tracking identifier in short time intervals. In these scenarios, the exposed battery information may allow an attacker to reinstantiate tracking identifiers in a manner similar to evercookies.

In order to prevent this, browser vendors might require user permissions for accesing the Battery Status API. Although this has been suggested by some concerned Mozilla developers [2], final decision was to make the API available without permissions. We believe, as a minimum, users should be able to choose to be asked for battery access by Web scripts. As an alternative, browsers can enforce the user permission requirement in their private browsing modes.

To the best of our knowledge, the only browser that has a strong defense against fingerprinting by the Battery Status API is Tor Browser. Tor Browser completely disables the API [22] to thwart possible fingerprinting attempts.

Finally, the information on the API use could be made available to the user to aid transparency. We are advocating for streamlining the information to users, either directly via the browser's user interfaces, or at least by allowing to read the respective information by custom-made browser extensions. In this way, software could allow the users to learn and be aware about the use of the battery information on devices they own.

7 Conclusion

We analyzed the privacy implications of the Battery Status API, with a focus on its implementation in Firefox for Linux operating system. Our analysis shows that the high precision battery level readings provided by Firefox can lead to an unexpected fingerprinting surface: the detection of battery capacity.

In short time intervals, Battery Status API can be used to reinstantiate tracking identifiers of users, similar to evercookies. Moreover, battery information can be used in cases where a user can go to great lenghts to clear her evercookies. In a corporate setting, where devices share similar characteristics and IP addresses, the battery information can be used to distinguish devices behind a NAT, of traditional tracking mechanisms do not work.

The analysis of Web standards, APIs and their implementations can reveal

unexpected Web privacy problems by studying the information exposed to Web pages. The complex and sizable nature of the new Web APIs and their deeper integration with devices make it hard to defend against such threats. Privacy researchers and engineers can help addressing the risks imposed by these APIs by analysing the standards and their implementations for their effect on Web privacy and tracking. This may not only provide an actionable feedback to API designers and browser manufactureres, but can also improve the transparency around these new technologies.

References

[1] Proposal for a smaller battery API . `https://groups.google.com/d/topic/mozilla.dev.webapi/6gLD78z6ASI/discussion`, 2012. Accessed: 24.6.14.

[2] Why is the battery API exposed to unprivileged content? `https://groups.google.com/forum/#!topic/mozilla.dev.webapi/V361K7c0olQ/discussion`, 2012. Accessed: 26.3.14.

[3] Battery Status API - Can I use... Support tables for HTML5, CSS3, etc. `http://caniuse.com/#search=battery`, 2014. Accessed: 24.6.14.

[4] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The Web never forgets: Persistent tracking mechanisms in the wild. In *21st ACM Conference on Computer and Communications Security (CCS)*, pages 674–689. ACM, 2014.

[5] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. FPDetective: Dusting the Web for fingerprinters. In *20th ACM Conference on Computer and Communications Security (CCS)*, pages 1129–1140. ACM, 2013.

[6] 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.

[7] Yi-Chao Chen, Yong Liao, Mario Baldi, Sung-Ju Lee, and Lili Qiu. OS Fingerprinting and Tethering Detection in Mobile Networks. pages 173–179, 2014.

[8] Bruce Dawson. FloatingPoint Determinism | Random ASCII. `https://randomascii.wordpress.com/2013/07/16/floating-point-determinism/`, 2013. Accessed: 31.8.15.

[9] Peter Eckersley. How unique is your web browser? In *Privacy Enhancing Technologies*, pages 1–18. Springer, 2010.

[10] David Fifield and Serge Egelman. Fingerprinting Web Users through Font Metrics. In *Financial Cryptography and Data Security (FC)*. Springer-Verlag, 2015.

[11] Richard Hughes. UPower Reference Manual. `http://upower.freedesktop.org/docs/`, 2010. Accessed: 22.6.14.

[12] Samy Kamkar. Evercookie. `http://samy.pl/evercookie`, 2010. Accessed: 24.6.14.

[13] Tadayoshi Kohno, Andre Broido, and Kimberly C Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, 2005.

[14] Anssi Kostiainen and Mounir Lamouri. Battery Status API. `https://www.w3.org/TR/battery-status/`, 2012. Accessed: 24.4.14.

[15] David Monniaux. The pitfalls of verifying floating-point computations. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 30(3):12, 2008.

[16] 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.

[17] Keaton Mowery and Hovav Shacham. Pixel perfect: Fingerprinting canvas in HTML5. In *Web 2.0 Workshop on Security and Privacy (W2SP)*. IEEE, 2012.

[18] Gabi Nakibly, Gilad Shelef, and Shiran Yudilevich. Hardware fingerprinting using HTML5. *CoRR*, abs/1503.01408, 2015.

[19] 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 (SP), 2013 IEEE Symposium on*, pages 541–555. IEEE, 2013.

[20] Lukasz Olejnik. Bug 1124127 - Round Off Navigator Battery Level on Linux. `https://bugzilla.mozilla.org/show_bug.cgi?id=1124127`, 2015. Accessed: 30.2.15.

[21] 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.

[22] Tor Bugs: TorBrowser Bundle. #5293 Hook charging+discharching rates in Battery API . https://trac.torproject.org/projects/tor/ticket/5293, 2012. Accessed: 24.6.14.

Publication

Leaky Birds: Exploiting Mobile Application Traffic for Surveillance

Publication Data

Vanrykel, E., Acar, G., Herrmann, M., and Diaz, C. Leaky Birds: Exploiting Mobile Application Traffic for Surveillance. In *Financial Cryptography and Data Security - 20th International Conference, FC 2016* (2016), Lecture Notes in Computer Science, Springer-Verlag, pp. 1–22

Contributions

- As one of the main authors, I helped with the design of the study. I did most of the data collection. I also contributed to the analysis of the data and plotting of the figures.

Leaky Birds: Exploiting Mobile Application Traffic for Surveillance

Eline Vanrykel[1], Gunes Acar[2], Michael Herrmann[2], and Claudia Diaz[2]

[1] KU Leuven, Leuven, Belgium
eline.vanrykel@gmail.com
[2] KU Leuven ESAT/COSIC, iMinds, Leuven, Belgium
{name.surname}@esat.kuleuven.be

Abstract. Over the last decade, mobile devices and mobile applications have become pervasive in their usage. Although many privacy risks associated with mobile applications have been investigated, prior work mainly focuses on the collection of user information by application developers and advertisers. Inspired by the Snowden revelations, we study the ways mobile applications enable mass surveillance by sending unique identifiers over unencrypted connections. Applying passive network fingerprinting, we show how a passive network adversary can improve his ability to target mobile users' traffic.

Our results are based on a large-scale automated study of mobile application network traffic. The framework we developed for this study downloads and runs mobile applications, captures their network traffic and automatically detects identifiers that are sent in the clear. Our findings show that a global adversary can link 57% of a user's unencrypted mobile traffic. Evaluating two countermeasures available to privacy aware mobile users, we find their effectiveness to be very limited against identifier leakage.

1 Introduction

Documents that have been revealed by the former NSA contractor Edward Snowden shed light on the massive surveillance capabilities of the USA and UK intelligence agencies. One particular document released by the German newspaper *Der Spiegel* describes the ways in which traffic of mobile applications (apps) is exploited for surveillance [16]. The document, which reads

"Exploring and Exploiting Leaky Mobile Apps With BADASS," provides a unique opportunity to understand the capabilities of powerful network adversaries. Furthermore, the document reveals that identifiers sent over unencrypted channels are being used to distinguish the traffic of individual mobile users with the help of so-called *selectors*. Similar revelations about the use of Google cookies to target individuals imply that BADASS is not an isolated incident [12, 34].

While it is known that a substantial amount of mobile app traffic is unencrypted and contains sensitive information such as users' location or real identities [24, 35, 43], the opportunities that mobile traffic offers to surveillance agencies may still be greatly underestimated. Identifiers that are being sent in the clear, may allow the adversary to link app sessions of users and thus to learn more information about the surveilled users than he could without. The purpose of this study is to evaluate this risk and to quantify the extent to that it is possible to track mobile app users based on unencrypted app traffic.

To this end we present a novel framework to quantify the threat that a surveillance adversary poses to smartphone users. The framework automates the collection and analysis of mobile app traffic: it downloads and installs Android apps, runs them using Android's *The Monkey* [18] tool, captures the network traffic on cloud-based VPN servers, and finally analyzes the traffic to detect unique and persistent identifiers. Our framework allows large-scale evaluation of mobile apps in an automated fashion, which is demonstrated by the evaluation of 1260 apps. We choose the apps among all possible categories of the Google Play store and of different popularity levels.

Our study is inspired by a recent work by Englehardt *et al.* [26]. They studied the surveillance implications of cookie-based tracking by combining web and network measurements. The evaluation method they use boils down to measuring the success of the adversary by the ratio of user traffic he can cluster together. We take a similar approach for automated identifier detection but we extend their work to capture non-cookie-based tracking methods that are suitable for user tracking. Moreover, we show how TCP timestamp-based passive network fingerprinting can be used to improve the clustering of the traffic and may allow to detect the boot time of Android devices.

1.1 Contributions

Large-scale, automated study on surveillance implications of mobile apps. We present an automated analysis of 1260 Android apps from 42 app categories and show how mobile apps enable third party surveillance by sending unique identifiers over unencrypted connections.

Table 1: Unique smartphone identifiers present on Android, an overview.

Name	Persistence	Permission
Android ID	until a factory reset	None
MAC Address	lifetime of the device	`ACCESS_WIFI_STATE`
IMEI	lifetime of the device	`READ_PHONE_STATE`
IMSI	lifetime of the SIM card	`READ_PHONE_STATE`
Serial number	lifetime of the device	None [41]
SIM serial number	lifetime of the SIM card	`READ_PHONE_STATE`
Phone number	lifetime of the SIM card	`READ_PHONE_STATE`
Google Advertising ID	until reset by the user	`ACCESS_NETWORK_STATE`, `INTERNET`

Applying passive network fingerprinting for mobile app traffic exploitation. We show how a passive network adversary can use TCP timestamps to significantly improve the amount of traffic he can cluster. This allows us to present a more realistic assessment of the threat imposed by a passive adversary. Further, we show how an adversary can guess the boot time of an Android device and link users' traffic even if they switch from WiFi to 3G or vice versa.

Evaluation of the available defenses for privacy aware users. We analyze the efficacy of two mobile ad-blocking tools: Adblock Plus for Android [13] and Disconnect Malvertising [14]. Our analysis shows that these tools have a limited effect preventing mobile apps from leaking identifiers.

2 Background and Related Work

Android apps and identifiers. Android apps and third-parties can access common identifiers present on the smartphone, such as MAC address, Google Advertising ID or IMEI number. We call these identifiers *smartphone IDs*. An overview of the Android smartphone IDs can be found in Table 1. Developers may also choose to assign IDs to users (instead of using smartphone IDs), for identifying individual app installations or simply to avoid asking for additional permissions [11]. We refer to such identifiers as *app assigned IDs*.

Privacy implications of mobile apps. Although privacy implications of

Android apps have been extensively studied in the literature [25, 28, 29], prior work has mainly focused on the sensitive information that is collected and transmitted to remote servers. Xia et al. showed that up to 50% of the traffic can be attributed to the real names of users [43]. Enck et al. developed TaintDroid [25], a system-wide taint analysis system that allows runtime analysis and tracking of sensitive information flows. While it would be possible to use TaintDroid in our study, we opted to keep the phone modifications minimal and collect data at external VPN servers. This allows us to have a more realistic assessment of application behavior and adversary capabilities.

Our work differs from these studies, by quantifying the threat posed by a passive network adversary who exploits mobile app traffic for surveillance purposes. We also show how the adversary can automatically discover user identifiers and use passive network fingerprinting techniques to improve his attack.

Passive network monitoring and surveillance. Englehardt et al. [26] show how third-party cookies sent over unencrypted connections can be used to *cluster* the traffic of individual users for surveillance. They found that reconstructing 62-73% of the user browsing history is possible by passively observing network traffic.

In addition to using identifiers to track smartphones, an eavesdropping adversary can use passive network fingerprinting techniques to distinguish traffic from different physical devices. Prior work showed that clock skew [31, 33, 44], TCP timestamps [23, 42] and IP ID fields [21] can be used to remotely identify hosts or count hosts behind a NAT. In this study, we use TCP timestamps to improve the linking of users' mobile traffic in short time intervals. We assume the adversary to exploit TCP timestamps to distinguish traffic of users who are behind a NAT. Moreover, we demonstrate how an adversary can discover the boot time of an Android device by exploiting TCP timestamps.

3 Threat Model

In this paper we consider passive network adversaries whose goal is to link app traffic of smartphone users. The adversaries observe unique identifiers that are being transmitted from mobile apps in the clear and apply network fingerprinting techniques. We consider that the adversaries cannot break cryptography or launch MITM attacks such as SSLstrip [32].

We distinguish between two types of passive adversaries: A *global passive adversary*, who can intercept all Internet traffic at all time; and a *restricted*

passive adversary who can only observe a limited part of the network traffic. Both adversaries have the capability to collect bulk data. This may be achieved in various ways, such as tapping into undersea fiber-optic cables; hacking routers or switches; intercepting traffic at major Internet Service Providers (ISP) or Internet Exchange Points (IXP) [3].

There can be several models in which an adversary may have limited access to the user's traffic. In this study we evaluate adversaries whose limitation is either *host-based* or *packet-based*. The host-based adversary is only able to see traffic bound to certain hosts; for example, because the adversary is only able to obtain warrants for intercepting traffic within its own jurisdiction. The packet-based adversary may only have access to a certain point in the Internet backbone and thus miss traffic that is being sent along other routes. For both adversaries, we evaluate the success based on different levels of network coverage (Section 7.2). We simulate partial network coverage by randomly selecting hosts or packets to be analyzed depending on the model. For instance, for the host-based model with 0.25 network coverage, we randomly pick one-fourth of the hosts and exclude the traffic bound to remaining hosts from the analysis.

4 Data Collection Methodology

4.1 Experimental Setup

We present the experimental setup[4] that is used for this paper in Fig. 1. It includes a controller PC, two smartphones and two VPN servers for traffic capture. The main building blocks of our framework are as follows:

Controller PC. The Controller PC runs the software that orchestrates the experiments and the analysis. It has three main tasks: 1) installing apps on the smartphones and ensuring that the experiment runs smoothly, e.g. checking the phone's WiFi and VPN connections, 2) sending SSH commands to the remote VPN servers to start, stop and download the traffic capture, 3) analyzing the collected data.

Smartphones. We conducted our experiments with two *Samsung Galaxy SIII Mini* smartphones running Android version 4.1.2. We *rooted* the phones to

[3] All these methods are feasible, as illustrated by the Snowden revelations [6, 8].

[4] The source code of the framework, as well as the collected data will be made available to researchers upon request.

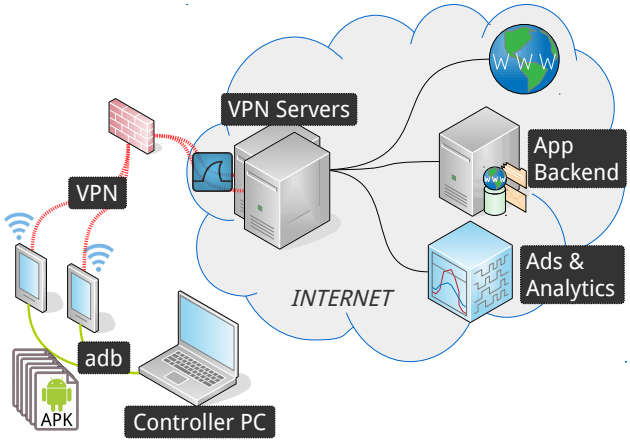


Figure 1: Our setup in this study consists of a Controller PC that manages the experiments, two Android phones that run apps, and two VPN servers that capture the network traffic.

address issues such as storage and uninstallation problems. Although we considered using the Android emulator as in other works [24, 36, 38], our preliminary tests [39] showed that the number of transmitted identifiers is significantly less in the emulator compared to the same setting with a real smartphone and the emulator lacks certain identifiers, such as the WiFi MAC address. We also chose not to intercept system API calls or instrument the operating system, such as in [25, 27], since we preferred a simpler and more portable solution.

The Monkey. We use *The Monkey* [18] tool to automate the experiments and simulate the user interaction at large scale. The Monkey generates a pseudo-random event stream that includes touch, motion and keyboard events.

Traffic Capture. The network traffic is captured by two remote VPN servers, using the *dumpcap* [5] command line tool. Using VPN servers, we could capture *all* the network traffic and not only HTTP traffic, which would be the case with an HTTP proxy. Also, since we record the traffic on remote machines, we ensure that there is no packet drop due to lack of buffer space on resource constrained devices [15]. However, we captured traffic locally on the phone during the evaluation of ad-blockers for Android. These tools use a proxy or VPN themselves to block ads. Since Android does not allow simultaneous VPN connections, we captured the traffic locally by running *tcpdump* on the smartphones. To ensure comparability, we exclude all the captures where we

observed packet drops from the analysis (20% of the cases, 171 apps in two experiments).

Traffic parser. For parsing the captured network traffic, we developed a Python script based on the `dpkt` [3] packet parsing library. The script allows us to decode IPv4 and IPv6 datagrams, reassemble TCP streams, decompress compressed HTTP bodies and to parse GRE and PPTP encapsulation used by the VPN. We extract HTTP headers and bodies, packet timestamps, IP addresses and port numbers from the packets for later use. Since it is outside of the scope of this study, we did not decrypt SSL/TLS records. However, for the TCP timestamp analysis described in Section 6 it is beneficial, yet not necessary, to extract TCP timestamps from all TCP packets, including the ones from encrypted HTTPS traffic. Note that this is within our adversary model, because TCP headers are sent in the clear and thus available to a passive adversary.

Having described the main building blocks of the experimental setup, now we outline the different modes and steps of the experiments:

Experiment modes. We run experiments in two different modes to evaluate the difference in identifier transmission; i) if the app is simply opened and ii) if the user actually interacts with the app. We refer to the former as *startscreen experiment* and to the latter as *interactive experiment*. The Monkey is used to simulate user interaction in the interactive experiments.

Evaluation of ad-blocker apps. We evaluate the effect of apps that block ads and trackers. While those apps are not specifically designed to prevent identifier leakage, they may still reduce the number of identifiers being sent in the clear. Specifically, we repeated the experiment of the top-popularity apps after we installed and activated the ad-blocker apps *Adblock Plus for Android* [13] and *Disconnect Malvertising* [14].

Steps of the experiment. Our framework executes the steps of the experiments in an entirely automated fashion. The Controller PC connects the smartphone to the VPN server by running a Python based *AndroidViewClient* [4] script that emulates the touch events necessary to start the VPN connection on the smartphone. Since installing all the apps at once is not possible due to storage constraints, our framework conducts the experiment in cycles. In each cycle we install 20 apps and then run them sequentially[5]. The apps for each cycle are randomly chosen from the entire set of apps, with the condition that each app is only picked once. Before running an app, the Controller PC kills the process of the previous app. This way we are able to prevent the traffic of the previously tested app mistakenly being recorded for the subsequent app. After

[5]We chose 20 since this was the maximum number of apps that can be installed on an Android emulator at once, which we used in the preliminary stages of the study.

finished running the 20 apps, the Controller PC runs all 20 apps a second time in the same order. Running each app twice enables the automated detection of identifiers outlined in Section 5.1. Finally, the Controller PC completes the current cycle by uninstalling all 20 apps.

4.2 Obtaining Android Applications

To obtain the Android apps, we developed scripts for crawling the Google Play store and, subsequently, to download APK files. Our scripts are based on the Python Selenium [17] library, the `APK downloader` browser extension and webpages [1]. Using this software, we crawled the entire Play Store and obtained information on $1,003,701$ different Android apps. For every app we collected information such as number of downloads, rating scores and app category. This allows us to rank the apps of every category according to their popularity.

For every app category we choose 10 apps from three different popularity levels: *top-popularity*, *mid-popularity* and *low-popularity*. While we use the most popular apps for the top-popularity category, we sample the mid-popularity and low-popularity apps from the 25th and 50th percentiles from each category. At the time we conducted the crawl, there were 42 different app categories and we therefore obtained a total of 1260 ($42 \times 10 \times 3$) apps. The average time for evaluating one app is 64 seconds.

5 Analysis Methodology

In the following we show how an adversary is able to extract identifiers from network traffic and then use these identifiers to cluster data streams, i.e. linking data streams as belonging to the same user. This is the same that an adversary with the goal of surveilling Internet traffic would do, i.e. extracting and applying a set of *selectors* that match unique and persistent mobile app identifiers.

5.1 Identifier Detection

Suitable identifiers for tracking need to be persistent and unique, i.e. the same ID cannot appear on different phones and IDs need to be observable over multiple sessions. Our framework automatically detects such unique identifiers in unencrypted mobile app traffic. While the overall approach is similar to the one in [19, 26] we extend the cookie-based identifier detection technique to cover mobile app traffic. We assume that the smartphone IDs (such as Android ID or MAC address) are not known a priori to the adversary. The adversary has to extract IDs based on the traffic traces only. Yet, we use smartphone IDs as the ground truth to improve our automated ID detection method by tuning its parameters.

For finding identifiers, we process HTTP request headers, bodies and URL parameters. Specifically, the steps of the unique identifier detection are as follows:

- Split URLs, headers, cookie contents and message bodies using common delimiters, such as "=", "&", ":", to extract key-value pairs. Decode JSON encoded strings in HTTP message bodies.
- Filter out cookies with expiry times shorter than three months. A tracking cookie is expected to have a longer expiry period [26].
- For each key-value pair, we construct an *identifying rule set* and add it to the potential identifier list. This is the tuple (host, position, key), where *host* is extracted from the HTTP message and *position* indicates whether the key was extracted from a cookie, header or URL.
- Compare values of the same key between runs of two smartphones.
  - Eliminate values if they are not the same length.
  - Eliminate values that are not observed in two runs of the same app on the same smartphone.
  - Eliminate values that are shorter than 10 or longer than 100 characters.
  - Eliminate values that are more than 70% similar according to the Ratcliff-Obershelp similarity measure [22].
- Add (host, position, key) to the rule set.

We tuned similarity (70%) and length limits (10, 100) according to two criteria: minimizing false positives and detecting all the smartphone identifiers (Table 1)

with the extracted rule set. We experimented with different limit values and picked the values that gave us the best results based on these criteria. A more thorough evaluation of these limits is omitted due to space constraints, but interested readers can refer to [19,26] for the main principles of the methodology.

5.2 Clustering of App Traffic

While the ultimate goal of the adversary is to link different app sessions of the same user by exploiting unique identifiers transmitted in app traffic, the first challenge of the adversary is to identify the traffic of one app. An app may open multiple TCP connections to different servers and linking these connections can be challenging. The user's public IP address is not a good identifier: several users may share the same public IP via a NAT. Moreover, IP addresses of mobile phones are known to change frequently [20].

In this work we consider two different clustering strategies. In the *TCP stream based linking*, the attacker can only link IP packets based on their TCP stream. The adversary can simply monitor creation and tear down of TCP streams and ensure that the packets being sent within one stream are originating from the same phone. The second, more sophisticated strategy employs passive network fingerprinting techniques to link IP packets of the same app session. We will refer this technique as *app session based linking* and outline it in Section 6.

Following Englehardt et al. [26] we present linking of the user traffic as a graph building process. We use the term *node* to refer to a list of packets that the adversary is certain that they belong to the same user. As explained above, this is either a TCP stream or an app session. For every node the adversary extracts the identifying rule set (host, position, key) as described in Section 5.1. Starting from these nodes, the adversary inspects the content of the traffic and then tries to link nodes together to so-called *components*.

Therefore, the attacker will try to match a node's identifiers to the identifiers of the existing components. We account for the fact that some developers do not use the smartphone ID right away as identifier, but derive an identifier from it by hashing or encoding. Thus the clustering algorithm will also try to match the SHA-1, SHA-256, MD5 and murmur3 hashes and base64 encoded form of the identifiers. For every node, there exist three possibilities when comparing the node's identifiers to a existing component's identifiers:

1. **The node's value (or its derivative) matches the identifiers of an existing component:** The node will be added to the component and

the respective identifiers are being merged, i.e. the newly added node may include identifiers not yet included in the component.

2. **None of the node's identifiers or their derivatives can be matched to an existing component:** The node creates its own component which is disconnected from all other components.

3. **The node shares identifiers with multiple components:** These components are merged together and the node is added to this component.

For the remainder of this paper, we refer to the component that contains the highest number of nodes as the *Giant Connected Component* (GCC). Furthermore, we define the ratio of number of nodes in GCC to the number of nodes in the whole graph as the *GCC ratio*. The *GCC ratio* serves as a metric for measuring the adversary's success for linking users' traffic based on the amount of traffic he observes.

5.3 Background Traffic Detection

The Android operating system itself also generates network traffic, for example to check updates or sync user accounts. Although we find in our experiments that the Android OS does not send any identifiers in the clear, we still take measures to avoid that this traffic pollutes our experiment data. Particularly, we captured the network traffic of two smartphones for several hours multiple times without running any app. A complete overview of all HTTP queries made during such captures can be found in [40]. We excluded all the HTTP requests to these domains during the analysis stage. Although we excluded background traffic from our analysis, the adversary may try to exploit the background traffic in a real-world attack.

6 Linking Mobile App Traffic with TCP Timestamps

In this section we elaborate on the adversary's ability to employ passive fingerprinting techniques to link different IP packets originating from the same smartphone. As mentioned in Section 5.2, this gives a significant advantage to the adversary when clustering the user traffic. In particular, the adversary is

able to analyze TCP timestamps for this task as they are commonly allowed by the firewalls [33].

TCP timestamps are an optional field in TCP packets that include a 32-bit monotonically increasing counter. They are used to improve the protocol performance and protect against old segments that may corrupt TCP connections [30]. While the exact usage of TCP timestamps is platform dependent, our inspection of the Android source code and capture files from our experiments revealed that Android initializes the TCP timestamp to a fixed value after boot and uses 100Hz as the timestamp increment frequency [2]. Thus, at any time t, TCP timestamp of a previously observed device can be estimated as follows: $TS_t = TS_{prev} + 100 \times (t - t_{prev})$, where TS_{prev} is the timestamp observed at t_{prev} and $(t - t_{prev})$ is the elapsed time. The adversary can therefore link different visits from the same device by comparing the observed TCP timestamps to his estimate. Prior studies have shown that distinguishing devices behind a NAT using TCP timestamps can be done in an efficient and scalable manner [23, 37, 42].

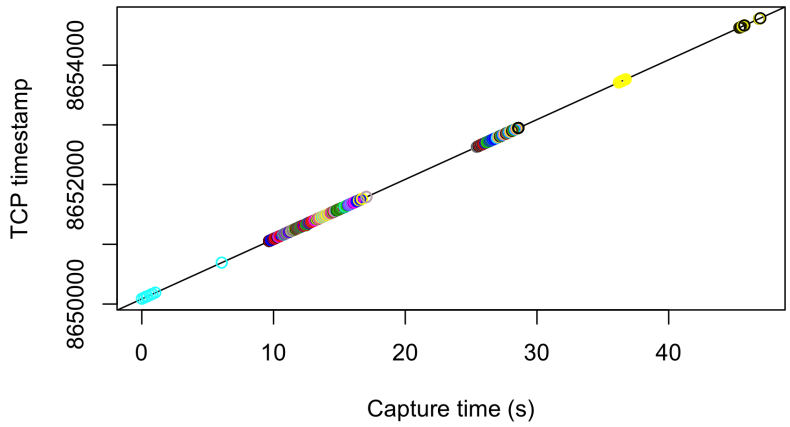


Figure 2: TCP timestamp vs. capture time plot of Angry Birds Space app follows a line with a slope of 100, which is the timestamp resolution used by Android. Different TCP sessions, indicated by different colors, can be linked together by exploiting the linearity of the TCP timestamp values.

Fig. 2 demonstrates the linear increase of the TCP timestamps of a phone running the "Angry Bird Space" app. To demonstrate the linkability of TCP

streams, every point in Fig. 2 is colored based on its TCP source and destination port. The straight line shows that the adversary can easily link different TCP streams of the same device by exploiting the linearity of the timestamps. The adversary is also able to consider TCP timestamps of encrypted communications, because TCP timestamps are sent unencrypted in the packet headers. This allows adversaries within our threat model to further increase the success of the linking. Furthermore, TCP timestamps can be used to link traffic even if the user switches from WiFi to mobile data connection or vice versa [40]. Finally, the linking is still feasible even if the adversary misses some packets, for instance, due to partial coverage of the network.

Limitations. During the background traffic detection experiments, we observed cases in which TCP timestamps are not incremented linearly. Consulting the Android System Clock Documentation, we determined that the CPU and certain system timers stop when the device enters the *deep sleep* state [10]. This power saving mechanism is triggered only when the screen is off and the device is not connected to the power outlet or USB port. Therefore, the phone will never go into deep sleep when a user is interacting with an app and the TCP timestamps will be incremented in a predictable way, allowing the linking of the traffic by app sessions.

Implications for traffic linking. We will assume the adversary can use TCP timestamps to cluster packets generated during the use of an app (app session), as the phone never enters *deep sleep* mode when it is in active use. As mentioned in Section 5.2, we will refer to this as *app session based linking*.

Android boot time detection. In addition to linking packets from different TCP streams, TCP timestamps can also be used to guess the boot time of remote devices [7]. Among other things, boot time can be used to determine if the device is patched with critical updates that require a reboot. Since it is not directly related to traffic linking attack considered in the study, we explain the boot time detection methodology in the unabridged version of this paper [40].

7 Results

7.1 Identifier Detection Rules

We present in Table 2 an overview of the identifying rule set that we detected by the methodology explained in Section 5.1. Recall that identifying rules

Table 2: The extracted ID detection rules and corresponding smartphone IDs. *SID*: Smartphone ID, *AAID*: App Assigned ID.

Exp. Mode	App popularity	Android ID	Google Ad ID	IMEI	MAC	Other SIDs	AAIDs	Total ID Rules
Interactive	top	165	111	63	29	16	193	577
Startscreen	top	115	56	45	19	11	91	337
Interactive	mid	56	28	20	6	5	60	175
Startscreen	mid	48	28	16	5	4	40	141
Interactive	low	73	61	22	15	8	53	232
Startscreen	low	47	24	16	7	8	33	135
Total		504	308	182	81	52	470	1597

correspond to "selectors" in the surveillance jargon, which allow an adversary to target a user's network traffic. In total we found 1597 rules with our method, of which 1127 (71%) correspond to a smartphone ID or its derivative. Our results show that the Android ID and Google Advertising ID are the most frequently transmitted smartphone IDs, accounting for 72% (812/1127) of the total. We group the least commonly transmitted smartphone IDs under the *Other Smartphone IDs* column, which include the following: device serial number, IMSI, SIM serial number and registered Google account email. Furthermore, we found 29% of the extracted rules to be app-assigned IDs.

Analyzing the extracted rules for the top-popularity, interactive experiments, we found that 50% of the identifiers are sent in the URI of the HTTP requests (291 rules). In 39% (225) of the rules, the IDs are sent in the HTTP request body, using the POST method. Only 3% (18) of the cases, the identifier was sent in a cookie. The average identifier length in our rule set is 26.4 characters. A sample of identifying rules is given in Table 3.

After extracting identifier detection rules, we apply them to the traffic captured during the experiments. Due to space constraints we present the detailed results on the transmitted IDs in the unabridged version of this paper [40].

Moreover, analyzing the traffic captures of the top-popularity apps, we found that certain apps send precise location information (29 apps), email address (7 apps) and phone number (2 apps) in the clear. Together with the linking

Table 3: Examples rules found in the constructed identifying rule set. The values are modified to prevent the disclosure of real identifiers of the phones used in the study.

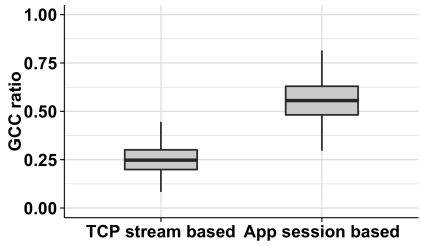
Host	Position	Key	ID	Value
data.flurry.com	Body	offset60	Android ID	AND9f20d23388...
apps.ad-x.co.uk	URI	custom_data / meta_udid	Unknown	19E5B4CEE6F5...
apps.ad-x.co.uk	URI	macAddress	WiFi MAC	D0:C4:F7:58:6C:12
alog.umeng.com	Body	header / device_id	IMEI	354917158514924
d.applovin.com	Body	device_info / idfa	Google Ad ID	0e5f5a7d-a3e4-..

Table 4: The most common third-party hosts found to collect at least an identifier over unencrypted connections. The listed hosts are contacted by the highest number of apps (based on 420 top-popularity apps, interactive experiment).

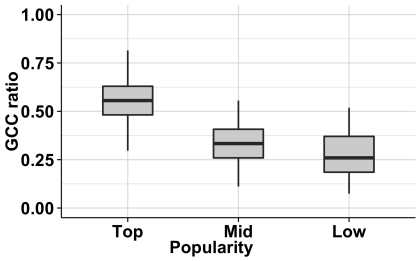
Host	# Apps	Collected IDs
data.flurry.com	43	Android ID
ads.mopub.com	32	Google advertising ID
apps.ad-x.co.uk	22	Google advertising ID, IMEI, Serial number, Android ID
alog.umeng.com	16	IMEI
a.applovin.com	16	Google advertising ID

attack presented in this paper, this allows an adversary to link significantly more traffic to real-life identities.

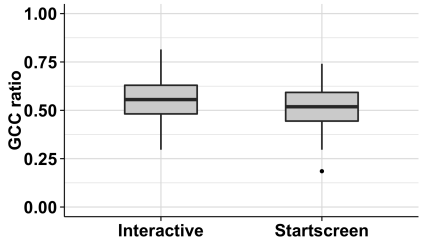
We found that 1076 different hosts were contacted over unencrypted connections during the experiments for the top-popularity apps in the interactive mode. The *data.flurry.com* domain is the most popular third-party domain collecting Android ID from 43 different apps (Table 4). Note that *data.flurry.com* received a notable mention in the slides of the BADASS program [16] for its identifier leakage.



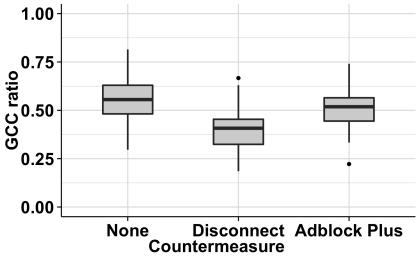
(a) GCC ratio for the top-popularity apps, shown for TCP stream and app session based linking.



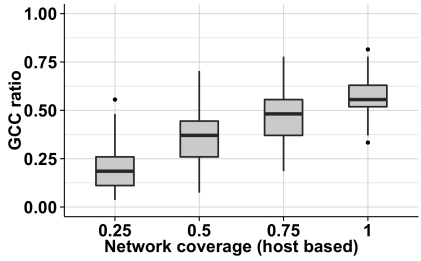
(b) GCC ratio for apps of different popularity levels for interaction mode.



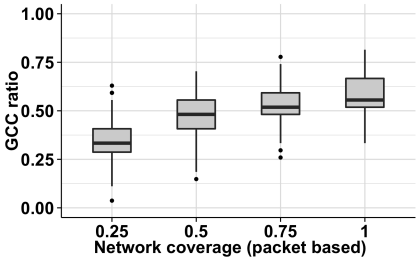
(c) GCC ratio for top-popularity apps, shown for interaction and startscreen mode.



(d) GCC ratio for the top-popularity apps, shown while using different privacy enhancing tools.



(e) GCC ratio for the top-popularity apps, shown for different network coverage levels of a *host based* restricted adversary.



(f) GCC ratio for the top-popularity apps, shown for different network coverage levels of a packet based restricted adversary.

Figure 3: The success of the adversary under different experimental settings and adversary models. The GCC ratio is the proportion of the unencrypted app traffic that the adversary can link together. The results are shown for 100 different randomly selected combinations of 27 apps.

7.2 Traffic Clustering

Here we evaluate the adversary's success in terms of unencrypted app traffic ratio (GCC ratio) that he can link together in different settings. We follow the analysis methodology explained in Section 5.2 and present clustering results for 100 randomly selected combinations of 27 apps. We pick 27 apps since it is the average number of apps used per month according to a recent survey [9]. Running 100 iterations with a different combination of (27) apps allowed us to reduce the variance between different runs and account for all the studied apps. We only consider apps that send at least one HTTP request and calculate the GCC ratio based on the unencrypted traffic. For the top-popular apps in interactive mode, these account for 69% of the apps. For simplicity, we will present the clustering results for only one phone and a single run of each app. The results from two phones did not have any significant difference.

Effect of using TCP timestamps for traffic linking. The left boxplot in Fig. 3(a), shows that when the adversary does not take TCP timestamps into account (TCP stream based linking), he can cluster 25% of users' unencrypted traffic. However, by exploiting TCP timestamps he can increase the GCC ratio to 57%.

Effect of app popularity Fig. 3(b) shows that popularity has a significant impact on the linking success of the adversary. The most popular apps allow the adversary to cluster 57% of the unencrypted traffic, while the apps from the mid-popular and low-popular level result in a GCC ratio of 32% and 28%, respectively.

Due to space constraints, we will only present results for the apps from the top-popularity level in the rest of this section.

Effect of user interaction. Fig. 3(c) shows the GCC ratio for two different experiment modes, *interaction* and *startscreen*. Although the number of identifiers sent in two modes are significantly different (577 vs. 337), the graph shows a similar GCC ratio around 53% for two modes. A possible explanation is that the identifiers that enable linking are already sent as soon as the app is started.

Effect of countermeasures. Fig. 3(d) shows that both ad-blocking apps provide a limited protection against linking of the app traffic. Using Adblock Plus leads to an average linking of 50%. Disconnect Malvertising performs better, with a GCC rate of 38%, reduced from 57%.

Restricted adversary. Fig. 3(e) shows that an adversary that can only intercept traffic to 50% of the hosts can link up to 38% of the unencrypted mobile app sessions. For the *packet based* restricted adversary model, we observe

that an adversary with a limited coverage of 25% of the user's packets can still link 37% of all app sessions together (Fig. 3(f)). In both models restricted adversary's success grows almost linear with his network coverage. Note that packet based restricted adversary can link significantly more traffic than the host-based model for the same network coverage ratio. This may be due to being able to observe packets from more hosts which will allow to link apps across sessions.

8 Limitations

Some apps may not be fully discovered by *The Monkey*, leading to an incomplete view of the network traffic. Also, apps that require user logins may not be sufficiently analyzed by our automated methodology. For those reasons, our results should be taken as lower bounds.

While we assume that the smartphones can be distinguished by their TCP timestamps, some middleboxes may interfere with user traffic. Firewalls, proxies or cache servers may terminate outgoing HTTP or TCP connections and open a new connection to the outside servers. Furthermore, end-user NAT devices may have various configurations and hence behave differently compared to enterprise NATs. In such cases, the adversary's ability to link traffic by TCP timestamps may be reduced.

We used *rooted* Android phones in our experiments. Although rooting the phones may introduce changes in the observed traffic, we assumed the changes to be minimal.

9 Conclusion

The revealed slides of the BADASS program have shown that unencrypted mobile app traffic is exploited for mass surveillance. Identifiers sent in the clear by the mobile applications allow targeting mobile users, linking of their traffic and building a database of their online activities.

In this study, we evaluated the surveillance threat posed by a passive network adversary who exploits mobile app traffic for surveillance purposes. We presented a novel framework that automates the analysis of mobile app network traffic.

Our framework and methodology is designed to be flexible and can be used in other mobile privacy studies with slight modifications.

Our results show that using TCP timestamps and unique identifiers sent in the unencrypted HTTP traffic, a global adversary can cluster 57% of users' unencrypted mobile app sessions. We demonstrated that a passive adversary can automatically build a rule set that extracts unique identifiers in the observed traffic, which serves as a "selector" list for targeting users.

Our results suggest that popular apps leak significantly more identifiers than the less popular apps. Furthermore, while interacting with the app increases the number of leaked identifiers, solely starting an app amounts to the same attack effectiveness.

We evaluated two countermeasures designed to block mobile ads and found that they provide a limited protection against linking of the user traffic. Encrypting mobile app traffic can effectively protect against passive network adversaries. Moreover, a countermeasure similar to HTTPS Everywhere browser extension can be developed to replace insecure HTTP connections of mobile apps with secure HTTPS connections on the fly.

Acknowledgment

Thanks to Yves Tavernier for sharing his valuable insights about middleboxes, and anonymous reviewers for their helpful and constructive feedback. This work was supported by the Flemish Government FWO G.0360.11N Location Privacy, FWO G.068611N Data mining and by the European Commission through H2020-DS-2014-653497 PANORAMIX and H2020-ICT-2014-644371 WITDOM.

References

[1] APK Downloader [Latest] Download Directly | Chrome Extension v3 (Evozi Official). http://apps.evozi.com/apk-downloader/.

[2] Cross Reference: /external/kernel-headers/original/asm-arm/param.h. http://androidxref.com/4.1.2/xref/external/kernel-headers/original/asm-arm/param.h#18.

[3] dpkt 1.8.6.2 : Python Package Index. https://pypi.python.org/pypi/dpkt.

[4] dtmilano/AndroidViewClient. https://github.com/dtmilano/AndroidViewClient/.

[5] dumpcap - The Wireshark Network Analyzer 1.12.2. https://www.wireshark.org/docs/man-pages/dumpcap.html.

[6] GCHQ taps fibre-optic cables for secret access to world's communications. http://www.theguardian.com/uk/2013/jun/21/gchq-cables-secret-world-communications-nsa.

[7] Nmap Network Scanning - Remote OS Detection - Usage and Examples. http://nmap.org/book/osdetect-usage.html.

[8] NSA Prism program taps in to user data of Apple, Google and others. http://www.theguardian.com/world/2013/jun/06/us-tech-giants-nsa-data.

[9] Smartphones: So many apps, so much time. http://www.nielsen.com/us/en/insights/news/2014/smartphones-so-many-apps--so-much-time.html.

[10] SystemClock | Android Developers. http://developer.android.com/reference/android/os/SystemClock.html.

[11] Identifying App Installations | Android Developers Blog. http://android-developers.blogspot.be/2011/03/identifying-app-installations.html, 2011.

[12] 'Tor Stinks' presentation. http://www.theguardian.com/world/interactive/2013/oct/04/tor-stinks-nsa-presentation-document, 2013.

[13] About Adblock Plus for Android. https://adblockplus.org/android-about, 2015.

[14] Disconnect Malvertising for Android. https://disconnect.me/mobile/disconnect-malvertising/sideload, 2015.

[15] Manpage of TCPDUMP. http://www.tcpdump.org/tcpdump_man.html, 2015.

[16] Mobile apps doubleheader: BADASS Angry Birds. http://www.spiegel.de/media/media-35670.pdf, 2015.

[17] Selenium - Web Browser Automation. http://docs.seleniumhq.org/, 2015.

[18] UI/Application Exerciser Monkey | Android Developers. `http://developer.android.com/tools/help/monkey.html`, 2015.

[19] Gunes Acar, Christian Eubank, and Steven Englehardt. The Web Never Forgets: Persistent Tracking Mechanisms in the Wild. *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 2014.

[20] Mahesh Balakrishnan. Where's That Phone?: Geolocating IP Addresses on 3G Networks. *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, pages 294–300, 2009.

[21] Steven M. Bellovin. A Technique for Counting NATted Hosts. *Proceedings of the second ACM SIGCOMM Workshop on Internet measurment - IMW '02*, page 267, 2002.

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

[23] Elie Bursztein. Time has something to tell us about network address translation. In *Proc. of NordSec*, 2007.

[24] Shuaifu Dai, Alok Tongaonkar, Xiaoyin Wang, Antonio Nucci, and Dawn Song. NetworkProfiler: Towards automatic fingerprinting of Android apps. *2013 Proceedings IEEE INFOCOM*, pages 809–817, April 2013.

[25] William Enck, Landon P Cox, Peter Gilbert, and Patrick Mcdaniel. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. *OSDI'10 Proceedings of the 9th USENIX conference on Operating systems design and implementation*, 2010.

[26] 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 *Proceedings of the 24th International Conference on World Wide Web*, pages 289–299, 2015.

[27] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android Permissions Demystified. *Proceedings of the 18th ACM Conference on Computer and Communications Security*, page 627, 2011.

[28] MC Grace, Wu Zhou, X Jiang, and AR Sadeghi. Unsafe Exposure Analysis of Mobile In-App Advertisements. *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, 067(Section 2), 2012.

[29] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. These aren't the droids you're looking for: Retrofitting Android to protect data from imperious applications. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 639–652. ACM, 2011.

[30] Van Jacobson, Robert Braden, Dave Borman, M Satyanarayanan, JJ Kistler, LB Mummert, and MR Ebling. RFC 1323: TCP extensions for high performance, 1992.

[31] Tadayoshi Kohno, Andre Broido, and Kimberly C Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, 2005.

[32] Moxie Marlinspike. New tricks for defeating SSL in practice. *BlackHat DC, February*, 2009.

[33] Steven J Murdoch. Hot or not: Revealing hidden services by their clock skew. In *Proceedings of the 13th ACM conference on Computer and Communications Security*, pages 27–36. ACM, 2006.

[34] Ashkan Soltani, Andrea Peterson, and Barton Gellman. NSA uses Google cookies to pinpoint targets for hacking. `https://www.washingtonpost.com/news/the-switch/wp/2013/12/10/nsa-uses-google-cookies-to-pinpoint-targets-for-hacking/`, 2013.

[35] Ryan Stevens, Clint Gibler, and Jon Crussell. Investigating User Privacy in Android Ad Libraries. *IEEE Mobile Security Technologies (MoST)*, 2012.

[36] Guillermo Suarez-Tangil, Mauro Conti, Juan E Tapiador, and Pedro Peris-Lopez. Detecting targeted smartphone malware with behavior-triggering stochastic models. In *Computer Security-ESORICS 2014*, pages 183–201. Springer, 2014.

[37] Ali Tekeoglu, Nihat Altiparmak, and Ali Şaman Tosun. Approximating the number of active nodes behind a NAT device. In *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on*, pages 1–7. IEEE, 2011.

[38] Alok Tongaonkar, Shuaifu Dai, Antonio Nucci, and Dawn Song. Understanding mobile app usage patterns using in-app advertisements. In *Passive and Active Measurement*, pages 63–72. Springer, 2013.

[39] Eline Vanrykel. Passive Network Attacks on Mobile Applications. Master's thesis, Katholieke Universiteit Leuven, 2015.

[40] Eline Vanrykel, Gunes Acar, Michael Herrmann, and Claudia Diaz. Exploiting Unencrypted Mobile Application Traffic for Surveillance (Technical Report). https://securewww.esat.kuleuven.be/cosic/publications/article-2602.pdf, 2016.

[41] David Weinstein. Leaking Android hardware serial number to unprivileged apps. http://insitusec.blogspot.be/2013/01/leaking-android-hardware-serial-number.html, 2013.

[42] Georg Wicherski, Florian Weingarten, and Ulrike Meyer. IP agnostic real-time traffic filtering and host identification using TCP timestamps. In *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pages 647–654. IEEE, 2013.

[43] Ning Xia, Han Hee Song, Yong Liao, and Marios Iliofotou. Mosaic: Quantifying Privacy Leakage in Mobile Networks. *SIGCOMM '13 Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, (ii):279–290, 2013.

[44] Sebastian Zander and Steven J Murdoch. An Improved Clock-skew Measurement Technique for Revealing Hidden Services. In *USENIX Security Symposium*, pages 211–226, 2008.

Publication

Shopping for privacy: Purchase details leaked to PayPal

Publication Data

Preibusch, S., Peetz, T., Acar, G., and Berendt, B. Shopping for privacy: Purchase details leaked to paypal. *Electronic Commerce Research and Applications 15* (2016), 52–64

Contributions

- I developed part of the data collection setup that intercepts HTTP request and response headers of the browser. I analyzed the tracking related behavior of PayPal. I ran half of the manual experiments, which involved simulating a checkout on about 440 online retail sites.

Shopping for privacy: Purchase details leaked to PayPal *

Sören Preibusch[1] †, Thomas Peetz[2], Gunes Acar[2], and Bettina Berendt[2]

[1] Microsoft Research, UK
mail@soeren-preibusch.de
[2] KU Leuven, Belgium
{name.surname}@kuleuven.be

Abstract. We present a new form of online tracking: explicit, yet unnecessary leakage of personal information and detailed shopping habits from online merchants to payment providers. In contrast to the widely debated tracking of Web browsing, online shops make it impossible for their customers to avoid this dissemination of their data. We record and analyse leakage patterns for the 881 most popular US Web shops sampled from actual Web users' online purchase sessions. More than half of the sites we analysed shared product names and details with PayPal, allowing the payment provider to build up fine-grained and comprehensive consumption profiles about its clients across the sites they buy from, subscribe to, or donate to. In addition, PayPal forwards customers' shopping details to Omniture, a third-party data aggregator with even larger tracking reach than PayPal itself. Leakage to PayPal is commonplace across product categories and includes details of medication or sex toys. We provide recommendations for merchants.

1 Introduction

1.1 Online payment providers process rich transaction data

Online payment handling is a key enabler for electronic retailing and a growing business opportunity as mobile commerce takes off. Contactless payments have

*Preliminary results were presented as a short paper at the Financial Cryptography and Data Security 2015 conference. This article contains the full analysis.

†Corresponding author

been pioneered in successful yet isolated applications, such as public transport (e.g., Oyster in London, touch & travel in Germany) or entertainment (e.g., Disneyland [62], Starbucks [21]). General-purpose digital wallets and near-field payment capabilities are now integrated in all major mobile phone operating systems [20, 27, 42] and promise wider adoption across verticals.

Payment providers are intermediaries between merchants and their customers who buy and then pay for goods and services. As intermediaries, payment providers necessarily gain insight into the transaction as they process personal information, just like the delivery company will need the customer's postal address. The minimum data requirements for payment handling are the order total, the receiving merchant and an authenticated payment instrument. This corresponds to data items traditionally collected during credit card transactions. However, a much richer set of data items becomes available for online, mobile and in-app purchases, including an itemised statement of the goods purchased or information about the buyer, allowing value-added services. Amongst credit card issuers, these data are known as Level II and III but have been rarely available for point-of-sale or transactions [28].

The move towards richer transaction details is driven and enabled by three factors: first, the extended role of payment providers as shopping cart solutions, so that itemised data availability becomes a necessity; second, technically enabled by the lack of data length restrictions found in legacy payment processing; third, the mining of detailed transaction data for fraud detection and prevention [32]. For instance, MasterCard reported acceptance by over 19 million merchants worldwide back in 2001, but only 1% would be able to "capture and transmit Level II and Level III data". These include itemised product descriptions, quantities and prices [39], but still fewer details than what new online payment providers collect.

1.2 Potential benefits of data collection by payment providers

Fraud detection and prevention is the most-publicised benefit of collecting and inspecting purchase details. The rise of riskier card-not-present transactions over the Web or on mobile has mandated new efforts in fighting crime. Between 2002 and 2012, the most recent year for which data is available, the annual fraud losses on UK-issued payment cards has decreased from £427 million to £388 million. Whereas counterfeit, lost or stolen card fraud has decreased from £257m to £97m (-62%) during that period, card-no-present fraud for electronic commerce alone has quintupled from £28m to £140m and now accounts for the majority of losses [9]. Despite continued e-commerce growth, fraud volumes have been decreasing since their peak in 2008. The industry attributes these

accomplishments to automated cardholder address verification and card security codes, to initiatives like MasterCard's SecureCode and Verified by Visa, and to the "effectiveness and sophistication of customer-profiling neural networks that can identify unusual spending patterns" [18]. The required collection of details about buyers and their purchases is therefore attractive for payment providers and merchants who can benefit from lower fees. As another example, the payment provider Klarna allows customers to pay after order placement and shipping. At the same time, it absorbs the credit risk for merchants and controls losses through risk assessment based on diverse factors, including purchase details [70].

Fighting payment fraud is only one of many more applications for purchase information. Payment providers have a twofold incentive to collect details for the transactions they process. One the one hand, they can use the additional data for operational efficiency in a broad sense; on the other hand, they can offer convenience features to consumers.

Operational efficiency. Payment providers operate in a highly regulated environment and some obligations cannot be fulfilled efficiently unless purchase details are known. They must comply with tax and legal requirements, such as products prohibited in certain regions (e.g., gambling, alcohol sales) or money laundering. They must also detect and prevent crime, such as fraud and policy violations. As an example of transaction monitoring, PayPal has "hundreds of highly trained specialists working around the clock to prevent fraudulent activity and identify suspicious transactions" [58]. Details from past transactions are also a shared secret between the provider and its customers, and can be used for additional authentication or account recovery. Purchase details can be monetised for product innovation, as market research, and through direct marketing on an individualised basis. Insofar as payment providers provide escrow services and help buyers who have been defrauded by the merchant, transaction details can be used for risk screening. For instance, PayPal's buyer protection only covers certain physical goods. Whilst mainly in the self-interest of the provider, operational efficiency enables payment services for consumers and merchants at acceptable fees in the long run.

Convenience features. Buyers can enjoy peace of mind when their purchase details are displayed back to them in the very moment when making the payment. They can also inspect the transaction history in their account and get a detailed statement of previous purchases. When payment providers collect purchase details, they can offer sought-after spending reports and financial self-analysis.

1.3 Privacy concerns

The large-scale collection and processing of personal details causes privacy concerns. Concern is no longer limited to traditional items of personal information like address or demographics, but increasingly about consumption behaviour. Despite the quantified-self movement and although Web users volunteer personal information with high prevalence (e.g., 55% knowingly entered their weekly spending behaviour into a Web form where this item was optional [38]), extended records of usage data are problematic. Widespread tracking of browsing patterns by Websites and aggregators has raised attention in mainstream media [75]. Browsing history leaked to advertisers [67], electricity consumption recorded by smart meters [40], or mobility trajectories in pay-as-you-drive insurance policies [64] have all been found to be associated with elevated privacy concerns. Of particular interest is shopping data, whose value is demonstrated through myriads of loyalty card schemes. Purchase tracking now happens across merchants and channels (online / offline) and even if users are not enrolled in a loyalty scheme [13, 72].

Our research looks at the tracking capabilities of payment providers, namely PayPal. An illustrative example is provided in Fig. 1 and Fig. 4.

Our research motivation is the ability of payment providers to collect purchase details at scale. As in the domains of Web tracking and analytics, a small number of providers cover multiple Websites (merchants) and can link transactions across those. Compared to cookie-like tracking, the privacy issues are exacerbated:

- Embedded tracking code is—in principle—ancillary to the core functionality of the Web page and can safely be filtered out (e.g., with ad-blockers or Tracking Protection in Internet Explorer). Payment handling is however essential to shopping, and users cannot complete the transaction without interacting with the payment provider.
- Unlike browsing patterns linked to a cookie identifier, consumption patterns linked to a payment method are not pseudonymous but identifiable through offline details such as credit card numbers or bank account details, which often include full name.
- Payment cards or account information serve as persistent identifiers, allowing longitudinal linkage of multiple transactions even across different logins or accounts with the payment provider.
- Consumers are typically unable to evade such data collection unless they refrain from shopping with the given merchant. The collection of shoppers'

details is a negative externality of the contract between the merchant and the payment provider.

- Payment handling is universal across sellers and sectors. Consumer details are collected and merged across transactions even for sensitive products and merchants. This includes pharmacies or adult entertainment, for instance, where shoppers deliberately moved out of the high street and onto the Web in a pursuit of privacy.

Privacy threats arise from detailed purchase patterns when more than the minimum data required are collected. The principle of data minimisation has long been codified in national law and international privacy guidelines, such as the "collection limitation principle" in the OECD privacy framework [44] or the Madrid Privacy Declaration [74]. With the European Union's upcoming General Data Protection Regulation, data minimisation is now becoming an enforceable principle [10].

1.4 Theoretical background: privacy and e-commerce payment intermediaries

The benefits of data collection by payment providers, but also the associated privacy concerns discussed above, can be interpreted in the general framework of e-commerce intermediaries and their roles.

Different streams of research, including information systems, have extensively discussed the privacy aspects of payment intermediaries between retailers and their customers since the advent of electronic commerce in the early 2000s. Much less effort has been devoted to examining the technical reality and evolving business practices of this tri-party relationship.

Intermediation is a technique to overcome a trust deficit that the seller may not adequately secure the customer's payment details. Using a payment provider rather than processing card details directly thus leads to more conversions for the potentially untrusted merchant [22]. An intermediary can overcome security and privacy concerns that can inhibit online shopping [24]. Whilst the trust boost is the payment providers' most important consumer-facing role, it only applies to sellers of inferior trust than the payment provider. Websites with a longer history on the Internet develop their own brand and exhibit less prominent use of trusted third parties [47]. Even then, merchants continue to benefit from aggregation features and cost savings for set-up and ongoing transactions offered by intermediaries [4, 6].

The technical expertise required to interface with a payment provider, albeit smaller than directly integrating with a credit card acquirer, still depends on which services of the provider are used (Section "Background: PayPal integration and information flows"). Smaller merchants with fewer resources are often encouraged by the providers themselves to start with simple integration methods. Efforts to ensure the security of customer accounts vary across online industries [8], larger, more popular, and more mature Websites have significantly better overall privacy protection [7].

Insofar as payment providers bridge trust gaps, they overcome consumers' privacy concerns vis-à-vis the merchant. New privacy issues are introduced, though, which have been explained above. On an institutional level, intermediaries may be a threat to privacy when shaping the ability to transact or adopt privacy-enhancing solutions: Examples include PayPal blocking customers who use Tor, a privacy-enhancing technology to hide their IP address [36]. PayPal has also withheld donated funds and prevented further donations to WikiLeaks, an activist organisation concerned about transparency and privacy [59]. Guidance issued by European data protection authorities unanimously classifies payment providers as data controllers rather than data processors, acknowledging the control they exercise in re-purposing customer data [26, 45].

1.5 Contribution and research questions

Ahead of tightening regulation regarding data minimisation, recognising that online payment handling is a growing market, noting that information privacy is becoming a positive competitive differentiator, we set out to explore the tracking capabilities of online payment providers.

As the first kind of such investigation, the focus is on exploring and describing current practices. We conducted the first industry-wide, empirical survey that quantifies the flows of customer data from N=881 merchants to PayPal. We describe current practices of data proliferation which can soon be deemed privacy leaks.

PayPal is chosen as the most pervasive online payment provider, covering Websites across strata of popularity [53]. We investigate which items of personal data and which transaction details merchants are sharing with PayPal as customers complete their checkout (Fig. 1 and Fig. 4). Our goal is to quantify the prevalence of data flows towards PayPal and to measure the amount of data shared above pure order totals. Our survey of the ecosystem also looks for per-sector differences in data sharing with payment providers or whether more popular Websites leak more or less personal details.

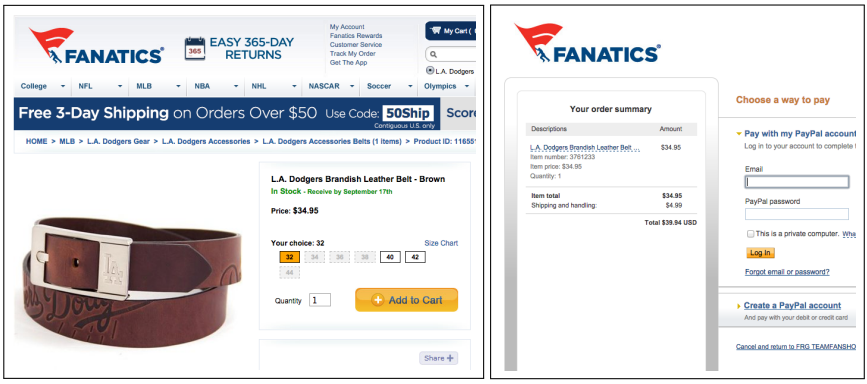


Figure 1: The Web shop passes the product name and item number on to PayPal.

2 Related Work

Our investigation complements and expands an existing body of literature that has empirically examined privacy and tracking practices at large. Bonneau and Preibusch studied privacy practices across the entire online social networking ecosystem [7]. They found unsatisfactory privacy practices throughout the industry, which were still better for more popular and mature sites. They also investigated data protection practices among different industries [8] and found that poor practices were commonplace regarding password security, although merchant sites did better than newspaper sites. Specifically for Web shops, more expensive shops were found to collect significantly more personal details than their cheaper competitors [69].

A number of Web privacy surveys studied the private information leakage, different tracking mechanisms and their prevalence on the Web. Krishnamurthy and Wills show how personally identifiable information leaks via online social networks, including the leakage by HTTP Referer header [34]. Roesner et al. presented a taxonomy of third-party tracking and developed tools for defending against tracking by social sharing buttons [63]. Multiple researchers surveyed the use of more advanced and resilient tracking mechanisms such as evercookies [1, 5, 41, 66], browser fingerprinting [1, 2, 14, 43] and cookie syncing [1, 46], commonly reporting on questionable practices and unexpected prevalence of such technologies.

Researchers studying tracking on mobile platforms found that many apps leak private information to third-party servers including precise location, personal

data and unique identifiers [15, 16, 19, 25]. A study intersecting the interface used for embedding mobile ad libraries found that apps share highly sensitive information such as ethnicity along with postal code, gender, age and income [9]. The same study found a positive correlation between app popularity and the privacy leakage. More recently, by analysing the unprecedented amount of 1.1 million Android apps, Viennot et al. showed how apps mishandling of authentication tokens may lead to unauthorized access to user data and resources on Amazon Web Services and Facebook [73].

Another line of research has looked into users' reaction to online tracking and behavioural advertising. A 2013 Pew Research study found, motivated by the concerns about online tracking, that 86% of Internet users have tried to be anonymous online and took some effort to avoid tracking [61]. Ur et al. found a majority of users in their study were either fully or partially opposed to online behavioural advertising, finding the idea smart but creepy [71]. Leon et al. studied the factors affecting users' willingness to share information with the advertisers and found that perceived sensitivity of information, data-retention policies and the scope of data use are the prominent factors [35].

Finally, researchers looked into consumers' privacy choices in online shopping. Buyers of sensitive products (vibrators) were found to pay a premium to shop with a retailer whose privacy practices were labelled as superior by a product search engine [68]. In the largest ever lab and field experiment in privacy economics, almost one in three Web shoppers paid one euro extra for keeping their mobile phone number private [31]. When privacy comes for free, more than 80% of consumers choose the company that collects less personal information [31]. Earlier results indicated that price discounts override online shoppers' privacy preferences [60].

3 Methodology

We conducted a blind field experiment with 1200 shopping Websites, by observing their inbound and outbound data flows during checkout. The Websites did not know they were subjected to data capture, which followed a strict experimental protocol. The data collection setup fleshes out the integration with PayPal, which is described first. We then provide details on the sampling, the experimental protocol, and describe additional data sources for data enrichment.

3.1 Background: PayPal integration and information flows

PayPal provides payment processing to merchants and has been a pioneer to offer payment acceptance to electronic retailers, albeit its product range now covers a plenitude of card and card-less payment and identity services for online, offline, and mobile transactions. Similar to a cloud service, PayPal's offerings are characterised by their ease of set-up, pay per use, and self-service.

PayPal offers multiple ways to be embedded in the shopping workflow, traditionally depending on the type of payment, for instance (e.g., donations, recurring subscriptions, one-time checkouts) [50]. On a technical level, there are two different integration routes depending on how the session data is transmitted from the merchant to PayPal: (1) server-to-server integration, where SOAP Web services or REST APIs are used to communicate transaction details from the merchant to PayPal; (2) integration via the client, where transaction parameters are passed exclusively through the query string (GET) by means of consumers' browsers.

Integration via GET is simple and readily available for hosted Websites, as no server-side communication is required. In PayPal parlance, this integration method is called "buttons". More sophisticated methods use server-to-server communication between the application server and the payment provider: the merchant creates a session with the payment provider when submitting all relevant transaction data. This session is then referenced through a session identifier or token ("EC token"), which is the only information that the client needs to pass on [49]. This method requires more technical expertise, but is less susceptible to manipulation by the client. However, server-to-server communication cannot be observed in a study like ours, where the client is instrumented. It would require server logs from PayPal or the merchant (or broad-coverage network sniffing capabilities). Also, when integrating with PayPal through payment buttons, merchants can still hide submitted information and prevent tampering by encrypting the transaction parameters [48].

A variant of server-to-server payment integration is the use of a further intermediary that calls and processes the PayPal workflows on behalf of the merchant. Such an intermediary is typically found for more complex integration, to mediate between multiple payment methods.

Whereas encrypted buttons are encountered rarely, payment sessions referenced via an EC token on the client side are very common (Table 3). The unobservable flow of personal information between servers is a challenge for our research. We therefore use personal data that PayPal displays back to the user to establish a lower bound for the privacy invasion by the data that is transmitted; this method

was confirmed to be accurate during data analysis (Section 'Data Analysis').

The "Legal Agreements for PayPal Services" [55] outline a number of requirements for merchants. All information submitted to the API must be "true, correct, and complete" [51]. Whereas all fields containing personal information are optional [56, 57], a "description field to identify the goods" and a URL linking back to the original product page must be provided for the popular Express Checkout method [57].

3.2 Sampling

We sample online shops that target US consumers and provide checkout in US Dollar via PayPal. The US market is chosen for its size and for being the home market of PayPal. We sample popular Web shops. Practices at these online destinations matter most as they impact a large consumer population. Stores are identified by their URL, as occurring before the PayPal checkout page in browser sessions. Data is collected from a sample of Internet Explorer users who opted in to share their browsing history.

Sampling originally yielded Website domains rather than product pages for each potential shop. These domains were ordered and processed in decreasing order of popularity and inspected manually as Websites may have ceased to offer PayPal, may have shut down, may be unreachable or otherwise no longer qualify by our sampling criteria. In particular, sampling by referrer produced false positives, such as Webmail providers or search engines. These sites were noted and excluded. All domains were inspected manually and if the site matched the sampling criteria, a single product was selected to measure data leakage during checkout. Product selection followed a simple protocol, choosing the first available product in the first product category except sale or seasonal categories.

We excluded Websites offering business services (B2B such as email marketing campaigns), banks and insurances, and restricted Websites which required a prior customer relationship such as utility companies. Airline Websites were often excluded for we were unable to complete the purchase according to our data collection protocol. EBay, PayPal internal and duplicate Websites were excluded (e.g., homedepot.ca as a duplicate of homedepot.com).

We deliberately included Websites selling non-material goods such as in-game purchases for extras or virtual currencies, or online account top-ups. In line with our research agenda, we also refrained from filtering out adult Websites.

Hosting sites (e.g., Yahoo! shops or Google Sites) were excluded and separated from the sample for future analysis. Such sites host multiple shops with differing

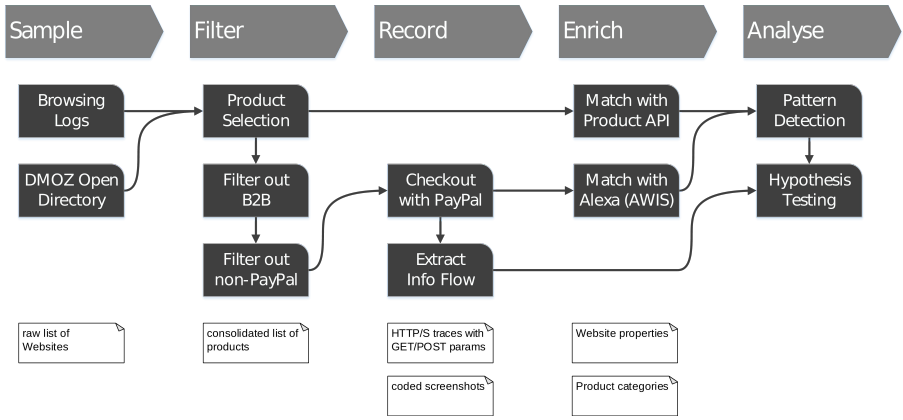


Figure 2: Steps of the data collection process; data collections are given in the lower portion.

implementation practices under a single domain. A few representative sub-shops were chosen for affiliate shops (e.g., spreadshirt.com) and shop-in-shop solutions (e.g., atgstores.com).

3.3 Experimental protocol

Before the main data collection, we ran a pilot study with separate 40 Websites sampled from the DMOZ / Open Directory Project in the (English) electronics Web shops category. This seeding sample covers a broad range of lesser known online retailers, which we inspected manually whether they offer checkout via PayPal or not. Based on this pilot, we then established the following data collection infrastructure and process.

For reliable results, a strict data collection protocol was followed during the main data collection. The details of the experimental setup and procedures are laid out in the Appendix. To avoid contamination of the results by residual cookies or other re-identification methods, a virtual machine was used and reset for every recording anew. Transaction data were recorded by navigating in the Web shop and to PayPal to the point of checkout; browsing was done in Firefox and all HTTP and HTTPS traffic was captured by mitmproxy [11] and stored. This includes GET and POST requests and the parameters submitted with them. Web forms were completed by using the same fictitious profile data

on every site, a woman in her 40s living in a major US city. A unique email address was used for each Website.

The entry point for each Website was the first available product in a product category ('department'). We excluded volatile categories such as featured items or sale promotions. Recording started at the product page identified during sampling and finished with the PayPal checkout screen. An example of this screen is given in Fig. 1. We captured and archived this PayPal screen (screenshot image and page mark-up) and manually tallied the presence of personal and product details displayed on the page. We later tested and confirmed the accuracy of inferring data transfer from the screenshot (Section 'Data Analysis').

Problems that we encountered during the data collection were recorded and dealt with consistently. For broken or unavailable Websites, we progressed as far into the purchase as possible. Unavailable items were substituted by the next available product according to the product sampling procedure. Websites that refused the existing profile data for any reason, were recorded but later excluded from the dataset. Some Websites redirected to a non-US version of the shop based on IP address geo-location, in which case we tried to navigate back to the US store. When possible, we completed the checkout without registering with the shops. Although data collection is tool-supported, there is always a human in the loop.

After the full data analysis was completed, we reached out to a sub-sample of online retailers to explain their point of view.

3.4 Data enrichment: adding metadata

To analyse privacy friendliness by industry, we manually and automatically annotated the Websites in our sample with the kind of service they offer and the products they sell. We also added metrics for popularity and technical quality.

Manual Categorisation of Web shops. We manually categorised the Websites as specialised in retail, commercial services, donations, dating, events, airlines, or other if they did not fit in any of these categories. The commercial services Websites were further subcategorised as educational, software, Website, or other.

Retail Web shops that sell physical goods, while commercial services are selling primarily nonphysical goods such as courses (educational), online-backups (software), or access to a Website (Website). Donation Websites do not provide anything in return for money, and technically need not even register the donor's name. Although small in number, the categories events and airlines were

Table 1: Leaked data by clusters ranked from good to bad privacy practices. The common leakage of product details is more worrying than the seeming absence of customer data: PayPal collects identity details directly during payment. Leaked by: □ = some sites, ■ = all sites, blank = no sites in that cluster.

	Site count	Cluster description	Address	Email	Phone	Shipping	Quantity	Prices	Description	Prod. Name	Leak min	Leak max
C_1	391 (44%)	Leaks nothing or at most one item.	□				□		□		0	1
C_2	34 (4%)	Leaks two of names, item numbers, and prices				□	□	□	□		1	3
C_3	292 (33%)	Leaks at least names, item numbers, and prices.					□	□	□	□	3	4
C_4	155 (18%)	Leaks at least most product details and always shipping costs.				■	□	■	□	□	4	5
C_5	9 (1%)	Leaks name and address in addition to product details.	■	□		□	□	■	□	□	6	7

included because they by nature will require more personal information than a retail store.

Automated Categorisation of Web shops. For a more fine-grained, product-driven categorisation of Web shops, we turned to the curated ontologies of DMOZ and AWIS (Alexa Web Information Service). They both turned out to only have data for the most popular Websites (35% coverage from AWIS keywords, 48% from AWIS categories, 52% from DMOZ—53% when combining DMOZ and AWIS), so we looked for an alternative solution.

Exploiting Amazon.com's status as one of the biggest warehouses of the Internet, we used the Amazon.com Product Advertising API [3] to obtain a better classification. This is a novel approach which has been applied for the first time in our investigation. For every URL in our dataset, we queried the API

Table 2: Manual Website categorisation, including sub-types for commercial services.

Type	Site Count		Subtype	Site Count
Retail	764			
Commercial services	66	→	Website	20
Donations	25		Software	19
Dating	9		Educational	11
Events	8		Other	16
Airlines	6			
Other	3			

Table 3: Distribution of PayPal endpoints used by merchant sites.

	PayPal endpoint URL	Site count (with/ without token)
1	`https://www.paypal.com/cgi-bin/webscr`	847(731/116)
2	`https://www.paypal.com/webscr`	25(18/17)
3	`https://www.paypal.com/cgibin/webscr`	4(4/0)
4	`https://www.paypal.com/checkoutnow`	4(4/0)
5	`https://www.paypal.com/bg/cgi-bin/webscr`	1(1/0)
	Total	881(758/133 : 86% with token)

with the HTML Meta keywords describing the merchant Website. This yielded a list of up to ten matching products, for each of which one or more product categories were listed. Amazon product categories are a graph structure, which can be browsed like a tree, similar to DMOZ. We applied a voting algorithm over the returned items for each Website; the items were sorted in descending relevance by the Amazon API.

The API also assigns exactly one node of Amazon's product type and product group type ontologies to each product. However, not much useful information could be extracted from these, and we largely discarded them from further investigation.

Although the Amazon product ontology was used for describing Website categories, we retained AWIS to assess popularity and technical quality, indicated by the 'traffic rank' and 'speed percentile' respectively.

3.5 Data sources

In summary, we use the following data sources for our analysis:

- A list of online retailers offering PayPal checkout, sampled from actually visited Web shops in real browsing sessions, as per the method described above.
- Website meta-data from external sources: (1) Website popularity, maturity and audience demographics licensed from the Alexa Web Information Service (AWIS) API; (2) manual and automated merchant categorisation via Website keywords and Amazon Product API.
- From captured HTTP traffic to the server: a trace of all personal information sent from the merchant to PayPal via query string or the presence of a server-initiated transaction session, as detected by the submission of a session token. Transcribed screen-shots of the PayPal login page, showing personal data that PayPal displays back to the customer, which was confirmed to equal data received from the merchant.
- From captured HTTP traffic from the server and from saved HTML pages: data on third-party trackers (Omniture) deployed on PayPal's checkout pages.
- Written responses received from the merchants we asked for clarification on their data sharing practices with PayPal.

4 Data Analysis

From an initial list of 1200 shopping domains, we successfully collected the data for 881 merchant Websites: HTTP(S) traffic traces until reaching the PayPal login page, and screenshot upon arrival. These parsed logs and transcribed screenshots were confirmed to be accurate evidence of personal identifiable information (PII) leakage.

First, we describe our data set in terms of PayPal API implementations and predominant patterns of PII leakage. We also explain how we used machine learning techniques to reduce the multitude of these patterns to a manageable number. Second, we explain the data enrichment we performed. We looked for predominant practices on the Internet by adding metadata to our data set, allowing us to slice the data set by Website and product categories.

4.1 Descriptive statistics and cluster analysis

Endpoints and Tokens. As described in the Background section, there are two basic methods for a Web shop to communicate with PayPal: using a token or using GET parameter transmission. For both methods, the logs indicate there exist a number of different PayPal API endpoints for the Web shops to use. Table 3 shows their distribution over our dataset.

The most predominant endpoint (1) is the only one that is currently mentioned in the PayPal API help documents [54]. The second-most used endpoint (2) appears to be an older endpoint; although we have no PayPal document confirming this, there is exactly one mention in the online help. From the context, it stands to reason that this was merely an oversight when updating the documentation. Endpoint 3 seems to exist to catch typos; there is no mention of it in any PayPal documents. Endpoint 4 has a more distinctive name, but is likewise undocumented. Finally, endpoint 5 is a localized version. We see multiple calls to such localized endpoints in the logs, but they generally forward all data to endpoint 1. This Bulgarian endpoint did not, so we included it as a special case.

Token usage is widespread: For endpoint 1, more than 84% of all Websites employ one, and 86% across all endpoints. For such Websites, we have to rely on the screenshots because PII leakage cannot be inferred from the HTTP GET logs.

Data accuracy. The PayPal API accepts various parameters, a subset of which is reserved for the customer PII. While some of this is readily displayed on the PayPal login screen before payment, we also investigated whether any PII was sent to PayPal by parsing the log files. Based on the PayPal API help documents [54], we determined all relevant parameters and parsed the logs for occurrences.

To verify our screenshot-based approach, we investigated whether the PayPal login screen always displays all PII that the API receives over the GET query-string. We were able to confirm that whenever customer or product data was leaked via GET, it showed up on the PayPal login screen. The only exception was for shipping costs of USD 0.00, which was forwarded but hidden in 36 cases. This means the transcribed screenshots give an accurate account of transmitted customer data.

Pure Leakage Patterns. After aggregating the screenshot and log data, three pure leakage patterns emerge. These pure patterns account for a total of 805 URLs (91%). The remaining 76 URLs form a long tail of 25 patterns, none of which occurs more than 13 times, and 15 patterns occur exactly once. With

338 out of 881 sites, the predominant pure pattern leaks no data at all. The other two both leak the item names, descriptions, numbers and the customer's name to PayPal, with the third also informing PayPal about the shipping costs.

Clustering of all Leakage Patterns. The leakage patterns form the backbone of our work. In order to analyse the data more deeply, we shorten the long tail of these patterns and reduce the number of distinct patterns by clustering all 881 URLs into only a few classes.

While the three pure leakage patterns identified above have very intuitive descriptions, a k-Means clustering [17] failed to identify similarly meaningful patterns for any value of k. We thus resorted to EM clustering [12], which automatically determines the appropriate number of clusters. The result is shown in Table 1.

Integration Patterns by Cluster. A natural question is whether a particular combination of endpoint and token usage enforces or prevents leakage. Analysing the clusters, it becomes obvious that there is no such relationship: None of the clusters are homogeneous with respect to endpoints and tokens, except for C_2, which does not contain any token implementations.

Privacy-friendly Websites tend to use a token more often: 98% of all Websites in Cluster C_1 were using a token, compared to 86% and 85% for C_3 and C_4, respectively ($p < 0.0001$, two-tailed Fisher's exact test).

We observe that no Websites leaking customer addresses rely on a token implementation. With a sample size of nine this holds little statistical significance, but we found no indication in the API documentation that this is a requirement on PayPal's side.

We used the association rule mining algorithm Apriori [23] to see whether the cluster membership of a Web shop correlates with its implementation. Requiring a confidence of at least 0.4, the resulting rules did not consistently link the usage of a token or an endpoint to any degree of privacy-friendliness. We conclude that PayPal's available API methods do not bias Web shops to treat customers' privacy in a specific way.

4.2 Leakage patterns by Website category

Table 2 shows a breakdown of the Website categories. The distribution of Website categories per cluster largely reflects the overall distribution over the data set. No clear trends can be identified, although a large majority of donation sites sit unnecessarily in the privacy-unfriendly C_3. Commercial services are also mainly found in this cluster. While the leaked PII is necessary for this

category of Websites, we do not see an immediate need to forward them to PayPal.

Prediction from the category of the Website showed no promise. With the exceptions outlined above, all clusters proved to be too mixed, and Apriori failed to produce rules with a confidence of even 0.4. Hinting towards a variety of attitudes towards privacy among Web shops, this is a positive result for customers.

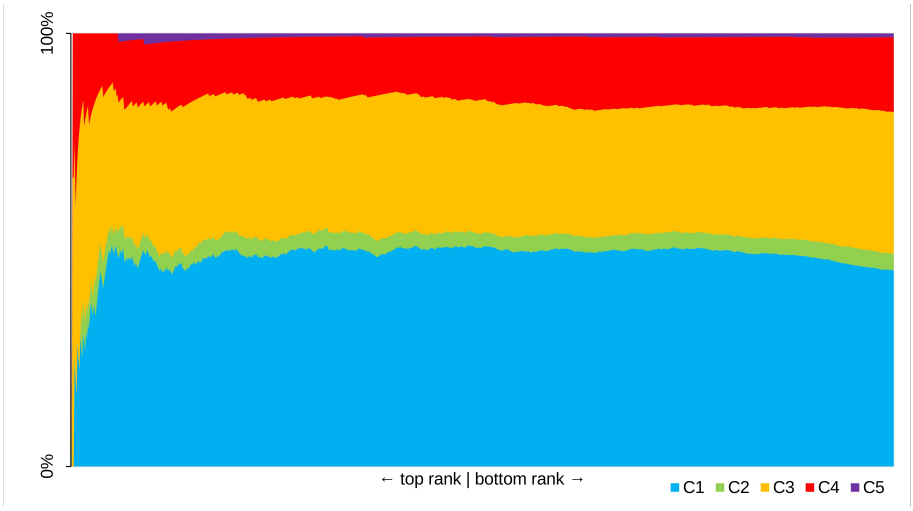


Figure 3: Cluster membership over the sites' popularity (traffic ranks) found in the sample.

Looking at the product categories associated with the Websites found in each cluster, C_3 contains categories from all over the category tree rather than specialising. C_5 and C_2 generally have too few entries to make substantiated claims, but books are clearly predominant. C_1 and C_4 both contain a lot of clothing and related items. While C_1 has some home appliances in it, C_4 has a tendency towards personal health products, including sports products.

Looking at Website popularity and quality, we found that technical implementation quality has no immediate bearing on cluster membership. Rather, we see that the number of sites from a certain cluster scale with the overall number of sites in the speed percentile. We further see that the distribution of sites from the clusters over the percentile bins follow no specific pattern. It can thus not be said that the speed of a Website has a positive correlation with its privacy-friendliness.

Less popular sites are found significantly more often in clusters that exhibit more leakage. More popular sites tend to leak less. For illustrative purposes, the average traffic rank is 0.4m for C_1, 1.0m for C_3 and 1.4m for C_4. A Mann-Whitney U test indicates a highly significant difference in the traffic ranks per cluster ($p = 0.001$ for both pairwise comparisons). Sites in the worst leakage C_5 do not appear among the 50 highest ranked in our sample (Fig. 3).

4.3 Third-party tracking facilitated by PayPal

Analysis of the HTTP traffic observed during the experiments revealed the use of Adobe's Omniture tracking software on PayPal checkout pages. When a user lands on the PayPal checkout page, two HTTP requests are sent to paypal.d1.sc.omtrdc.net and paypal.112.2o7.net subdomains, both of which belong to Adobe's Omniture tracking software [29]. The requests contain metadata about the payment to be made, such as currency and transaction token, along with the user's browser characteristics such as plugins, screen dimensions and software versions [30]. Remarkably, PayPal also shares the referrer URL of the checkout page, which reveals the URL of the Web shop, and potentially the product to be purchased. This leakage enables Adobe to build a better profile of 152 million PayPal users [53], by combining payment details with other online activities recorded on more than 300,000 Omniture-tracked Websites [37], which notably includes 50 of the Web shops analysed in this study.

Note that the leakage described here is different from the indirect information leakage via referrer headers as studied in [33], since the PayPal checkout page actively collects and sends the referrer of the checkout page, which would not be shared otherwise with the Omniture domains. Furthermore, by sending high-entropy browser properties such as plugins and screen dimensions, PayPal make it possible for Omniture to track users by their browser fingerprints even if they block or delete their cookies [14].

According to its privacy policy, PayPal may share customers' personal information with third-party service providers [52] who are limited to use PayPal customers' information "in connection with the services they perform for [PayPal]." Assuming the information shared with Omniture is subject to a similar agreement, it is hard to make sure whether payment information, product URL or browser characteristics are interpreted as personal information or not, given the possible interpretations of the policy and lack of transparency around PayPal's contracts with third-parties.

As of September 14th, 2014, long after we finished with the experiments, the PayPal checkout page no longer references a third-party tracker, though

Omniture is still used on the PayPal homepage (see Appendix for reproduction).

4.4 Flash evercookies and browser fingerprinting for internal tracking:

We also found the use of two questionable tracking mechanisms by PayPal internally, namely, evercookies and browser fingerprinting. Upon registering for a new account, PayPal places a Flash cookie named paypalLSO.sol, which includes a 70 character long identifier string. On each visit to PayPal checkout page, this Flash cookie is read by an invisible Flash object (mid.swf) and appended to the payment form as a hidden element.

Evercookies (also known as supercookies or zombie cookies) make use of obscure browser storage mechanisms to store tracking identifiers. Being a resilient tracking technology, evercookies can be used to restore standard (HTTP) cookies intentionally removed or blocked by users. In the past, use of such techniques led to lawsuits and multi-million dollar settlements [65].

Browser fingerprinting is another advanced tracking mechanism employed by PayPal. Through a field study, EFF's Panopticlick study showed that combining multiple browser properties, one can extract a fingerprinting that can be used to track users without relying on the stored identifiers such as cookies [43].

When a user visits the PayPal checkout page, a script (pa.js) collects multiple browser properties including browser plugins, screen dimensions and 36 different Windows ActiveX components to get information about the installed software and language packages. The Appendix lists the properties collected by PayPal's script. Unfortunately, PayPal's privacy policy is not as explicit about fingerprinting as it is for Flash cookies.

Despite using the same technology, the assessment of PayPal's user re-identification through persistent cookies has to be more favourable than third-party online tracking. In a payment scenario, these advanced tracking techniques can be used to prevent account hijacking or similar fraudulent activities. PayPal's privacy policy explains that "cookies, pixel tags, 'Flash cookies,' or other local storage [...]" are used to "recognize you as a customer; customize PayPal Services, content, and advertising; measure promotional effectiveness; [...] mitigate risk and prevent fraud [...]." [52]. Nonetheless, for a company that manages millions of payments each day, there are many improvements to make, beginning from preventing information leakage to third parties, being more transparent about in-house tracking mechanisms and strictly isolating the use of advanced tracking tools to combat fraud.

4.5 Responses received from the merchants

After the full data analysis was completed, we tried to contact a sample of 59 online retailers for which the proliferation of purchase details could be particularly embarrassing, including adult entertainment, pest control, shapewear, vitamins and medication. We used the same text across all shops in our request, and asked: "What data do you transfer to PayPal when a customer of your shop decides to use that payment option for a purchase?" We clearly mentioned "the context of a scientific study on online payment providers".

Most retailers could be contacted by email (38 shops, 64%); for 17 shops (29%), we had to use an online contact form. Four shops (7%) provided no means of getting in touch. A single merchant replied the next day and explained that they "transfer the absolute minimal data that is required by Paypal" (sic!), which however was not consistent with our records. No other shops replied to our enquiry within four weeks (or any time after for that matter); twelve auto-replies were received but never followed by an actual response.

The overall lack of responsiveness from the retailers is bad news for consumers who are left alone with their privacy concerns. Furthermore, the only retailer who made an effort to reply seemed unaware of more far-reaching data proliferation. This might suggest that merchants need tools to identify data flows, and better guidance on how to implement a privacy-friendly checkout procedure. Our article aims to help with both issues.

5 Limitations

As outlined above, our sampling strategy combined Web shop URLs from different sources to cover both larger and smaller merchants. We expect our dataset to contain an equal distribution over more and less professional Websites, as well as more and less frequented ones. The Websites we analysed are sites that are actually visited by Web users, as the sampling was guided by browsing session logs.

This comes at the price of diversity of goods that are sold. It easily observed that there are more Web shops selling physical goods than there are commercial dating Websites, for instance, and our manual categorisation of Web shops reiterates this. As a result, our categories are not evenly distributed over the dataset at all. This makes statistically significant statements about privacy practices hard, if not impossible. The non-existence of association rules with high confidence is an immediate result of the skewed distribution of categories.

Further breaking down the manually assigned categories into a more fine-grained version was done in a semi-automated way. This is necessarily prone to errors, stemming from incomplete or faulty inferences from the information provided by the Amazon Product Advertising API. Naturally, this API can only deliver information based on Amazon's product range. Labelling our dataset using this information will again be biased: Our method for assigning labels to Websites discarded relatively infrequent labels. The biggest problem here is not primarily that we may not have enough shops per label in our dataset, but rather that the labels may be infrequent because Amazon has few goods from a certain category. Again, under a skewed distribution, statistical significance is difficult to obtain.

For obvious reasons, our data collection setup could not cover server-to-server communication, which, according to PayPal documentation [54], can be used by merchants to communicate with PayPal. Also, in our experiments we did not go beyond the PayPal checkout page to complete the payments. As a result, the data collected and leaked after the PayPal checkout page is not covered in our analysis.

6 Conclusion and discussion

We presented a new species in the zoo of online tracking systems: explicit leakage of personal information and detailed shopping habits from online merchants to payment providers. In contrast to the widely debated tracking of Web browsing, online shops make it impossible for their customers to avoid this proliferation of their data.

By mediating online payments between merchants and buyers, payment providers are in a position to access sensitive payment details that can be used to build a detailed profile of shopping habits. Being the most popular payment provider, PayPal learns how much money its 152 million customers are spending and where. These customers are identified by name, email and postal address and through their bank details. We have demonstrated that merchant Websites are unnecessarily forwarding product details to PayPal that give a detailed view on consumers' purchases.

According to the 881 sites studied in our analysis, 52% of the most popular US Web shops shared product names, item numbers and descriptions with PayPal. Besides the negative privacy impact, consumers whose data are proliferating could suffer from less favourable payment terms (e.g., unavailable payment methods of higher interest rates on consumer loans based on their purchase patterns). On the other hand, the remaining 388 sites did not share any purchase

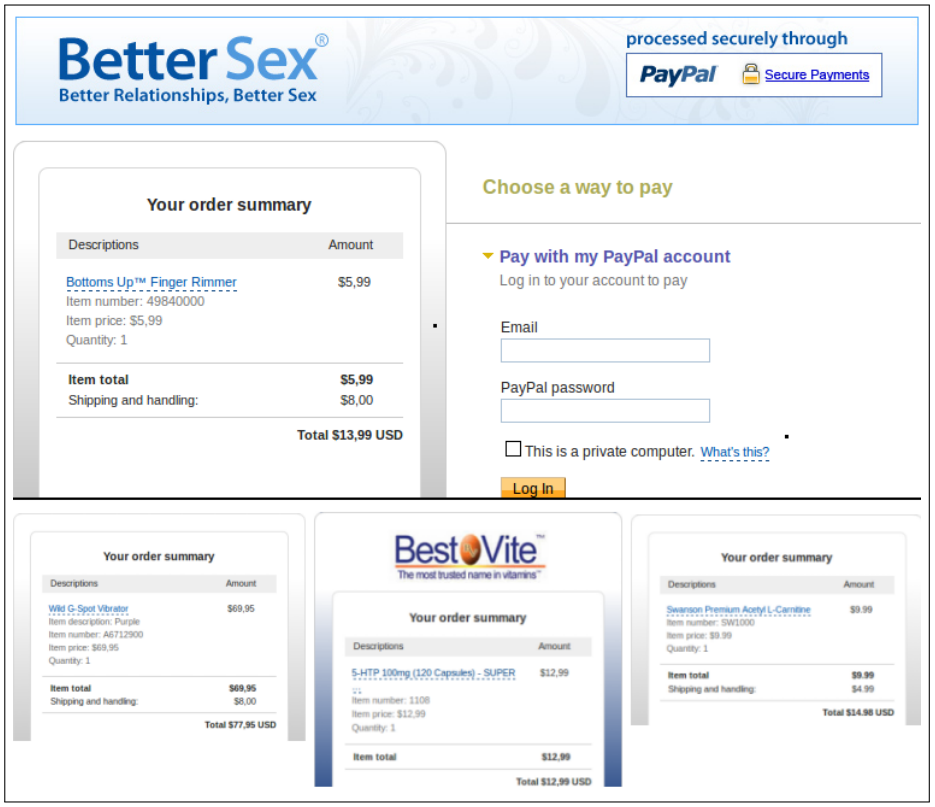


Figure 4: Also sites that sell sensitive products leak product details to PayPal. Two examples of adult toys (finger rimmer, vibrator) and medication: 5-HTP addresses depression, anxiety, sleep disorders, and MDMA hangover; acetyl carnitine is used for many indications including Alzheimer's disease and depression.

details except the amount to be paid, confirming that sharing sensitive details is not necessary for electronic retailers.

Further, we reported on the PayPal's use of the tracking service Omniture, which amplifies the privacy concerns by exposing transaction details to a widely deployed third-party tracker. A third-party tracker that has access to general Web tracking information, as well as to the details of successfully completed transactions, is in a particularly privileged situation to monitor consumption choices at large.

Web shops that use the technically more advanced token-based integration are

often more privacy-friendly. Also, less popular sites are significantly more often among those that leak more personal information. There are no systematic differences across product categories, meaning that all kinds of shoppers are exposed.

To the extent that PayPal, as an example of payment providers in general, collects personal information at scale, it becomes a constituent part of the online shopping experience: neither researchers nor enforcement authorities can reduce its role to a passive intermediary when assessing the privacy impact of e-commerce transactions.

By exploring the alternative privacy preserving practices that can be followed by Web shops, we distilled the following suggestions for merchants: (1) apply data minimization principle—do not leak information that is not required for processing the transaction; (2) inform customers about the data sharing in your privacy policy; (3) offer alternative, privacy-friendly payment methods, such as direct debit or pre-payment; (4) use a payment gateway to prevent leakage of product URL via referrer header.

Future research through qualitative interviews with decision-makers and engineers at merchants should look at the drivers and motives behind PayPal integration choices and their privacy consequences. On the technical side, expanding the scope to mobile and in-app payments promises valuable for these growing, yet opaque transactions. Better privacy practices for handling online payments is not only desirable for end users, but also for the merchants and payment providers whose bussinesses depend on the users' trust.

At times when personal information is said to be new currency on the Web, it seems unfair that consumers are charged twice during checkout.

References

[1] Gunes Acar, Christian Eubank, Steven Englehardt, Marc Juarez, Arvind Narayanan, and Claudia Diaz. The Web never forgets: Persistent tracking mechanisms in the wild. In *21st ACM Conference on Computer and Communications Security (CCS)*, pages 674–689. ACM, 2014.

[2] Gunes Acar, Marc Juarez, Nick Nikiforakis, Claudia Diaz, Seda Gürses, Frank Piessens, and Bart Preneel. FPDetective: Dusting the Web for fingerprinters. In *20th ACM Conference on Computer and Communications Security (CCS)*, pages 1129–1140. ACM, 2013.

[3] Inc. Amazon Web Services. Understanding browsenode results when drilling down. `http://docs.aws.amazon.com/AWSECommerceService/latest/DG/UnderstandingBrowseNodeResultsWhenDrillingDown.html`, 2013.

[4] Alapan Arnab and Andrew Hutchison. Using payment gateways to maintain privacy in secure electronic transactions. In *IFIP International Information Security Conference*, pages 277–288. Springer, 2007.

[5] M. Ayenson, D. J. Wambach, A. Soltani, and N. Good und C. J. Hoofnagle. *Flash Cookies and Privacy II: Now with HTML5 and ETag Respawning.* SSRN, 2011.

[6] Joseph P Bailey and Yannis Bakos. An exploratory study of the emerging role of electronic intermediaries. *International Journal of Electronic Commerce*, 1(3):7–20, 1997.

[7] Joseph Bonneau and Sören Preibusch. The privacy jungle: On the market for data protection in social networks. In *Eighth Workshop on the Economics of Information Security (WEIS 2009)*, pages 121–167, 2009.

[8] Joseph Bonneau and Sören Preibusch. The password thicket: Technical and market failures in human authentication on the web. In *Ninth Workshop on the Economics of Information Security (WEIS)*, 2010.

[9] Theodore Book and Dan S Wallach. A case of collusion: A study of the interface between ad libraries and their apps. In *Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM'13)*, 2013.

[10] European Commission. Proposal for a regulation of the european parliament and of the council on the protection of individuals with regard to the processing of personal data and on the free movement of such data (general data protection regulation), 2012.

[11] Aldo Cortesi. mitmproxy: a man-in-the-middle proxy. `http://mitmproxy.org/`.

[12] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[13] Charles Duhigg. How companies learn your secrets. *The New York Times*, 2012.

[14] P. Eckersley. How unique is your web browser? In *Proceedings of the 10th international conference on Privacy enhancing technologies (PETS*, 2010.

[15] Manuel Egele, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. Pios: Detecting privacy leaks in ios applications. In *NDSS*, pages 177–183, 2011.

[16] W. Enck, P. Gilbert, B. g. Chun, L. P. Cox, J. Jung, and P. McDaniel und A. N. Sheth. Taintdroid: an information flow tracking system for real-time privacy monitoring on smartphones. *Communications of the ACM, Bd.*, 57:99–106, 2014.

[17] Vladimir Filkov and Steven Skiena. Integrating microarray data by consensus clustering. *International Journal on Artificial Intelligence Tools*, 13(04):863–880, 2004.

[18] UK Financial Fraud Action. Fraud the facts 2013, 2013.

[19] Clint Gibler, Jonathan Crussell, Jeremy Erickson, and Hao Chen. Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale. In *International Conference on Trust and Trustworthy Computing*, pages 291–307. Springer, 2012.

[20] Google. Google wallet - shop in stores. `http://www.google.com/wallet/shop-in-stores/`, 2014.

[21] M. Hamblen. Starbucks invests Â£16m in us mobile payment venture. `http://www.computerworlduk.com/news/mobile-wireless/3374970/starbucks-invests-25m-mobile-payment-venture-in-us/`, August 2012.

[22] E van Heck and Peter Vervest. Web-based auctions: How should the chief information officer deal with them. *Communications of the ACM*, 41(6):99–100, 1998.

[23] Jochen Hipp, Ulrich Güntzer, and Gholamreza Nakhaeizadeh. Algorithms for association rule mining - a general survey and comparison. *ACM SIGKDD Explorations Newsletter*, 2(1):58–64, 2000.

[24] Donna L Hoffman, Thomas P Novak, and Marcos Peralta. Building consumer trust online. *Communications of the ACM*, 42(4):80–85, 1999.

[25] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. These aren't the droids you're looking for: Retrofitting Android to protect data from imperious applications. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 639–652. ACM, 2011.

[26] Information Commissioner's Office (ICO). Data controllers and data processors: what the difference is and what the governance implications are, 2014.

[27] Apple Inc. Apple - iphone 6 - apple pay. http://www.apple.com/iphone-6/apple-pay/, 2014.

[28] Sage Software Inc. Level 3 processing data. enhanced credit card processing, 2014.

[29] Adobe Systems Incorporated. Digital marketing | Adobe Marketing Cloud. http://www.adobe.com/solutions/digital-marketing.html, 2014.

[30] Adobe Systems Incorporated. SiteCatalyst variables and query string parameters. http://helpx.adobe.com/analytics/using/digitalpulse-debugger.html#id_1298, 2014.

[31] Nicola Jentzsch, Sören Preibusch, and Andreas Harasser. Study on monetising privacy: An economic model for pricing personal information. 2012.

[32] Klarna. Klarna checkout. https://klarna.com/sell-klarna/our-services/klarna-checkout, 2013.

[33] 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.

[34] Balachander Krishnamurthy and Craig E Wills. On the leakage of personally identifiable information via online social networks. In *Proceedings of the 2nd ACM Workshop on Online social networks (WOSN)*, pages 7–12. ACM, 2009.

[35] Pedro Giovanni Leon, Blase Ur, Yang Wang, Manya Sleeper, Rebecca Balebako, Richard Shay, Lujo Bauer, Mihai Christodorescu, and Lorrie Faith Cranor. What matters to users?: factors that affect users' willingness to share information with online advertisers. In *Proceedings of the Ninth Symposium on Usable Privacy and Security (SOUPS)*, page 7. ACM, 2013.

[36] A. Lewman. The team of paypal is a band of pigs and cads! https://lists.torproject.org/pipermail/tor-talk/2010-August/002978.html, 2010.

[37] BuiltWith Pty Ltd. Websites using Omniture SiteCatalyst. http://trends.builtwith.com/websitelist/Omniture-SiteCatalyst, 2014.

[38] Miguel Malheiros, Sören Preibusch, and M Angela Sasse. "fairly truthful": The impact of perceived effort, fairness, relevance, and sensitivity on personal data disclosure. In *International Conference on Trust and Trustworthy Computing*, pages 250–266. Springer, 2013.

[39] MasterCard. Mastercard corporate purchasing card implementation guide, 2001.

[40] Patrick McDaniel and Stephen McLaughlin. Security and privacy challenges in the smart grid. *IEEE Security and Privacy*, 7(3):75–77, 2009.

[41] 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.

[42] Microsoft. Wallet faq for windows phone | windows phone how-to (united states). `http://www.windowsphone.com/en-us/how-to/wp8/apps/wallet-faq`, 2014.

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

[44] OECD. The OECD Privacy Framework, 2013.

[45] Isle of Man Information Commissioner. Data protection act - data controller or data processor?, 2015.

[46] L. Olejnik and T. Minh-Dung und C. Castelluccia. *Selling off privacy at auction.* in NDSS, 2014.

[47] Jonathan W Palmer, Joseph P Bailey, and Samer Faraj. The role of intermediaries in the development of trust on the www: The use and prominence of trusted third parties and privacy statements. *Journal of Computer-Mediated Communication*, 5(3):0–0, 2000.

[48] PayPal. Encrypted website payments - technical overview. `https://www.paypal.com/us/cgi-bin/webscr?cmd=p/xcl/rec/ewp-techview-outside`, 2013.

[49] PayPal. Getting started with express checkout. `https://developer.paypal.com/webapps/developer/docs/classic/express-checkout/integration-guide/ECGettingStarted/`, 2013.

[50] PayPal. How would you like to integrate with paypal? `https://developer.paypal.com/webapps/developer/docs/`, 2013.

[51] PayPal. Paypal developer agreement. https://www.paypal.com/us/webapps/mpp/ua/xdeveloper-full, 2013.

[52] PayPal. Privacy policy. https://www.paypal.com/webapps/mpp/ua/privacy-full, 2013.

[53] PayPal. About paypal. https://www.paypal-media.com/about, 2014.

[54] PayPal. How would you like to integrate with PayPal? https://developer.paypal.com/docs/, 2014.

[55] PayPal. Legal agreements for paypal services. https://www.paypal.com/us/webapps/mpp/ua/legalhub-full, 2014.

[56] PayPal. REST API Reference - PayPal Developer. https://developer.paypal.com/docs/api/, 2014.

[57] PayPal. SetExpressCheckout API Operation (NVP). https://developer.paypal.com/docs/classic/api/merchant/SetExpressCheckout_API_Operation_NVP/, 2014.

[58] PayPal. Purchase protection - how to stay safe and sound with paypal. https://cms.paypal.com/cgi-bin/marketingweb?cmd=_render-content&content_ID=security/buyer_protection, 2015.

[59] K. Poulsen. Paypal freezes wikileaks account. http://www.wired.com/2010/12/paypal-wikileaks/, 2010.

[60] Sören Preibusch, Dorothea Kübler, and Alastair R Beresford. Price versus privacy: an experiment into the competitive advantage of collecting less personal information. *Electronic Commerce Research*, 13(4):423–455, 2013.

[61] Lee Rainie, Sara Kiesler, Ruogu Kang, Mary Madden, Maeve Duggan, Stephanie Brown, and Laura Dabbish. Anonymity, privacy, and security online. *Pew Research Center*, 5, 2013.

[62] Finextra Research. Disneyland paris to test contactless payments. http://www.finextra.com/news/fullstory.aspx?newsitemid=20321, July 2009.

[63] F. Roesner and T. Kohno und D. Wetherall. Detecting and defending against third-party tracking on the web. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI*, 2012.

[64] Leslie Scism. State farm is there: As you drive. *Wall Street Journal*, 2013.

[65] Ryan Singel. Online tracking firm settles suit over undeletable cookies. http://www.wired.com/2010/12/zombie-cookie-settlement/, 2010.

[66] A. Soltani, S. Canty, Q. Mayo, and L. Thomas und C. J. Hoofnagle. Flash cookies and privacy. Technical report, in Intelligent Information Privacy Management, Papers from the 2010 AAAI Spring Symposium, 2010.

[67] TRUSTe. Behavioral targeting: Not that bad?! truste survey shows decline in concern for behavioral targeting. http://www.truste.com/about-TRUSTe/press-room/news_truste_behavioral_targeting_survey, 2009.

[68] Janice Y Tsai, Serge Egelman, Lorrie Cranor, and Alessandro Acquisti. The effect of online privacy information on purchasing behavior: An experimental study. *Information Systems Research*, 22(2):254–268, 2011.

[69] S. Preibusch und J. Bonneau. The privacy landscape: product differentiation on data collection. In *Economics of Information Security and Privacy III*, pages 263–283. 2013.

[70] K. Gustafsson und N. Magnusson. Risk algorithm paves global expansion for klarna payment system. http://www.bloomberg.com/news/articles/2014-02-02/risk-algorithm-paves-global-expansion-for-klarna-payment-system, February 2014.

[71] Blase Ur, Pedro Giovanni Leon, Lorrie Faith Cranor, Richard Shay, and Yang Wang. Smart, useful, scary, creepy: perceptions of online behavioral advertising. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 4:1–4:15, New York, NY, USA, 2012. ACM.

[72] Jennifer Valentino-Devries and Jeremy Singer-Vine. They know what you're shopping for. *The Wall Street Journal*, 2012.

[73] Nicolas Viennot, Edward Garcia, and Jason Nieh. A measurement study of google play. In *ACM SIGMETRICS Performance Evaluation Review*, volume 42, pages 221–233. ACM, 2014.

[74] The Public Voice. The madrid privacy declaration: Global privacy standards for a global world. http://thepublicvoice.org/madrid-declaration/, 2009.

[75] WSJ Online. What They Know - Wsj.com. http://online.wsj.com/public/page/what-they-know-digital-privacy.html, 2013.

A Appendix

A.1 A sample HTTP request collected during the experiments

a) Request URL

https://paypal.d1.sc.omtrdc.net/b/ss/paypal-global/1/H.25.3/s6844
9009894746?AQB=1&ndh=1[trimmed – see below]

b) Request parameters (URL decoded)

AQB: 1

ndh: 1

t: 16/5/2014 3:37:43 1
-120

fid: 09476854BACDB25F-
2A6965C_4F340BABA

vmt: 51437A79

vmf: paypal.112.2o7.net

ce: UTF-8

ns: paypal

pageName: main:ec:::start

g: https://www.paypal.com
/cgi-bin/webscr?cmd=
_express-checkout&token=
EC-81F4649270038960T

r: https://www.poolparts
online.com/ShoppingCart
.aspx?add=true&ReturnUrl=/p-
59779-covers-umbrella
-furniture.aspx

cc: USD

ch: ec

server: main

products: ;ec

c1: xpt/Checkout/ec/Login

c5: 2P234932RV746033J

c7: none

v7: none:none:none

c8: none

c9: none

c17: Pay with a PayPal
account - PayPal

c19: main:ec:::start

v19: D=c7

c20: 1402882661

c21: EC-81F4649270038960T

c25: main:ec:::start:member:1:

v25: main:ec:::start:member:1:

v28: tnc-a-ecg-cntl

c30: glb

v31: main:ec:::start

c35: out

c36: paypal.com/cgi-bin/
webscr?cmd=_
express-checkout

v36: US

c37:member:1:

c39:D=pageName

c40:c97f8013a3fba

c47:D=pageName

c50:en_us

v50:0nehgENrmdgxT5WkliOJ
GPcyFf6%2bLQoeXAFWsgetQy
DTdJbVzxWcfz6BH%2bVCL
RtF1d9zERVjOXU%3d_146a25
299b6

c53:h.25.3|01.17.2013

c56:yes

c64:2294ec411f0df

c72:UTF-8

h1:main_ec__

s:1024x768

c:8

```
j:1.8.5

v:N

k:Y

bw:1024

bh:613

p:Shockwave Flash;iTunes
Application Detector;
QuickTime Plug-in
7.6.6;VLC Multimedia
Plugin (compatible

Totem 3.0.1);Windows
Media Player Plug-in
10 (compatible;
Totem);DivX©Web Player;

AQE:1
```

Listing 1. The request parameters sent to Adobe's Omniture tracking suite domain (omtrdc.net) leaks the product name and ID as a part of the referrer URL (parameter `r`). The plugin details, screen dimensions, browser window size and software versions are collected and sent to the tracking endpoint.

A.2 Browser properties collected by the PayPal analytics script pa.js

Script URL: https://www.paypalobjects.com/pa/js/pa.js

- User Agent string (navigator.userAgent)
- File name, description and version of installed browser plugins
- Browser plug-ins (navigator.plugins)
- All content types that the browser can handle (navigator.mimeTypes.type)
- Screen resolution and colour depth, browser window width & height
- JavaScript version
- Version numbers of 36 different Windows ActiveX components such as Outlook Express, MSN Messenger Service and Microsoft virtual machine

A.3 Data collection setup

We used a consistent setup to capture the information that merchants share with PayPal when their customers proceed to checkout.

Virtual machine: We used a clean virtual machine for each session in a best effort to prevent profiling by cookies or other client side data.

Proxy: We used mitmproxy [64] for intercepting and recording Web traffic. By adding mitmproxy's certificates to the browser, we recorded all HTTP(S)

requests and responses in decrypted form, including message bodies. We stored the network dump (.dmp) generated by mitmproxy to enable playback of the exact Web traffic. This helped us to ensure the reproducibility of our analysis.

Browser: We used a Firefox browser (version 25.0), configured to relay all communications through the intercepting proxy. The browser configuration was slightly modified from the default, as explained below; however, all privacy-related settings were left unchanged, simulating a default user.

Browser add-ons: Data collection was supported by browser add-ons for auto-filling forms and capturing the screen. 'Autofill Forms' (`https://addons.mozilla.org/en-us/firefox/addon/autofill-forms/`) helped us by quickly providing various information we are expected to fill in to forms on shopping sites. The use of the form-filler also ensured the same profile data was used on every site. We used 'Screengrab' (https://addons.mozilla.org/en-US/firefox/addon/screengrab-fix-version/) to take the screen capture of the PayPal page we were redirected to complete the payment.

Browser configuration: The browser configuration was kept closely to the default to mimic the privacy exposure of a mainstream user. We therefore kept all privacy-related settings unchanged and used the default non-private browsing mode. We continued to allow popups, Flash, Silverlight and Java (if available) and did not actively block script execution. We allowed and recorded requests to phishing/malware databases, which are enabled by the original settings.

At the same time, the following options were turned off to prevent cluttering of the recorded Web traffic: auto-update search engines, spell checking, crash report, Firefox health report and OCSP certificate verification.

Proxifying script: We used a Python script to launch mitmproxy and the browser with the product URL. The same script was used for parsing the network dump captured by mitmproxy and for outputting the captured HTTP(S) requests and responses for further analysis. The logs generated by the script were fed to a parser script that mined the data in the Web traffic and detected information flows.

A.4 Reproducing the original Omniture tracking on PayPal's checkout pages

Note that the PayPal checkout pages from the Internet Archive can be used to validate our findings about Omniture tracking:
`https://web.archive.org/web/20140228095312/https://www.paypal.com/cgi-bin/webscr?cmd=_flow&SESSION=_4T7tr0uKMzmNjcRwv_KSFh0Dminf5Q`

```
m11xcKqg33aOA_Q80mcRJXTVjTxK&dispatch=50a222a57771920b6a3d7b60623
9e4d529b525e0b7e69bf0224adecfb0124e9b61f737ba21b0819827f0298a8d83
82cff5df9729c4c3c2b2
```

Bibliography

[1] Acar, G. Position paper: Obfuscation for and against device fingerprinting. In *Symposium on Obfuscation* (New York City NY USA, 2014), p. 5.

[2] Acar, G., Alsenoy, B. V., Piessens, F., Diaz, C., and Preneel, B. Facebook Tracking Through Social Plug-ins. Technical report prepared for the Belgian Privacy Commission, 2015.

[3] Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., and Diaz, C. The Web never forgets: Persistent tracking mechanisms in the wild. In *21st ACM Conference on Computer and Communications Security (CCS)* (2014), ACM, pp. 674–689.

[4] Acar, G., Juarez, M., Nikiforakis, N., Diaz, C., Gürses, S., Piessens, F., and Preneel, B. FPDetective: Dusting the Web for fingerprinters. In *20th ACM Conference on Computer and Communications Security (CCS)* (2013), ACM, pp. 1129–1140.

[5] Juarez, M., Afroz, S., Acar, G., Diaz, C., and Greenstadt, R. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014), ACM, pp. 263–274.

[6] Nikiforakis, N., and Acar, G. Browse at your own risk. *IEEE Spectrum 51*, 8 (2014), 30–35.

[7] Olejnik, L., Acar, G., Castelluccia, C., and Diaz, C. The leaking battery - A privacy analysis of the HTML5 battery status API. In *Data Privacy Management, and Security Assurance - 10th International Workshop, DPM 2015, and 4th International Workshop, QASA 2015, Vienna, Austria, September 21-22, 2015. Revised Selected Papers* (2015), pp. 254–263.

[8] Preibusch, S., Peetz, T., Acar, G., and Berendt, B. Purchase details leaked to PayPal. In *Financial Cryptography and Data Security - 19th International Conference, FC 2015* (Puerto Rico, 2015), Lecture Notes in Computer Science, Springer-Verlag, p. 10.

[9] Preibusch, S., Peetz, T., Acar, G., and Berendt, B. Shopping for privacy: Purchase details leaked to paypal. *Electronic Commerce Research and Applications 15* (2016), 52–64.

[10] Vanrykel, E., Acar, G., Herrmann, M., and Diaz, C. Exploiting Unencrypted Mobile Application Traffic for Surveillance. Technical report, 2016.

[11] Vanrykel, E., Acar, G., Herrmann, M., and Diaz, C. Leaky Birds: Exploiting Mobile Application Traffic for Surveillance. In *Financial Cryptography and Data Security - 20th International Conference, FC 2016* (2016), Lecture Notes in Computer Science, Springer-Verlag, pp. 1–22.

Curriculum

Gunes Acar was born in Istanbul and studied Electrical and Electronics Engineering at Middle East Technical University. After working for five years as a programmer in the industry, he started working as a research assistant at the Ankara University. In 2007, he obtained his masters degree in Linguistics from Ankara University.

He joined COSIC in 2012 as pre-doctoral student. In 2013, he started his doctoral programme under the supervision of Claudia Diaz.

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF ELECTRICAL ENGINEERING
COSIC
Kasteelpark Arenberg 10, bus 2452
B-3001 Leuven
{gunes.acar,claudia.diaz}@esat.kuleuven.be
https://securewww.esat.kuleuven.be/cosic/